UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA

ESCUELA UNIVERSITARIA

DE

INGENIERIA TECNICA DE TELECOMUNICACION

TRABAJO FIN DE CARRERA

IMPLEMENTACION DEL PROTOCOLO LAP-D EN EL MICROCONTROLADOR DE COMUNICACIONES MOTOROLA 68302

AUTOR: SANTIAGO MARTÍN ROMANÍ

ESPECIALIDAD: TELEFONIA Y TRANSMISION DE DATOS

TUTOR: Dr. D. JUAN DOMINGO SANDOVAL GONZALEZ

CO-TUTOR: D. ALFONSO MEDINA ESCUELA

SEPTIEMBRE 1995

UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA

ESCUELA UNIVERSITARIA

DE

INGENIERIA TECNICA DE TELECOMUNICACION

TRABAJO FIN DE CARRERA

IMPLEMENTACION DEL PROTOCOLO LAP-D EN EL MICROCONTROLADOR DE COMUNICACIONES MOTOROLA 68302

PRESIDENTE **SECRETARIO** **VOCAL**

TUTOR **COTUTOR**

AUTOR **NOTA**

Agradecimiento y dedicatoria

Desde aquí quiero agradecer de forma muy especial a Juan Domingo Sandoval su dedicación, su tiempo y su amabilidad... ¡Gracias Kiko!

Y quisiera dedicar este trabajo a todos mis amigos:
Marta, Víctor, Javi, Fernando, David, Manolo...
y a mi familia.

El Autor
Santiago Martín Romaní

Septiembre de 1995

Capítulo 1: Introducción a la Red Digital de Servicios Integrados

Capítulo 2: Descripción global del Trabajo Fin de Carrera

Capítulo 3: Descripción Normativa

Capítulo 4: Descripción Entorno Global

Capítulo 5: Implementación

Capítulo 6: Entorno de Pruebas en MS-DOS. Explicación del software

Capítulo 7: Entorno de Pruebas en Sistema Empotrado. Explicación del software

Capítulo 8: Banco de Pruebas

Anexos

Bibliografía

Entorno de Desarrollo

Capítulo 1: Introducción a la Red Digital de Servicios Integrados

1.1 INTRODUCCIÓN

La red telefónica que todos conocemos y que, prácticamente, apenas ha variado en lo relativo a filosofía de funcionamiento y prestaciones desde lo que se bautizó como la "era de la automatización del servicio", está sufriendo en los últimos años una muy importante transformación merced a la introducción en la misma de las técnicas digitales en la transmisión y conmutación, que, al presentar una mayor eficacia y capacidad de tratamiento de información, están revolucionando, más que modificando, progresivamente los cimientos más básicos de lo que hasta la fecha se ha denominado como "telefonía".

La primera etapa de esta progresiva introducción, ha sido la digitalización de los nodos o elementos de conmutación de la red y de las arterias de transmisión, permaneciendo como único elemento todavía analógico el acceso de abonado, incluyendo bucle y equipo terminal. Es lo que se denomina Red Digital Integrada (RDI), y que como notable aportación plantea ya, aunque sea únicamente entre nodos de red, un tratamiento de la información codificada en forma binaria, y estructurada a una velocidad de 64 Kb/s, independientemente del tipo (voz o datos) y forma en la que se ha generado (teléfono u otros terminales).

De esta manera se obtiene una vía de conexión digital entre el usuario y la red y, por fin, entre los usuarios extremos de cada comunicación, por la que potencialmente se podría ofrecer una gran diversidad de servicios, entre los cuales estarían la gran mayoría de los que actualmente se ofrecen de una manera dispersa a través de las redes existentes (Red Telefónica Analógica, Red Iberpac, Ibermic...)

El tratamiento único e integrado de todos estos servicios en una única red, precisa de un muy importante intercambio de información de control o señalización entre los usuarios y la red, y entre los nodos de red. Se hace necesaria, pues, la introducción de un mecanismo de señalización potente, capaz de soportarlo, para lo que se ha elegido el Sistema de Señalización por Canal Común n°7 del CCITT, que dentro de la red hace uso de vías separadas de las propias de transferencia de información, lo cual confiere al sistema en su conjunto, una inusitada flexibilidad y potencia.

El usuario, en su acceso más elemental con la red, tiene dos vías o canales a 64 Kb/s, independiente una de la otra, aunque de uso simultáneo, y una vía para diálogo o señalización con la red, a 16 Kb/s, que eventualmente puede ser utilizada para transferencia de información de usuario.

Estos elementos, estructura de red, conectividad digital, sistema de señalización como el descrito, disponibilidad de diversidad de servicios y acceso de usuario único y multicanal, conforman la estructura de la Red Digital de Servicios Integrados, que no es sino la realización en donde confluyen los procesos de evolución de la hasta ahora Red Telefónica Básica.

En la figura 1.1 podemos observar gráficamente una posible configuración de la RDSI en la que se remarca la característica diferenciadora con la Red Telefónica Conmutada convencial: la conectividad digital extremo a extremo. También podemos observar la posibilidad de interconexión de la RDSI con la Red Telefónica Básica y con la Red IBERPAC. En el apartado 1.2.6 se describe este aspecto con más detalle.

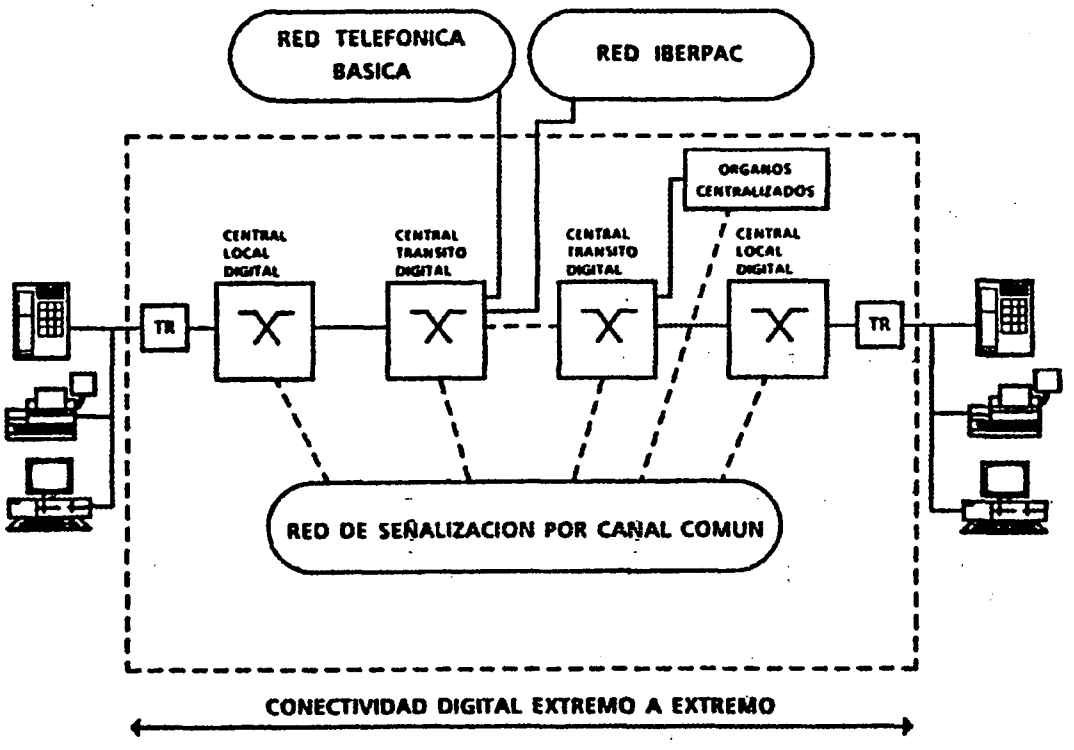


Figura 1.1 Configuración de la RDSI

1.2 CONCEPTOS BASICOS

1.2.1 Definición de RDSI

El CCITT (Comité Consultivo Internacional Telegrafico y Telefonico), la define como:

"Red que procede por evolución de la Red Digital Integrada y que facilita conexiones digitales extremo a extremo para proporcionar una amplia gama de servicios, tanto de voz como de otros tipos, y a la que los usuarios acceden a través de un conjunto definido de interfaces normalizados".

Es, pues, una red que procede por evolución, de la red telefónica existente y por tanto está basada en "*conexiones por conmutación de circuitos a 64 Kb/s* ", si bien en un futuro próximo incorporará elementos de conmutación de paquetes, para así convertirse paulatinamente en la "*red única*" que integre los servicios actuales.

Mientras dure este proceso de evolución, la RDSI coexistirá con las redes convencionales de telefonía y datos, pero incorporando elementos de

interfuncionamiento para su interconexión con dichas redes.

La RDSI considerada hasta ahora, se denomina de Banda Estrecha (RDSI-BE) puesto que trabaja con conexiones conmutadas a 64 Kb/s, pudiéndose llegar en un futuro próximo con la misma tecnología, hasta los 2 Mb/s.

A más largo plazo, la RDSI incorporará elementos conmutadores a velocidades superiores, lo que propiciará la aparición de otra nueva generación de servicios, (como por ejemplo distribución de programas de TV, videotelefonía de alta calidad o transmisión de datos a muy alta velocidad), tendremos así la RDSI de Banda Ancha (RDSI-BA). A partir de ahora, y siempre que No Se Especifique Lo Contrario, Estaremos Haciendo Referencia A La RDSI De Banda Estrecha

1.2.2 El acceso de abonado a la RDSI

En la figura siguiente se muestra la configuración de referencia del acceso de usuario a la RDSI, definida por el CCITT, en la que se señala el conjunto de *"puntos de referencia"* y de "agrupaciones funcionales" que se han considerado dentro del proceso de normalización de estos acceso.

Los **"punto de referencia"** son las separaciones entre funcionalidades distintas, identificadas en las agrupaciones funcionales, y pueden representar interfaces reales o virtuales.

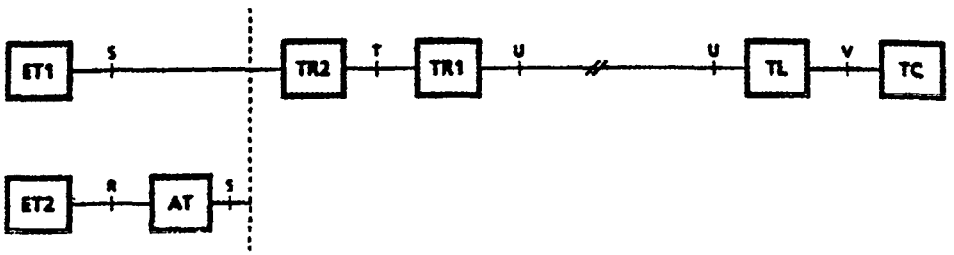


Figura 1.2 Configuración de referencia del acceso de usuario

Se han definido los siguientes:

- Punto de Referencia S:

Representa el interfaz usuario-red y constituye el punto de conexión física de los

terminales de abonado a la RDSI. Es un interfaz a cuatro hilos, dos para transmisión y dos para recepción.

- Punto de Referencia T:
Representa la separación entre las instalaciones del usuario y los equipos de transmisión de línea. Posee las mismas características mecánicas y eléctricas que el interfaz S.

- Punto de Referencia U:
Define la línea de transmisión entre las dependencias del abonado y la central telefónica, y se corresponde físicamente con el bucle de abonado analógico a dos hilos actualmente existente.

- Punto de Referencia V:
Representa la separación entre los elementos de transmisión y los de conmutación dentro de la central local RDSI.

- Punto de Referencia R:
Es el punto de conexión de cualquier terminal que soporte un interfaz normalizado, por ejemplo, terminales modo paquete X25, terminales con interfaz V.24 o terminales con interfaz analógico a dos hilos.

De esta manera y mediante el acoplo del adaptador correspondiente, se pueden conectar los terminales correspondientes al entorno de la RDSI.

Las **"agrupaciones funcionales"** que aparecen en la configuración de referencia, representan distintos niveles de funciones entrelazadas mediante el interfaz que representa cada punto de referencia, y pueden corresponder a un equipo físico o simplemente a una parte o función del mismo. Tienen el siguiente significado:

- ET1 (Equipo Terminal 1):
Es el equipo terminal que soporta directamente el interfaz S, es decir, está diseñado para conectarse directamente a la RDSI. Como ejemplo pueden citarse los teléfonos digitales RDSI, videotex RDSI, Fax Grupo 4 con interfaz S, etc.

Actualmente se encuentra en fase de desarrollo una gama muy amplia de

terminales que se conectan directamente a la RDSI.

- ET2 (Equipo Terminal 2):

Representa cualquier terminal que no soporta directamente el interfaz usuario-red de la RDSI, como los terminales modo paquete, terminales analógicos a dos hilos, terminales V.24, etc.

- AT (Adaptador de Terminal):

Este equipo soporta por un lado un interfaz R determinado y por el otro el interfaz S del acceso de usuario, y permite, por tanto, la conexión de los ET2 comentados anteriormente, a la RDSI. Como ejemplo, se pueden citar el Adaptador analógico AT a/b, para terminales analógicos a dos hilos y el Adaptador AT X.25 para terminales modo paquete.

- TR1 (Terminación de Red 1):

Supone la separación física entre las instalaciones de usuario y la red exterior, y realza la funciones de transmisión entre éstas y la central.

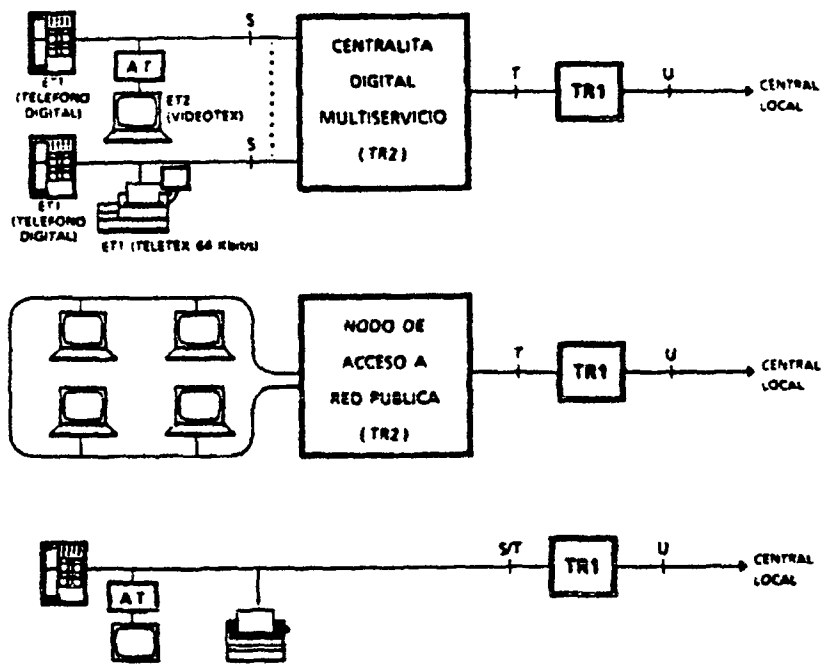


Figura 1.3 Diferentes configuraciones de Usuario

- TR2 (Terminación de Red 2):

Realiza funciones de conmutación y control en el interior de las instalaciones de abonado. La TR2 podría ser una centralita, una red de área local o un pequeño sistema multilínea, dependiendo del tamaño de dichas instalaciones. En el caso más sencillo, podría llegar a desaparecer, coincidiendo entonces físicamente los interfaces S y T.

- TL (Terminación de Línea):

Es el equipo de transmisión situado en el lado de la central local.

- TC (Terminación de Central):

Representa la separación entre los equipos de transmisión de línea y los equipos de conmutación en la central.

1.2.3 Canales de acceso

Para la transferencia de información y señalización se han definido en la RDSI los siguientes tipos de canales o vías de transferencia de información:

- Canal B:

es un canal a 64 Kb/s que transporta la información generada por el terminal del usuario.

- Canal D:

es un canal a 16 Kb/s ó 64 Kb/s, dependiendo de la estructura de acceso del abonado, que se utiliza para transportar la señalización en el interfaz usuario-red. También puede utilizarse para transmitir información de usuario a baja velocidad.

- Canal H:

Permite la transferencia de información de usuario a velocidades superiores a 64 Kb/s.

Existen tres modos de canal H:

- Canal Ho:

De 384 Kbps: Equivale a 6 canales de 64 Kb/s.

- Canal H11:
De 1536 Kb/s, equivale a 24 canales de 64 Kb/s, utilizable en países con jerarquía digital a 1544 Kb/s (USA y Japón).

-Canal H12:
De 1920 Kbps equivale a 30 canales de 64 Kb/s, utilizable en países con jerarquía digital a 2 Mb/s.

1.2.4 Estructuras del acceso de abonado

Los canales de información y señalización podrían combinarse para formar diversos tipos de acceso de usuario. Por el momento se han normalizado dos estructuras de acceso diferentes, atendiendo al número y tipo de canales de información y señalización que contengan y se denominan ACCESO BASICO y ACCESO PRIMARIO.

- Acceso Básico:
Está constituido por dos canales B a 64 Kb/s para la transmisión de información, y un canal D a 16 Kb/s para la señalización de usuario.

En el lado de las Instalaciones de usuario, (Interfaz S/T), supone una velocidad de transmisión total de 192 Kb/s (B + B + D + Control + Sincronismo + Mantenimiento) y está soportado por una configuración a cuatro hilos, dos para transmisión y dos para recepción. Permite una conexión máxima de 8 terminales.

En el lado de red (Interfaz U), la velocidad en línea es de 160 Kb/s y la transmisión es full-dúplex con técnicas de cancelación de eco, utilizándose como soporte el bucle de abonado analógico existente.

-Acceso Primario:
Está constituido por 30 canales B y un canal D a 64 Kb/s, con una velocidad total de 2 Mb/s.

En el lado de las instalaciones de usuario aparece el "Interfaz T", que puede ser totalmente equivalente al S o presentar una estructura tipo 30B+D, dependiendo de su tamaño, y que se utiliza para unir la unidad funcionalTR1, interfaz de línea, con la

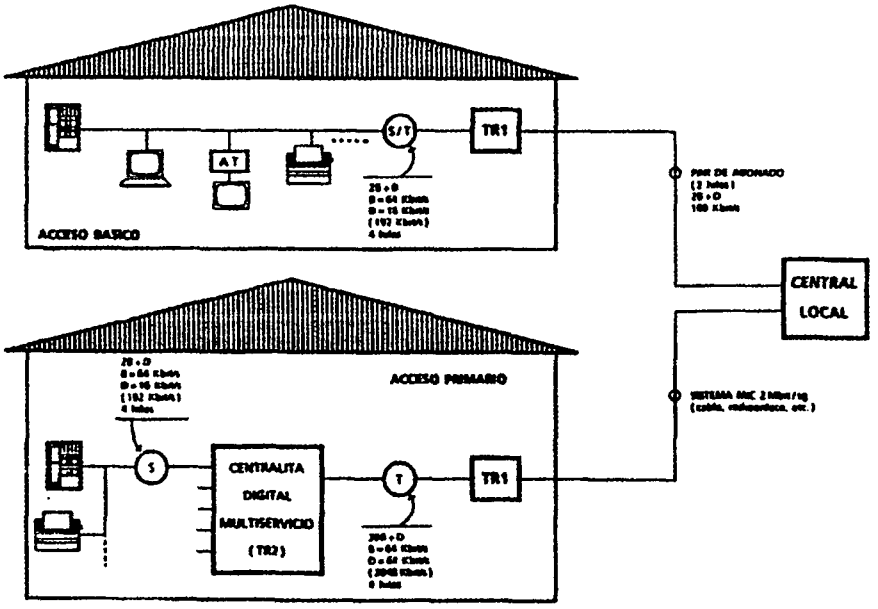


Figura 1.4 Estructuras de acceso de usuario

TR2, normalmente una centralita digital, cuyas extensiones pueden ser líneas de interfaz S.

En el lado de red, está soportado por un sistema de transmisión MIC a 2 Mb/s.

Este acceso primario puede soportar, y así está previsto, otras combinaciones de canales, aunque siempre respetando la velocidad global de 2048 Kb/s, como por ejemplo, 5Ho+D o H12+D, etc.

1.2.5 Red de tránsito

La red de tránsito de la RDSI está constituida por *"centrales de conmutación y sistemas de transmisión digitales"*.

Las centrales de conmutación realizan conexiones por conmutación de circuitos a 64 Kb/s y en un futuro próximo incorporarán elementos de conmutación de paquetes. Además soportan el sistema de señalización por canal común N°7 del CCITT.

Estas centrales presentan un alto grado de inteligencia, por lo que pueden suministrar facilidades adicionales, como mensajería electrónica o sistemas

avanzados de mantenimiento y operación de red.

Los sistemas de transmisión digital de la RDSI interconectan las centrales entre sí sirviendo de soporte a la conexiones y proporcionando la base para la continuidad digital extremo a extremo que define a esta red.

1.2.6 Interconexion de la RDSI con otras redes

- RDSI y RTC:

La RDSI va a estar conectada a la red telefónica convencional mediante unidades de interfuncionamiento, lo que va a permitir cursar tráfico entre ambas redes, en cualquiera de las direcciones.

La imbricación de las dos redes es hasta tal punto íntima, que cualquier abonado de la RDSI, por el hecho de serlo, tiene la consideración y tratamiento de la red telefónica convencional, consecuencia por otro lado lógica si se tiene en cuenta que la primera no es sino la evolución o transformación tecnológica de la segunda como ya se ha indicado.

-RDSI e IBERPAC:

Ambas redes se interconectan mediante Unidades de Adaptación de Red (UAR's), que presentan por un lado el interfaz X.25 de Iberpac, y por otro el interfaz S de la RDSI.

Los terminales modo paquete se conectan a la RDSI mediante Adaptadores tipo AT X25.

En estos momentos las centrales RDSI no disponen de manejadores de paquetes, por lo que la conmutación debe hacerse en la red Iberpac, proporcionando la RDSI un acceso transparente a la misma (Este entorno de utilización se conoce con el nombre de Escenario de Integración Mínima, en oposición al Escenario de Integración Máxima, que supone la disponibilidad de elementos de conmutación de paquetes en las centrales RDSI).

1.3 SERVICIOS OFRECIDOS

La RDSI va a permitir la introducción de una gama de nuevos servicios, basados en la capacidad de conexión digital a 64 Kb/s. Ello, va a poner a disposición de los usuarios un amplio repertorio de servicios suplementarios, que va a permitir entre otras cosas, por ejemplo, la identificación del usuario llamante ó la posibilidad de atender una llamada cuando la línea está ocupada. Sin embargo se dedicará un capítulo entero a los servicios ofrecidos por la RDSI, en el cual se especificará cada uno de ellos, por lo que ahora solo se nombrarán.

Dentro de la RDSI podemos considerar tres grandes grupos de servicios:

- Servicios portadores ó servicios de red
- Servicios finales ó teleservicios
- Servicios suplementarios

1.3.1 Servicios portadores

Estos servicios ofrecen al usuario, mediante una serie de interfaces normalizados, una capacidad de transporte de información independiente de su contenido y aplicación, pudiendo utilizar cualquier terminal que incorpore las capacidades requeridas para su comunicación.

Los servicios portadores prestados por la RDSI ofrecen la posibilidad de transferir información entre los puntos de referencia S y T de la configuración básica de usuario, y soportan únicamente los tres niveles inferiores del modelo de referencia ISA (Interconexión de Sistemas Abiertos).

A continuación, se van a enumerar los diferentes servicios portadores posibles en la RDSI, si bien algunos se encuentran todavía en fase de normalización. Podemos clasificarlos en dos grandes grupos:

Servicios Portadores Modo Circuito:

- Servicio Portador a 64 Kb/s sin Restricciones.
- Servicio Portador a 64 Kb/s estructurado a 8 Khz.
- Servicio Portador a 64 Kb/s estructurado a 8 Khz para información de audio.
- Transmisión alternada de conversación y 64 Kb/s sin restricciones.
- Servicios Portadores a n*64 Kb/s sin restricciones, estructurados a 8 Khz.

Servicios Portadores Modo Paquete:

- Servicio Portador de Llamada Virtual.
- Servicio portador de Circuito Virtual Permanente.

1.3.2 Servicios finales o teleservicios

Definen la capacidad de comunicación ofrecida al usuario RDSI en todos sus aspectos, tanto en lo referente a la conexión de red, como en organización y presentación. Por lo tanto, la prestación del servicio se recibe a través de un tipo determinado de terminal.

Los teleservicios disponibles incluyen los ya ofrecidos por las redes existentes (Red telefónica e Iberpac), así como una nueva familia basada en las posibilidades de las conexiones a 64 Kb/s. Estos teleservicios son:

- Telefonía
- Telefonía a 7 Khz
- Facsímil Grupos 2 y 3
- Facsímil Grupo 4
- Teletex
- Videotex
- Videotefonía
- Otros Teleservicios

1.3.3 Servicios suplementarios

El repertorio de servicios suplementarios disponibles en la RDSI es muy extenso, pudiendo seguir creciendo en el futuro a medida que las centrales RDSI vayan incorporando nuevas facilidades. Algunos están disponibles en las centralitas digitales existentes, o incluso en los equipos multilínea, otros sin embargo, constituyen una auténtica novedad y sólo son posibles debido a las posibilidades de señalización de red y de usuario que la RDSI incorpora

A continuación y a modo de ejemplo, se expone los siguientes:

- Grupo Cerrado de Usuarios
- Identificación del Usuario llamante
- Restricción de la Identificación del Usuario Llamante
- Identificación del usuario conectado
- Restricción de la Identificación de Usuario Conectado
- Indicación de Llamada en Espera

- Marcación Directa de Extensiones
- Múltiples Números de Abonado
- Subdireccionamiento
- Portabilidad de Terminales
- Señalización Usuario-Usuario
- Línea Directa sin Marcación
- Marcación Abreviada
- Conferencia a tres
- Desvío de Llamadas
- Transferencia de llamadas dentro del Bus Pasivo
- Llamada Completada sobre Abonado Ocupado
- Grupo de Captura
- Información de tarificación
- Prioridad de Llamadas
- Retención y Recuperación de Llamadas

1.4 TERMINALES QUE SE PUEDEN CONECTAR

La aparición de la RDSI va a propiciar la aparición de una nueva generación de terminal que soporta directamente el interfaz usuario-red (Interfaz S) y que aprovechan la capacidad transmisión a 64 Kb/s, independientemente de la naturaleza de la información a transmitir. Ello posibilita conectar terminales muy diferentes a una misma línea de acceso RDSI.

Como ya se ha indicado, los terminales existentes podrán conectarse a la RDSI mediante utilización de adaptadores adecuados, por lo que se pueden considerar dos grandes bloques terminales susceptibles de conectarse a una línea RDSI:

- Terminales existentes
- Terminales específicos para la RDSI.

1.4.1 -Terminales Existentes:

Los adaptadores disponibles para utilizar el parque de terminales conectados a redes existentes (RTC e Iberpac), en el ámbito de la RDSI, son los que indica la figura 1.5 .

- Adaptador Analógico "AT a/b":

Permite conectar a un acceso básico RDSI cualquier terminal que actualmente conecta a una línea analógica convencional de la RTC:

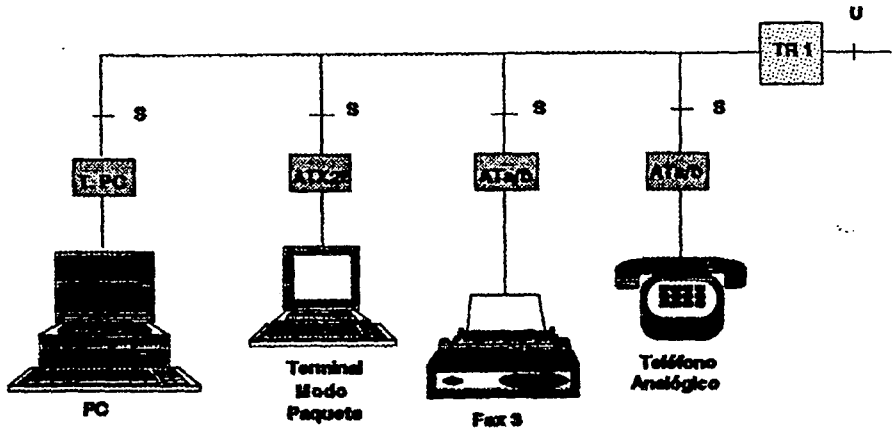


Figura 1.5 Tipos de adaptadores para terminales no RDSI

- Fax Grupos 2 y 3
- Videotex
- Modems de la serie V a dos hilos
- Teléfonos analógicos
- etc.

- Adaptador X.25 (AT X25):

Permite la conexión aun acceso básico RDSI de terminales modo paquete:

- Terminales con procedimientos de acceso a Iberpac por línea dedicada
- Terminales con procedimientos de acceso a Iberpac por línea conmutada a través de RTC (procedimientos X32).
- Terminales Teletex.

- Tarjeta Adaptadora para PC:

Permite la conexión de Ordenadores Personales compatibles a la RDSI, convirtiéndolos así en una potente estación de trabajo que puede servir de base para el desarrollo y soporte de numerosas aplicaciones, sobre todo en el campo de la recuperación de imágenes.

Además, estas tarjetas permiten la conexión de facilidades de voz, por lo que el PC se convierte así en un terminal integrado de voz y datos.(ver figura 1.6)

1.4.2 -Nuevos Terminales RDSI:

Estos terminales soportan directamente el interfaz usuario-red de la RDSI (interfaz S), y por tanto, se aprovechan de la ventajas de la transmisión de información a 64 Kb/s en lo referente a velocidad, fiabilidad y calidad.

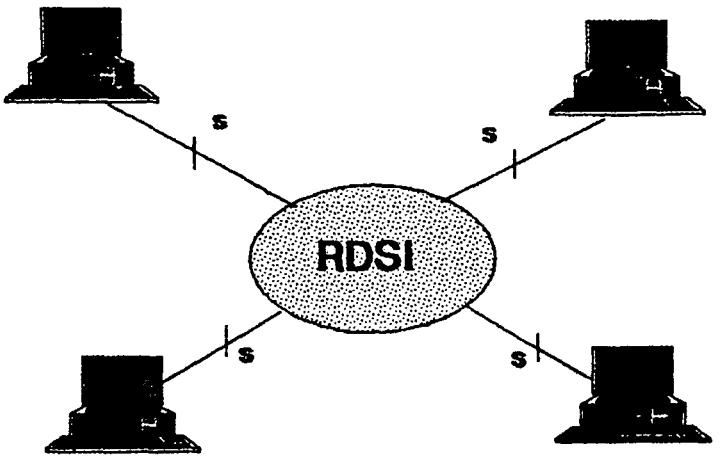


Figura 1.6 La RDSI y los ordenadores personales

Aquí se pueden considerar dos grandes apartados:

Terminales RDSI que son evolución de terminales existentes:

- Teléfonos Digitales RDSI
- Fax Grupo 4
- Videotex RDSI (alfageométrico)
- Teletex RDSI
- Videoteléfonos RDSI
- Faxtex RDSI

- Terminales RDSI especializados:

- Estaciones de trabajo avanzadas
- Dispositivos de monitorización, telemedida y telecontrol
- Terminales punto de venta integrados

Pueden considerarse también una serie de equipos, que, sin ser propiamente terminales, pueden jugar un papel muy importante en la configuración de *soluciones de abonado* basadas en RDSI:

- PBX's RDSI
- Pequeños sistemas de abonado RDSI (similares a los actuales sistemas multilínea)
- Puertas de acceso (Gateways) para la conexión en el entorno RDSI de Redes de Area Local (LAN s) y Grandes Ordenadores

1.5 Modelado de la RDSI

1.5.1 El modelo de referencia OSI

El suministro de servicios de comunicaciones es altamente variado a los usuarios bajo un acceso óptimo y unas condiciones de coste que hacen necesario tomar un cierto número de precauciones:

- La optimización de los intercambios entre terminales o máquinas, que son generalmente suministradas por diferentes fábricas, sin utilizar equipos de adaptación, que son siempre, complejos y caros;
- Independencia del servicio con la red de interconexión - una aplicación dada entre dos terminales debe ser capaz de funcionar y ofrecer al usuario la calidad requerida en el servicio (tiempo para establecer la comunicación, tiempo para transferencia de datos,...);
- Restricción de las modificaciones requeridas en el desarrollo del servicio para esas funciones directamente requeridas en dar este servicio.

Para satisfacer estas condiciones la ISO ha dividido el conjunto de funciones relativas a los servicios de comunicaciones en subconjuntos o "capas funcionales" en base del siguiente criterio funcional:

- *Homogeneidad de las funciones dentro de una capa.* Este concepto queda perfectamente definido a través de un ejemplo: es obvio que la detección y corrección de errores de transmisión son funciones relacionadas que no se relacionan con la sintaxis de la información que se intercambia entre los dos terminales.
- *Definición de capas, limitando su interacción tanto como sea posible.* La identificación de estos límites entre capas obedece a una estandarización de interface.
- *Restricción del número de capas funcionales a un valor razonable* para impedir que la descripción de los servicios se convierte en algo muy complejo.

El resultado de estos estudios, cubiertos por la Recomendación X.200 del CCITT, es un modelo con siete capas funcionales. Normalmente, se hace una

distinción entre capas inferiores (1-3) y superiores (4-7).

Las capas inferiores hacen referencia a las funciones necesarias para asegurar, con el funcionamiento requerido, la transferencia de información entre dos terminales a través de una red de telecomunicación. Estas capas inferiores se definen como sigue:

Capa 1 (capa física)

Maneja los aspectos físicos de la conexión de los terminales a las líneas de comunicación; interfaces eléctricas y mecánicas y protocolos de intercambio de elementos binarios.

Capa 2 (capa de enlace de datos)

Corresponde a la transferencia de información en las líneas de comunicaciones; conteniendo mecanismos de protección contra errores de transmisión.

Capa 3 (capa de red)

Asegura el establecimiento y desenganche de la comunicación, así como el ruteado de la información de usuario a través de la red, haciendo uso de conexiones establecidas en el itinerario elegido para conectar dos terminales.

Las capas superiores hacen referencia a funciones más específicas de las aplicaciones disponibles al usuario (fax, teletex, videotex,...). Estas funciones son manejadas por el equipo terminal, terminales o servidores y posiblemente por la propia red. Estas capas superiores del modelo OSI se definen como sigue:

Capa 4 (capa de transporte)

Suministra un seguimiento fin a fin de la transmisión de la información a través de la red. Es particularmente aplicable a aspectos como dirección del final, procedimiento de conexión, sincronización en los intercambios, posibilidades de error y control de flujo. Esta capa tiene el efecto de máscara desde el usuario.

Capa 5 (Capa de sesión)

Define la organización de los intercambios y la estructura de diálogo entre aplicaciones.

Capa 6 (Capa de aplicación)

Define la sintaxis de la información intercambiada (alfabeto, presentación de información gráfica en una pantalla,...). También incluye los mecanismos involucrados en la seguridad de acceso a la información en los servidores.

Capa 7 (Capa de aplicación)

Contiene los mecanismos comunes que pueden ser implementados por varios servicios (programas, bases de datos). El usuario accede a los servicios OSI por esta capa.

La representación de este modelo (ver figura 1.6) muestra la distinción entre procesos de aplicación local y procesos de aplicación de comunicación, siendo el último los elementos de usuario de la aplicación facilitada por la capa 7.

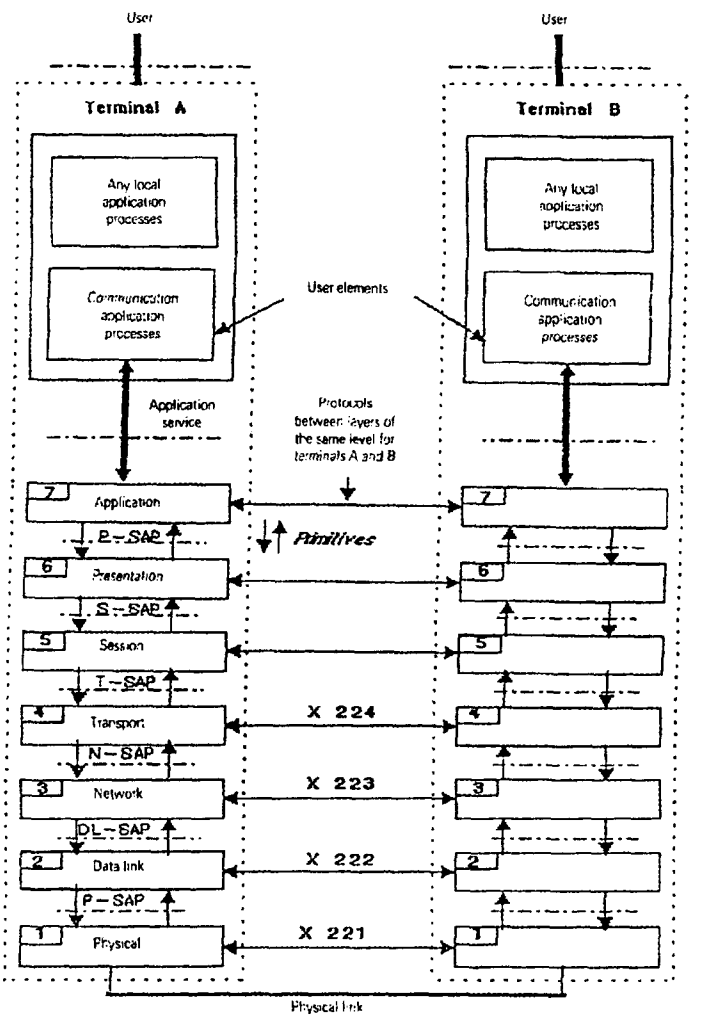


Figura 1.6 Modelo de Capas ISO

Los **puntos de acceso al servicio (SAPs)** identifican las interfaces entre capas adyacentes dentro del equipo: la interface entre la capa 1 y 2 se designa como PH-SAP (PH por física), la interface entre la capa 2 y 3 como DL-SAP (DL por enlace de datos), la interface entre la capa 3 y 4 como N-SAP (N por red) y así, sucesivamente.

Las **primitivas** constituyen la base para el diálogo entre capas adyacentes dentro del equipo. Hay cuatro tipos: petición, indicación, respuesta y confirmación. Cierta información y parámetros extras son añadidos a estas primitivas; por ejemplo, el número del abonado llamado está asociado con la primitiva de "petición de conexión". Estas primitivas están también asociadas a una capa N que pide un servicio dado desde la capa N-1 y lo ofrece a la capa N+1.

Los **protocolos** son reglas que definen el diálogo entre las capas del mismo nivel para dos terminales en comunicación. La especificación precisa de los protocolos asumen que hay una división detallada de funciones entre estos terminales.

La figura 1.7 ilustra la relación entre dos terminales de teletex durante la fase de comunicación en el caso de una transmisión de datos en modo paquete por una retransmisión que puede ser un centro de conmutación. El protocolo del nivel inferior (X.25 en la capa 3, protocolo de acceso al enlace tipo B (LAP B) en la capa 2, y X.21 o X.21 bis para la 1) son manejados aquí punto a punto. En realidad, el intercambio tiene que manejar estos protocolo para asegurar el establecimiento de las comunicaciones y el ruteado de la información a través de la red. En el caso considerado, los terminales tienen un diálogo directo entre cada uno del mismo nivel de las capas 4-7.

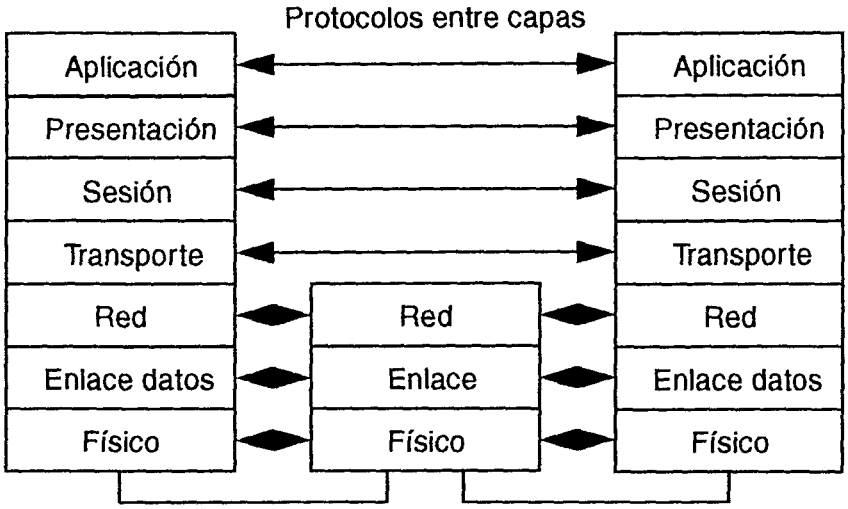


Figura 1.7 Ejemplo de aplicación de un modelo OSI

1.5.2 Método para describir los servicios de telecomunicación en la RDSI

La implementación de un servicio de telecomunicación por un usuario requiere tres elementos: una red, un terminal y reglas comerciales y de funcionamiento (tarifas, calidad del servicio,...). Éstas son especificadas por el operador de la red quien puede ofrecer al usuario dos tipos de servicio.

(1) Servicios de transmisión de información a diferentes velocidades y diferentes niveles de calidad de transmisión. Estos servicios ofrecidos por la capa 3 del modelo OSI se llaman **"servicios portadores"** en la RDSI. La provisión de un servicio portadores por el operador no está acompañada de una garantía de compatibilidad de los terminales en comunicación.

(2) "Teleservicios" para que se garantize la compatibilidad de los terminales de comunicación. Esto significa que terminales involucrados en un teleservicio dado deben utilizar los mismos protocolos para todas las capas 1-7. Como en el caso de los servicios portadores, el operador aplicará las reglas de funcionamiento y comercial para cada uno de los teleservicios ofrecidos.

La figura 1.8 muestra los puntos de acceso correspondiente a los dos tipos de servicio: un servicio portadores se ofrece en la interface a la red (RDSI) mientras que

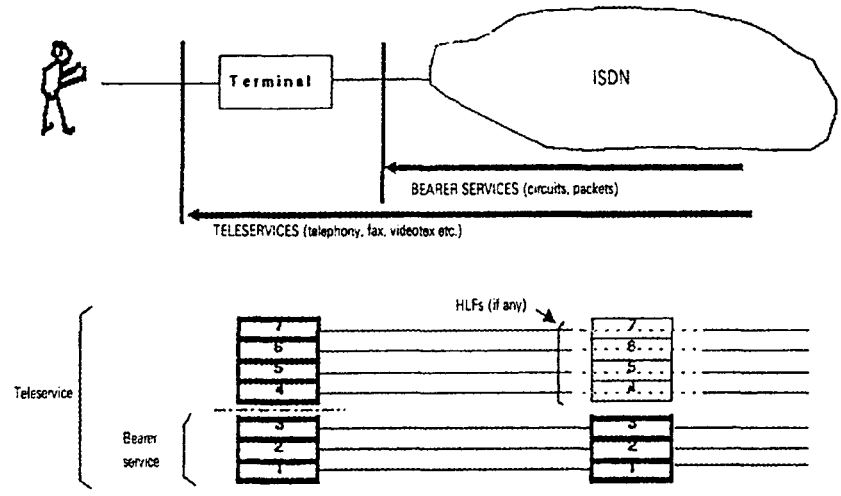


Figura 1.8 Servicios Portadores y Teleservicios en la RDSI

un teleservicio se realiza a través de la interface terminal-usuario. También, esta figura indica la localización de las funciones de las capas OSI en un terminal y en la red. Cuando el último interviene para suministrar un valor añadido (conversión de protocolo, conversión de velocidad,...), tiene que manejar todos o algunos de los protocolos de capa superior; se dice entonces que la red contiene *"funciones de alto nivel"* (HLFs). Estos conceptos de servicios *portador* y *teleservicios* no son nuevos: la transmisión de información en la banda de 300-3400 Hz en la red telefónica analógica es idéntica a un servicio portadores, y un servicio teletex que usa la red de conmutación de paquetes, puede considerarse como ejemplo de un teleservicio.

Desde que la RDSI se formó para ofrecer un rango lo más ancho posible en cuanto a servicios de teléfono y de datos (no sólo datos), es necesario definir todos estos servicios completamente y sin ambigüedad. Para esto, el CCITT tiene definido en un primer paso todos los atributos que hacen posible caracterizar los servicios portadores y teleservicios. La lista de estos atributos y sus definiciones están incluidas en la Recomendación I.130. La asignación de un valor a cada uno de los atributos hace posible describir un servicio portadores (o teleservicio). Este método de servicios descrito se llama "método del atributo".

En los *servicios portadores* se utilizan tres categorías de atributos:

- *Atributos de transmisión de información:* modo de transmisión (circuito, paquete,...), velocidad (64, 384, 1920,...Kbit/s), modo de establecimiento (demanda, reservado, permanente), simetría de la comunicación (bidireccional simétrica, unidireccional,...), configuración de la comunicación (punto a punto, emisión,...).
- *Atributos de acceso:* canal de transmisión utilizado, protocolo de señalización, protocolo de transmisión de datos (capas 1-3 del modelo OSI).
- *Atributos generales* (calidad del servicio, facilidad del servicio, tarifa,...)

Para los *teleservicios,* hay atributos que definen:

- Los protocolos de las capas superiores (capas 4-7 del modelo OSI),
- El tipo de información de usuario (voz, texto,...)
- Generales (calidad de servicio, facilidad del servicio, tarifa,...)

y son añadidos a los atributos anteriores para transmisión de la información y acceso.

La referencia explícita a las capas del modelo OSI en los atributos de los servicios portadores y teleservicios muestran que la RDSI no está totalmente inventada. El método de atributo sólo contribuye a los elementos adicionales necesarios para una descripción completa de los servicios ofrecidos al usuario.

1.6 Arquitectura RDSI

1.6.1 Arquitectura básica de la RDSI

La figura 1.9 representa el modelo básico de arquitectura de la RDSI, por ejemplo, el conjunto total de posibilidades funcionales para conmutación y señalización contiene también las relaciones de la RDSI con redes dedicadas.

Aunque, este modelo general podría parecer elemental expresa las principales características y potencial de la RDSI:

- Estandarización de la interface de usuario con la RDSI, que se designa por S/T;
- La universalidad de esta interface con respecto al rango de facilidades ofrecidas

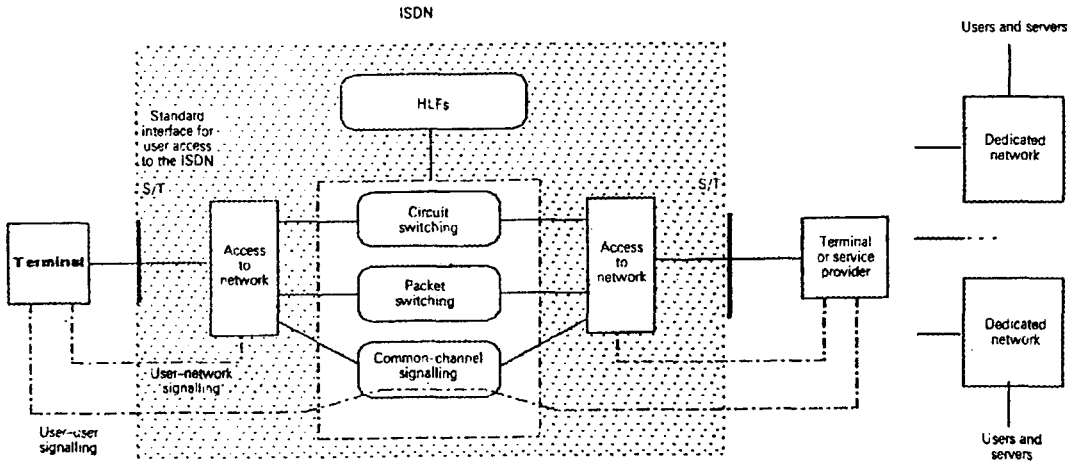


Figura 1.9 Modelo de la arquitectura básica de RDSI

por la RDSI con todos los recursos (transmisión, conmutación, señalización) públicos o privados, implementado en el establecimiento de una relación entre dos terminales RDSI.

Las diversas posibilidades de conmutación están clasificadas de acuerdo a tres criterios: velocidad, modo de conmutación (circuito, paquete), establecimiento de la comunicación (llamada por demanda, permanente, reservada). Estas corresponden a los servicios portadores de la RDSI.

Se ofrecen las siguientes señales de señalización:

- Entre el usuario y la RDSI para implementar los servicios portadores, los teleservicios y las facilidades asociadas al servicio;
- Dentro de la RDSI, por ejemplo entre los intercambios para rutear las llamadas;
- De usuario a usuario (la red es transparente con respecto a este tipo de señalización).

Las funciones de las capas superiores 4-7 necesarias para el suministro de teleservicios puede ser implementadas dependiendo del caso.

- Sólo en los terminales para teleservicios no se utilizan ninguna función extra de la RDSI, ni de un servidor ni de una red dedicada.

- En terminales RDSI y servidores, públicos y privados, conectados por la interface estándar S/T con la RDSI: el servidor está entonces incluidos en el suministro de "valor añadido".
- En los terminales y en la RDSI consigo mismo, que pueden ofrecer cierto valor añadido.
- En los terminales y servidores públicos y privados conectados a redes dedicadas: aquí nos encontramos con todos los factores de interconexión necesarios para establecer las relaciones entre los usuarios de RDSI y los usuarios del servicio.

1.6.2 Grupos funcionales y puntos de referencia

La figura 1.10 indica los diversos grupos funcionales de la RDSI.

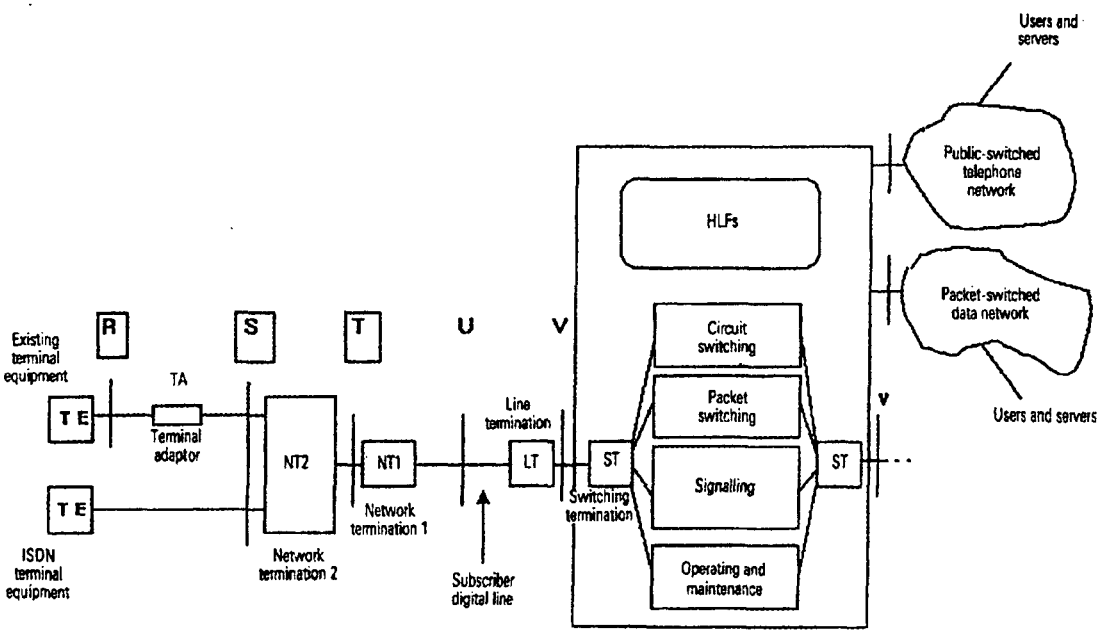


Figura 1.10 Agrupaciones Funcionales y Puntos de Referencia RDSI

Para no permitir ninguna confusión entre estos y los equipos físicos, la CCITT emplea la expresión "puntos *de referencia*" para designar los límites entre estos grupos. Esta precaución deja el camino abierto para múltiples configuraciones de equipo, en particular a nivel de instalaciones de usuario. Sin embargo, los equipos y entidades funcionales o interfaces y puntos de referencia pueden confundirse.

Por otro lado la figura 1.10 muestra lo siguiente.

- El *equipo terminal (TE)* accede a la red con un punto de referencia S si hay terminales RDSI, el acceso al punto de referencia S para terminales que existen satisfacen las recomendaciones de las series V y X de la CCITT (punto de referencia genérica R) por lo que requiere un adaptador de terminal R/S.
- *Terminación de Red 2 (NT2):* Los puntos de referencia de un NT2 son S en el lado del terminal y T en el lado de la red. El punto de referencia T está en ciertos casos en la línea de demarcación entre el dominio público y el privado. Las principales funciones de un NT2 son el control del acceso a la red pública.
- Equipos de *Terminación de Red 1 (NT1)* en el lado de instalación de usuario y *terminación de línea (LT)* en el lado de intercambio de línea al abonado: aparte de las funciones de transmisión que dependen del medio y técnica empleada, NT1 y LT también realizan funciones adicionales de protección, alimentación, monitorización de la calidad de transmisión,. La interface entre estas dos entidades es la interface U.
- *Funciones de señalización, conmutación y posiblemente de alto- nivel* (conversiones de protocolo, conversiones de velocidad,...) de la RDSI: en el lado del abonado el punto de referencia V marca el límite entre transmisión (LT) y conmutación (terminación de intercambio (ET)). En la práctica, estas funciones se dividen entre los intercambios locales, los intercambios interurbanos, los puntos de transmisión de señalización y los centros de mantenimiento y operación.
- Los *servidores de RDSI públicos y privados* (bases de datos, sistemas de manejo de mensajes) conectados de acuerdo con la interface estándar S/T para acceder a la RDSI.

La RDSI también suministra el acceso a redes ya existentes y a través de ellas a los usuarios y servidores.

1.7 Recomendaciones del CCITT para la RDSI

1.7.1 Presentación general de las Recomendaciones

Las Recomendaciones CCITT en la RDSI son un conjunto en las

Recomendaciones de las series I del grupo de estudio XVIII:

- Series I.100: General (plan de las Recomendaciones, terminología, métodos)
- Series I.200: Capacidad del servicio (servicios de soporte, teleservicios)
- Series I.300: Aspectos y funciones de red (principio funcional de la RDSI, modelos de referencia, direccionamiento, tipos de conexión, funcionamiento)
- Series I.400: Interface usuario-red en RDSI (capas 1-3 de las interfaces de base, multiplexaje, soporte de interfaces existentes)
- Series I.500: Interfaces entre red.
- Series I.600: Principios de mantenimiento.

Una parte significativa de las Recomendaciones de las Series I las podemos encontrar bajo otras referencias, como: X, Q,... Estas recomendaciones contienen un gran número de mecanismos de expansión progresiva de los servicios de RDSI, lo que implica, que no todas tienen el mismo grado de precisión y estabilidad.

1.7.2 Los puntos de referencia T y S

Los puntos de referencia T y S ocupan un lugar particularmente importante en las Recomendaciones del CCITT en RDSI. Estos puntos definen todas las características que determinan las posibilidades de RDSI en términos de servicios ofrecidos, como tipos de canales de transmisión, protocolos para acceso al usuario, etc,.

1.7.2.1 El punto de referencia T

Para encontrar los objetivos de desarrollo de la RDSI habrá que comenzar con las redes de distribución ya instaladas, la definición de las interfaces usuario-red tendrían en cuenta el funcionamiento de las técnicas de transmisión digital por una red local en términos de rango y velocidad. La elección final es el acceso al usuario llamado "acceso *básico*" con una velocidad de utilización de 144 Kbit/s. Además, para servir con una alta capacidad a las instalaciones del abonado, se selecciona un segundo acceso.

Es obvio que dependiendo de la capacidad de las instalaciones, se podría utilizar uno o más interfaces usuario-red a 144 Kbps (instalaciones pequeñas y medianas) y uno o más interfaces de usuario-red en 1984 Kbps ó 1536 Kbps (instalaciones de alta capacidad). Estos dos multiplexores están organizados en canales de dos tipos principalmente:

- Canal B a una velocidad de 64 Kbit/s: Se usaría para telefonía, datos por módem, fax, teletex modo mixto y videotex fotográfico.
- Canal D utilizado en todos los casos para señalizar al usuario- red pero también para el transporte de datos con velocidad bajo de naturaleza esporádica (señales de telecontrol, de telemedida, ordenador iteractivo, videotex,...)

La estructura del acceso básico queda entonces 2B+D para una velocidad total 144 Kbps, en este caso, la velocidad del canal D será 16 Kbps.

1.7.2.2 El punto de referencia S

En la definición de las características del punto de referencia S, una consideración que es de vital importancia ha sido reconocer que uno o más terminales serían capaces de conectarse por un acceso sencillo directamente a la red pública, por ejemplo, sin NT2. El interés de esta configuración es obvia para instalaciones privadas muy pequeñas. Bajo estas condiciones, los puntos de referencia S y T (acceso básico) son indistinguibles.

Las características del punto de referencia S en términos de velocidad y estructura de los canales útiles son por lo tanto idénticas al punto de referencia T (para el acceso básico): 144 Kbps y estructura 2B+D.

Si la variedad de terminales RDSI se tienen en cuenta, es obvio que un gran número de ellos no necesitan una velocidad de 144 Kbps. Esta fue la justificación para compartir este recurso total de 144 Kbps entre diversos terminales sin las restricciones de tráfico y la naturaleza de estos terminales.

Este diseño de la interface S también presenta la ventaja de una amplia cabida

para terminales multifunción. De hecho, como este recurso de 144 Kbps esta disponible, como un terminal que puede usarse dependiendo de la aplicación del momento, sólo el canal D para acceder a una base de datos de videotex (una aplicación de teleacción), un canal B para telefonía (una transferencia de fichero) o dos canales B para una aplicación multimedia.

La gran similitud entre S y T explica porque el CCITT pone las Recomendaciones que se aplican a estos dos puntos de referencia bajo la misma cabecera de "interfaces usuario-red":

- I.410: Principios generales
- I.411: Configuraciones de referencia
- I.412: Estructuras de interface y capacidades de acceso
- I.430: Capa 1 del acceso básico
- I.440/441: Capa 2 del acceso usuario-red, (LAP D)
- I.450/451: Capa 3 de la interface de usuario-red

Capítulo 2: Descripción global del Trabajo Fin de Carrera

2.1 GENERAL

El presente Trabajo Fin de Carrera consiste en la implementación del protocolo LAP-D en lenguaje C para el Microcontrolador de Comunicaciones Motorola 68302 (MC68302). Se implementaron cinco ficheros de código correspondientes a otros tantos procesos cuyas funciones están definidas de la forma Kernighan & Ritchie debido a que el compilador M68K necesario para el entorno del MC68302 sólo soportaba esta opción.

Los cinco ficheros o programas creados implementan la capa o nivel dos (Capa 2 o Capa de Enlace de Datos) de un Acceso Básico RDSI en el lado del Equipo Terminal y en el lado del Terminal de Red.

A partir de estos ficheros, se crearon y ejecutaron una serie de procesos bajo un sistema operativo multiproceso sobre un hardware digital basado en el MC68302 (implementación capa física). Estos ficheros o procesos implementan las entidades de Capa o Nivel 2 (Capa de Enlace de Datos) de los lados del Equipo Terminal y del Terminal de Red, las entidades de Gestión de Capa de los lados del Equipo Terminal y del Terminal de Red, y además, se creó un quinto fichero auxiliar que da origen a un proceso multiplexor o concentrador necesario en el nivel o capa 2 del lado del Terminal de Red.

Cada proceso se caracteriza por su contador de programa, sus variables

propias y las variables compartidas entre éste y otro(-s) proceso(-s).

También se crearon las funciones necesarias para probar la correcta implementación de los ficheros simulando las capas o niveles físico y capa tres (capa 3) en el entorno MS-DOS. Finalmente se probaron los procesos implementados en el sistema empotrado (hardware basado en el MC68302).

Los procesos creados se ayudan de un módulo software que llamamos kernel (núcleo). El módulo kernel constituye un conjunto de funciones que proporciona el control de procesos, de pipes (caminos o estructuras de comunicación tipo cola entre procesos), de temporizadores (o contadores de tiempo), de memoria dinámica y del reloj en tiempo real. También controla la inicialización del hardware.

2.2 DESCRIPCION DE LOS CINCO FICHEROS IMPLEMENTADOS

Para la implementación de la capa 2 del lado del Equipo Terminal se crearon los dos siguientes ficheros:

El fichero n2_et.c: Consiste en la implementación de una entidad de Capa de Enlace de Datos para el lado del Equipo Terminal.

El fichero ged_et.c: Consiste en la implementación de la entidad de Gestión de Capa de Capa de Enlace de Datos del lado del Equipo Terminal, que será quién, entre otras cosas, pida un Identificador de Equipo Terminal (IET) para establecer la conexión de enlace de datos punto a punto.

Para la implementación de la capa 2 del lado del Terminal de Red se crearon los tres siguientes ficheros:

El fichero n2_tr.c: Consiste en la implementación de una entidad de Capa de Enlace de Datos para el lado del Terminal de Red.

El fichero ged_tr.c: Consiste en la implementación de la entidad de Gestión de Capa de Enlace de Datos del lado del Terminal de Red. Para las dos entidades pares de Gestión de Capa de Enlace de Datos se desarrollan los procedimientos de gestión

de los identificadores de Equipo Terminal (en cada entidad de gestión de capa la parte que le corresponde). Estos procedimientos son los relacionados con la asignación, comprobación, verificación y supresión de un Identificador de Equipo Terminal (IET).

El fichero muxn2_tr.c: Consiste en la implementación de un proceso que llamaremos multiplexor o concentrador, y que será un proceso intermedio entre las entidades de Capa de Enlace de Datos del lado del Terminal de Red y la capa física (implementada por medio de determinadas funciones). Posteriormente se explicará detalladamente la necesidad de este proceso así como la existencia de varias procesos o entidades de Capa de Enlace de Datos en el lado del Terminal de Red.

2.3 IDEA ORIGINAL

Este trabajo Fin de Carrera nace de la idea de conseguir un bucle de abonado para el Acceso Básico RDSI (con la posibilidad de conectar hasta los ocho Equipos Terminales posibles) a partir de un bucle de abonado en configuración punto a punto.

Utilizar un bucle de abonado en configuración punto a punto tiene como ventaja el poder disponer de una distancia máxima de cableado de 1200 metros (determinada por la capacidad mutua del par (34 nF/Km) y la atenuación del cable a 96 KHz (5 dB/Km); el calibre del conductor será de 0.6 mm).

El gran inconveniente de esta configuración es la posibilidad de conectar un único Equipo Terminal al extremo del mismo. En la figura 2.1 podemos observar gráficamente esta situación.

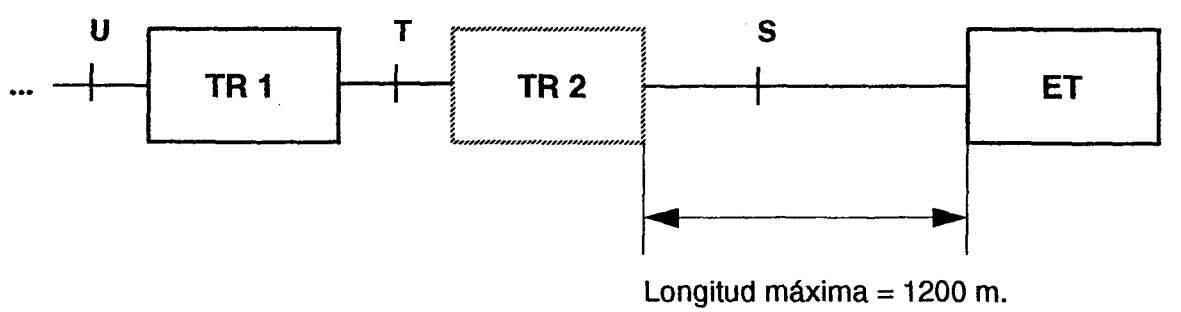


Figura 2.1 Bus pasivo en configuración punto a punto

Nuestra idea es crear un equipo que haga por un lado las funciones de Equipo Terminal (de cara al TR 2) y por el otro, las funciones de un nuevo Terminal de Red (TR). De esta forma podríamos conectar un nuevo bus pasivo al extremo del bus pasivo punto a punto.

Esta idea se observa gráficamente en la siguiente figura 2.2 , donde ya podemos apreciar la existencia de dos buses pasivos diferenciados (bus pasivo A y bus pasivo B).

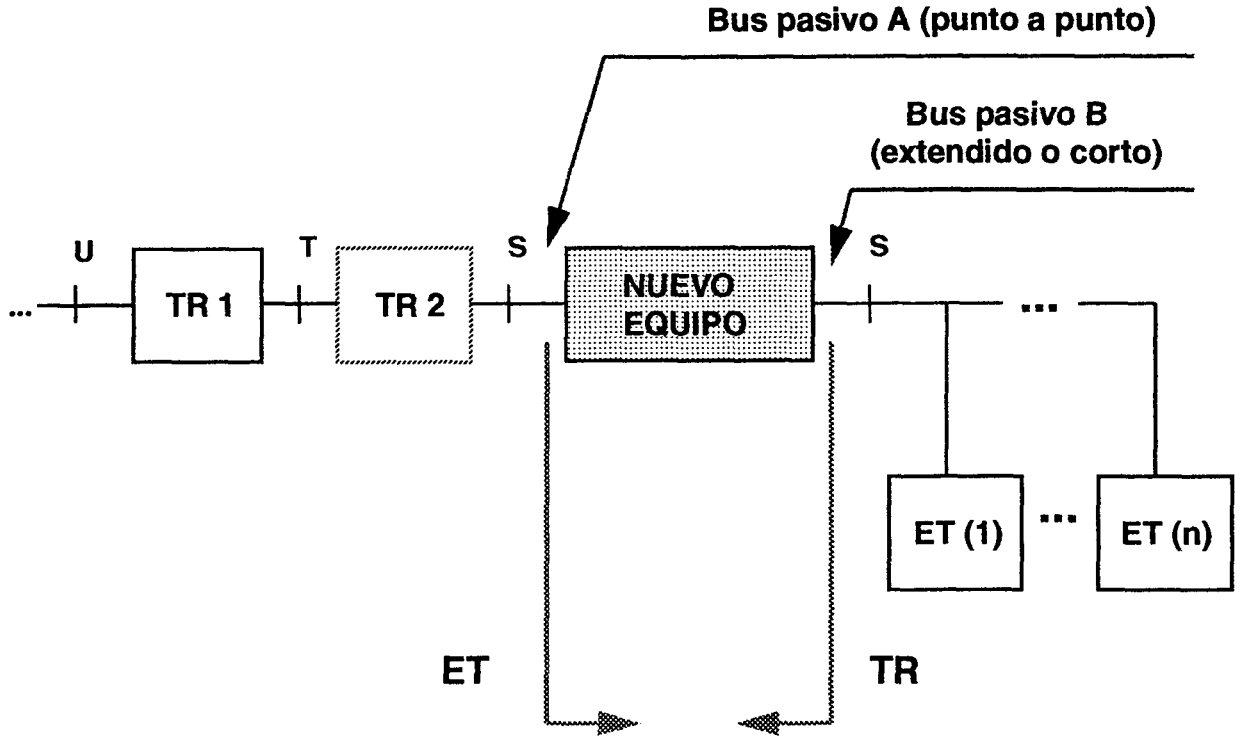


Figura 2.2 Inserción del nuevo equipo en el bus pasivo corto

El bus pasivo A tendría una configuración punto a punto, mientras que el bus pasivo B podría tener una configuración en bus pasivo corto (con la posibilidad de conectar hasta ocho Equipos Terminales con una longitud de hasta 220 m), o bus pasivo extendido (con la posibilidad de conectar hasta cuatro Equipos Terminales con una longitud de hasta 900 m).

Nuestro equipo sólo actuará sobre el canal D, tratando de forma transparente

los dos canales B. El nivel 3 será el encargado de asignar los canales B al correspondiente Equipo Terminal.

Comentar por último que para el Acceso Básico el punto de referencia S posee las mismas características mecánicas, eléctricas, de velocidad y de estructura de los canales que el punto de referencia T: 144 Kbps y estructura 2B+D. Por lo tanto en nuestro diseño el Terminal de Red 2 (TR 2) puede suprimirse sin que el funcionamiento del nuevo equipo se afecte. Por esta razón se dibuja el contorno del cuadro que representa el TR 2 en las figuras de distinta forma (puesto que lo consideramos opcional).

En resumen, siempre existirá un TR 1, pero el TR 2 será un elemento opcional en un bus pasivo para el Acceso Básico RDSI.

2.4 DESCRIPCION DEL EQUIPO

De forma explícita el equipo que pretendemos crear debe comportarse como un Equipo Terminal (ET) de cara al Terminal de Red 2 (TR 2) cumpliendo las recomendaciones del CCITT para los niveles o capas física, capa de Enlace de Datos y capa 3 para un Equipo Terminal.

Y, por otra parte, debe comportarse como un Terminal de Red (TR 2 ó TR 1) de cara al nuevo bus pasivo B cumpliendo las recomendaciones del CCITT para los niveles o capas física, de Enlace de Datos y capa 3 para un Terminal de Red.

Será también necesario crear un software aplicación que será quién, en última instancia, comunique a ambos lados del equipo.

La capa física se basa en un hardware digital cuyo componente principal es el Microcontrolador de Comunicaciones MC68302 de Motorola. Además existe un sencillo software que sirve de interfaz para con la capa 2 (funciones que implementan la capa física).

La capa 2 o capa de Enlace de Datos es el objetivo de este Trabajo Fin de Carrera cuya implementación se detalla en el capítulo 5.

Las capa 3 y la aplicación son también elementos software. La capa física, la capa 3 y la aplicación son los respectivos Trabajos Fin de Carrera de mis compañeros en el diseño del equipo global.

Algunas de las características de la aplicación que se pretende crear son las siguientes:

- **Restricción de llamadas entrantes y/o salientes en horarios y/o para terminales según selección**
- **Información de tarificación**
- **Estadística de llamadas**
- **Marcación abreviada**
- **etc**

En la siguiente figura 2.3 se representa gráficamente las capa o niveles y la aplicación que será necesario implementar en nuestro equipo. Cada una de ellas será un proceso o grupo de procesos en ejecución simultánea (trabajamos sobre un sistema operativo multiproceso). Se conectan las figuras que representan las capas o niveles mediante dos flechas para simbolizar la comunicación bidireccional simultánea (full-duplex) que existirá entre ellas. Se dibuja los canales B (2B) exteriormente al equipo para enfatizar que este nuevo equipo trata de forma transparente a los canales B, actuando sólo en el canal D.

2.4.1 Terminal asíncrono conectado al puerto serie del MC68302

Opcionalmente, los parámetros de la aplicación podrán ser modificados externamente desde un terminal asíncrono (por ejemplo un PC compatible) conectado vía puerto serie al Puerto de Comunicaciones Serie (SCP) del Microcontrolador MC68302.

Se ejecutará un software en el PC que se comunicará a través de dicho puerto serie con el proceso software aplicación de nuestro equipo. De esta forma se podrán modificar los parámetros de la aplicación, por ejemplo, los relacionados con las

restricciones de llamadas. Además la aplicación podrá enviar información al software del PC, por ejemplo, la información concerniente a la estadística y tarificación de llamadas.

El acceso al software del PC se restringirá mediante una clave o "password".

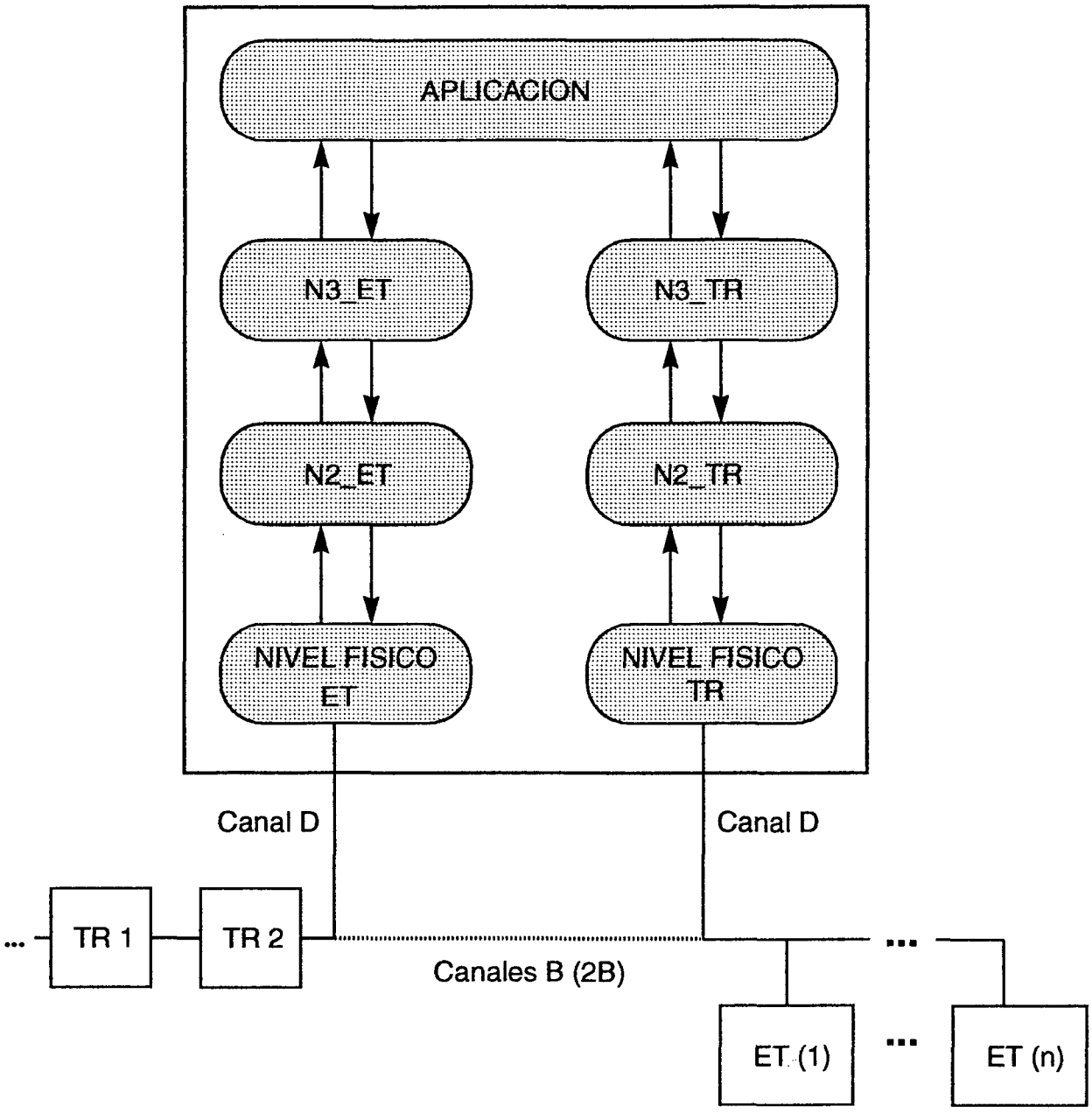


Figura 2.3 Capas o niveles del nuevo equipo

2.4.2 Síntesis

Resumiendo, lo que se pretende es aprovechar las ventajas de la distancia del bucle de abonado punto a punto de longitud máxima para poder conectar un nuevo bucle de abonado y desarrollar una aplicación que permita entre otras cosas mantener un control restrictivo y estadístico de las llamadas entrantes y salientes haciendo las funciones de una "centralita inteligente".

Los parámetros de este nuevo equipo o centralita inteligente podrían ser modificados desde un terminal asíncrono (PC) conectado al MC68302 vía puerto serie. En la figura 2.4 podemos observar como quedaría el esquema final del bus punto a punto con el nuevo equipo y el terminal asíncrono. .

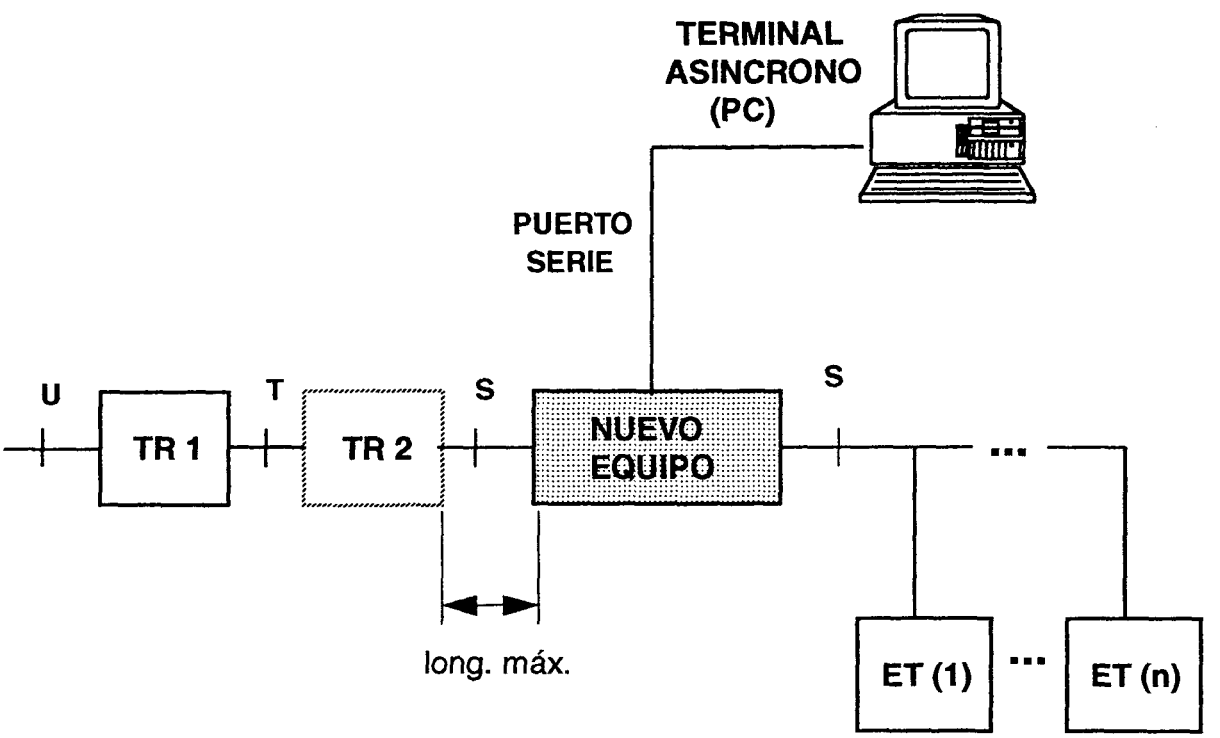


Figura 2.4 Instalación del nuevo equipo con el Terminal Asíncrono (PC) conectado al Puerto Serie (SCP) del MC68302

2.5 OBJETIVOS DE ESTE TRABAJO FIN DE CARRERA

En este Trabajo Fin de Carrera, como ya hemos comentado, realizamos la implementación del protocolo LAP-D cuyo objetivo es la transferencia de información entre entidades de capa 3 a través del interfaz usuario-red de la RDSI, utilizando el canal D.

Implementamos los procesos-entidades correspondientes a la Capa de Enlace de Datos tanto del lado del Equipo Terminal como del lado del Terminal de Red.

En el nuevo equipo existirá una serie de procesos ejecutándose, uno por cada entidad o proceso de la capa 2 o capa de Enlace de Datos (entidades de capa de Enlace de Datos, entidades de gestión de capa de Enlace de Datos y el proceso multiplexor o concentrador) que sea necesario.

Los procesos son creados por el módulo Kernel a partir de los ficheros implementados que, serán linkados conjuntamente para formar un solo programa global. El kernel comenzará la ejecución de un proceso a partir de la función principal de ese proceso (a partir de la función principal de un fichero).

Las funciones principales de los ficheros creados son:

n2_ETmain: Implementa el proceso-entidad de capa de Enlace de Datos del lado del Equipo Terminal

n2_TRmain: Implementa el proceso-entidad de capa de Enlace de Datos del lado del Terminal de Red

n2_GED_ETmain: Implementa el proceso-entidad de gestión de capa de la capa de Enlace de Datos del lado del Equipo Terminal

n2_GED_TRmain: Implementa el proceso-entidad de gestión de capa de la capa de Enlace de Datos del lado del Terminal de Red

n2_MuxTRmain: Implementa el proceso multiplexor o concentrador intermedio entre las entidades de capa de Enlace de Datos del lado del Terminal de Red y la capa física

2.5.1 Lado del Equipo Terminal

En la figura 2.5 podemos observar un esquema de las entidades o procesos de capa 3, de capa de Enlace de Datos y de la capa Física del lado del Equipo Terminal:

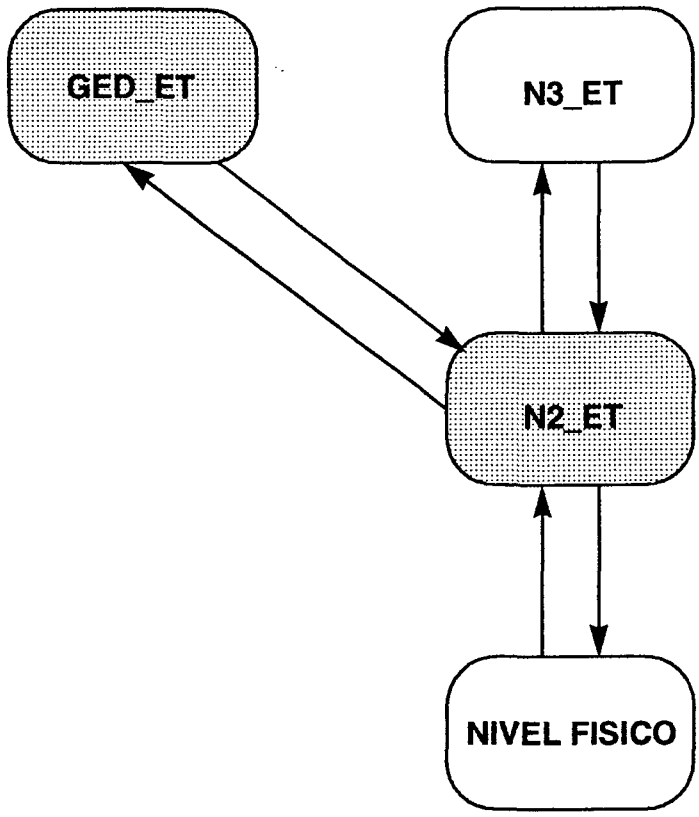


Figura 2.5 Procesos o niveles del lado del Equipo Terminal

2.5.1.1 Entidad-proceso de Capa de Enlace de Datos

En el lado del Equipo Terminal utilizaremos un sólo proceso para implementar una sola entidad de capa de Enlace de Datos.

El proceso de aplicación junto con el proceso-entidad de capa 3 serán los encargados de enviar la información transferida desde las entidades-procesos de capa de Enlace de Datos del lado del Terminal de Red (provenientes de los distintos Equipos Terminales conectados al bus pasivo B), a la entidad-proceso de capa de Enlace de Datos del lado del Equipo Terminal y; ésta a su vez, enviará dicha información utilizando el protocolo LAP-D a través del nivel físico del lado del Equipo

Terminal hacia el TR 2, o en su defecto, al TR 1.

2.5.1.2 Entidad-proceso de gestión de Capa de Enlace de Datos

Será necesario, además, un proceso que implemente la entidad de gestión de capa del lado de usuario (o lado del ET) para, entre otras cosas, solicitar un IET o Identificador de Equipo Terminal que identifique la conexión de Enlace de Datos.

La entidad-proceso de gestión se comporta como una entidad de capa 3 para la entidad de capa de Enlace de Datos, por eso dibujamos la entidad de gestión y la entidad de capa 3 a la misma altura en la figura 2.5 .

2.5.2 LADO DEL TERMINAL DE RED

En la figura 2.5 podemos observar un esquema de las entidades o procesos de capa 3, de capa de Enlace de Datos y de la capa Física del lado del Terminal de Red. Además observamos los procesos multiplexores o concentradores que ha sido necesario implementar y que, posteriormente, explicaremos.

2.5.2.1 Entidades-procesos de Capa de Enlace de Datos

En el lado del Terminal de Red (TR) deberá existir al memos un proceso-entidad de capa de Enlace de Datos (ED) por cada Equipo Terminal conectado al bus pasivo (bus pasivo B).

Cada proceso-entidad de Capa de Enlace de Datos del lado del Terminal de Red podrá establecer una conexión de Enlace de Datos con una entidad de Capa de Enlace de Datos de uno de los Equipos Terminales conectados al bus pasivo B.

2.5.2.2 Proceso multiplexor o concentrador

Existirá un proceso que llamaremos multiplexor o concentrador y que será un proceso intermedio entre las entidades de capa de Enlace de Datos del lado TR y la capa física (implementada por medio de determinadas funciones). Este proceso es necesario para dirigir las tramas LAP-D generadas por las entidades de capa de Enlace de Datos a la capa física, es decir, para enviarlas al puerto físico que representa el canal D del bus pasivo B; también es necesario para enviar las tramas

que se reciben del bus pasivo desde el nivel físico a sus respectivas entidades-procesos destino de capa de Enlace de Datos. En síntesis, el proceso multiplexor es necesario al ser necesarios los distintos procesos de entidad de Capa de Enlace de Datos en el lado del Terminal de Red.,

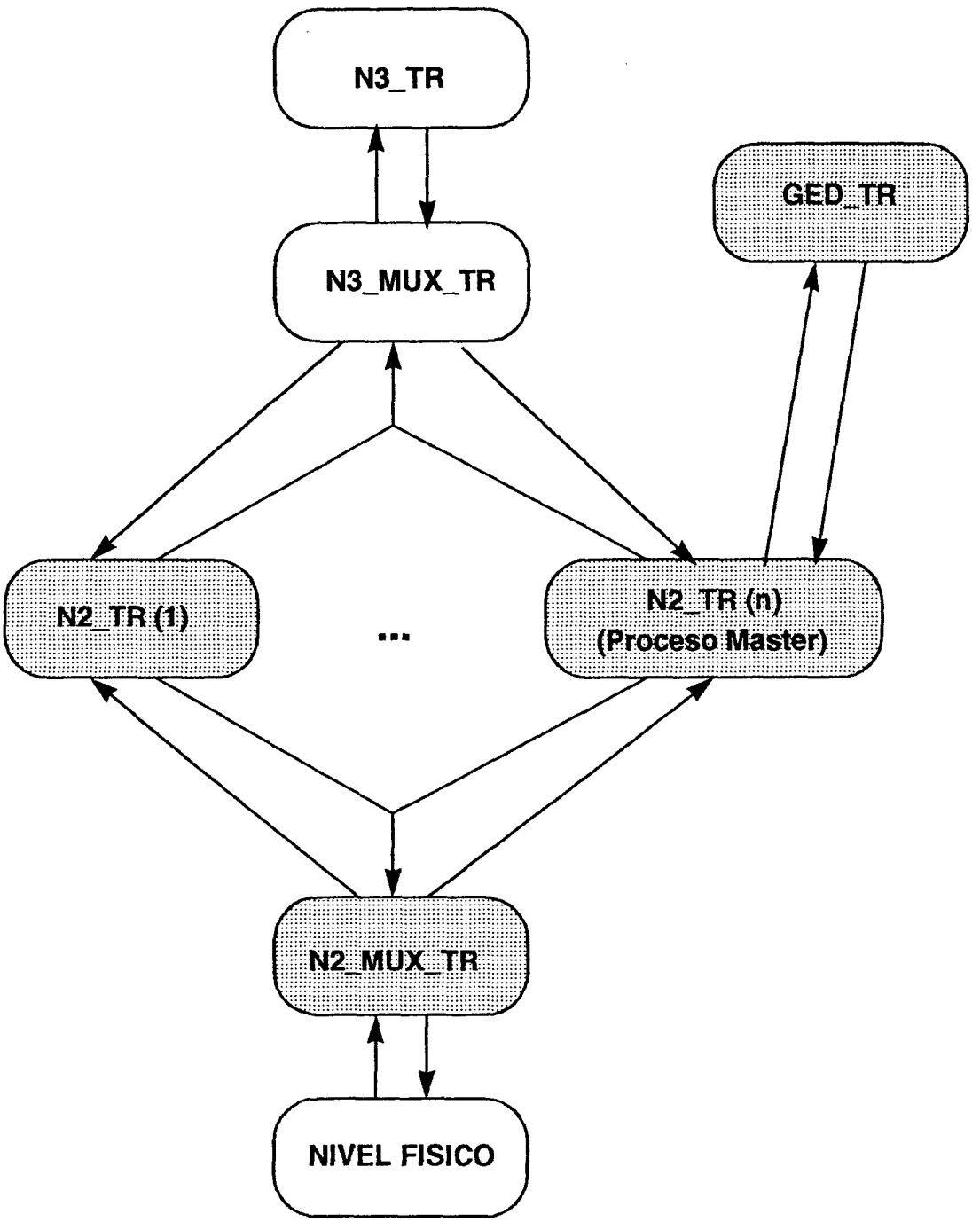


Figura 2.5 Procesos o niveles del lado del Terminal de Red

2.5.2.3 Entidad-proceso de gestión de Capa de Enlace de Datos

De forma análoga al lado del Equipo Terminal, en el lado del Terminal de Red existirá una entidad-proceso de gestión de capa. La entidad de gestión de capa de Enlace de Datos del lado del Terminal de Red "mantendrá un intercambio de tramas de instrucción UI" (Información no numerada) con las entidades de gestión de capa de los Equipos Terminales conectados al bus pasivo B, enviándose los mensajes utilizados para los procedimientos de gestión de capa o procedimientos de gestión de IET o Identificador de Equipo Terminal.

2.5.2.4 Número de entidades-procesos de Capa de Enlace de Datos

El número de procesos de Capa de Enlace de Datos del Terminal de Red será igual a un valor definido "NUM_PROCESOS_N2_TR" que hemos dimensionado a un valor de 12. Este valor puede ser aumentado en cualquier momento a un valor mayor (recompilando el software) si se comprueba que así es necesario. En la figura 2.5 se representa la existencia de "n" procesos de Capa de Enlace de Datos con lo que se quiere hacer referencia a la posibilidad de dimensionar este valor según las necesidades de cada momento, según el número de Equipos Terminales que se vayan a conectar al bus pasivo B.

Este número de procesos de Capa de Enlace de Datos siempre debe ser mayor que el número de Equipos Terminales conectados al bus pasivo B.

La necesidad de mantener un número de procesos de Capa de Enlace de Datos mayor que el número de Equipos Terminales conectados al bus pasivo "B" viene determinada por las siguientes razones:

1.- Puede producirse la caída (desconexión o apagado) de un Equipo Terminal sin notificación al Terminal de Red (a la entidad-proceso de capa de Enlace de Datos del lado del Terminal de Red). En este caso puede transcurrir (T203 + (3*T200)) segundos *-máximo tiempo permitido sin tener lugar un intercambio de tramas sobre la conexión de Enlace de Datos más el tiempo correspondiente a un número de tres*

retransmisiones de una trama de supervisión (RR) cada un segundo- antes de que el lado del Terminal de Red (la entidad de Capa de Enlace de Datos del lado del TR) decida que la conexión de Enlace de Datos que empleaba ese determinado IET se ha liberado, y permitir que ese proceso-entidad de Capa de Enlace de Datos pueda establecer otra conexión de Enlace de Datos utilizando un nuevo IET.

2.- Por otra parte, un Equipo Terminal puede solicitar más de un Identificador de Equipo Terminal (IET) para comunicarse "a través" de varias conexiones de Enlace de Datos con su entidad par (entidades) de Capa de Enlace de Datos (en esta implementación con distintos procesos-entidades de Capa de Enlace de Datos).

2.5.2.5 Entidad-proceso master

Uno de los procesos-entidades de Capa de Enlace de Datos del lado del Terminal de Red hará las funciones de "proceso master" que será el único proceso que se comunique con el proceso-entidad de gestión de Capa de Enlace de Datos del lado del Terminal de Red.

Este proceso master recibirá y enviará las tramas de información no confirmada a la entidad de gestión de capa del lado del Terminal de Red. También estará encargado de enviar al nivel 3 la información sin acuse de recibo recibida en las tramas UI que le llegan al nivel físico procedentes de los Equipos Terminales conectados al bus pasivo B.

2.5.2.6 Multiplexor o concentrador de nivel 3

Por último señalar que debe existir un proceso de nivel 3 que haga las funciones de multiplexor o concentrador previo a la entidad de nivel 3 en el lado del Terminal de Red. Será un proceso intermedio entre los procesos-entidades de Capa de Enlace de Datos del lado del Terminal de Red y el proceso-entidad de capa de la capa 3. La implementación de este proceso corresponde al compañero que desarrolla el software de nivel 3.

Capítulo 3: Descripción Normativa

3.1 OBJETO

El presente capítulo tiene por objeto la descripción de las normas que especifican la capa de enlace de datos (capa 2) de la interface usuario/red en la Red Digital de Servicios Integrados (RDSI): Recomendaciones Q.920 y Q.921 (Comisión de Estudio XI) del Comité Consultivo Internacional Telegráfico y Telefónico (CCITT). Estas recomendaciones se encuentran recogidas en el Tomo VI - Fascículo VI.10 del Libro Azul.

En la Recomendación Q.920 se describe de forma general el procedimiento de acceso al enlace en el canal D (LAPD). El LAPD tiene por objeto transferir información entre entidades de capa 3 a través del interfaz usuario-red de la RDSI, utilizando el canal D.

En la Recomendación Q.921 se especifican la estructura de trama, los elementos de procedimiento, los formatos de los campos y los métodos para el funcionamiento correcto del procedimiento de acceso al enlace en el canal D, LAPD.

3.2 CONSIDERACIONES GENERALES

El LAPD tiene por objeto transferir información entre entidades de capa 3 a través del interfaz usuario-red de la RDSI, utilizando el canal D.

En la definición del LAP-D se han tenido en cuenta los principios y la terminología de las siguientes recomendaciones y normas:

- Las Recomendaciones X.200 y X.201 - Modelo de referencia y convenios de

servicio de capa para la interconexión de sistemas abiertos (ISA).

- La Recomendación X.25 LAPB - Interfaz usuario-red para terminales en el modo paquete.

- Las normas ISO 3309 e ISO 4335 - Especificación para la estructura de trama y elementos de procedimiento de control de alto nivel para enlaces de datos (HDLC).

El LAPD es un protocolo que opera en la capa de enlace de datos de la arquitectura OSI (Interconexión de Sistemas Abiertos) de ISO (Organización Internacional de Estándares).

De acuerdo con la arquitectura de interconexión de sistemas abiertos, OSI, la interface usuario/red está estructurada en tres capas o niveles: capa física (capa 1), capa de enlace de datos (capa 2) y capa 3.

Cada capa contiene entidades. Las entidades de una misma capa, pero de sistemas diferentes que deben intercambiar información para realizar un objetivo común se denominan "entidades pares". La interacción entre entidades de capas adyacentes se realiza a través de su límite común. Los servicios proporcionados por la capa de enlace de datos son el resultado de los servicios y funciones proporcionados tanto por la capa enlace de datos como por la capa física.

Un punto de acceso al servicio (PAS) de capa de enlace de datos es el punto que utiliza la capa de enlace de datos para proporcionar servicios a la capa 3. Asociados con cada PAS de capa enlace de datos hay uno o varios puntos extremos de conexión. Un punto extremo de conexión de enlace de datos se identifica mediante un identificador de punto extremo de conexión de enlace de datos visto desde la capa 3, y mediante un identificador de conexión de enlace de datos (ICED), visto desde la capa enlace de datos (todo esto se explica detalladamente en el apartado titulado Identificación de la conexión de enlace de datos).

La cooperación entre entidades de capa enlace de datos se rige por un protocolo entre pares específico a la capa. A fin de poder intercambiar información

entre dos o más entidades de capa 3, se tiene que establecer una asociación entre las entidades de capa 3, utilizando en la capa enlace de datos un protocolo de capa enlace de datos. Esta asociación se denomina conexión de enlace de datos. Las conexiones de enlace de datos las proporciona la capa enlace de datos entre dos o más PAS.

Las unidades de mensaje de capa de enlace de datos se transfieren entre entidades de capa enlace de datos por medio de una conexión física.

3.3 INTERCAMBIO DE PRIMITIVAS

Las primitivas representan de un modo abstracto el intercambio lógico de información y de control entre las entidades funcionales de capas adyacentes y pertenecientes al mismo sistema originando los protocolos de comunicación entre capas. Este intercambio de primitivas no especifica ni implica ninguna limitación en el modo de implantación y funcionamiento de las entidades funcionales o del interfaz de señalización.

En lo que se refiere a la entidad de la capa de enlace de datos el intercambio de primitivas tendrá lugar con la entidad de la capa 3 y la entidad de gestión de capa de la capa 2 a las que considerará como entidades de capa superior y con la entidad de la capa física a la que considerará como entidad de capa inferior.

Tal como se indica de un modo genérico en la figura 2.1 , las primitivas que se intercambian entre dos entidades adyacentes pueden ser de uno de los cuatro tipos siguientes:

PETICION: Utilizada cuando una entidad de la capa superior está solicitando un servicio de la entidad de la capa inferior.

INDICACION: Utilizada por la entidad de la capa que está proporcionando un servicio para notificar a la entidad de la capa superior de cualquier actividad relacionada con el servicio. Una primitiva del tipo INDICACION puede ser el resultado de una actividad de la capa inferior (protocolo entre pares) desencadenada por una

primitiva del tipo PETICION en la entidad par.

RESPUESTA: Utilizada por la entidad de una capa para acusar recibo de la recepción de una primitiva INDICACION procedente de la capa inferior.

CONFIRMACION: Utilizada por la entidad de la capa que está proporcionando un servicio previamente solicitado para confirmar a la entidad de la capa superior que la actividad relacionada con dicho servicio ha sido completada.

Una primitiva del tipo CONFIRMACION puede ser el resultado de una actividad de la capa inferior (protocolo entre pares) desencadenada por una primitiva del tipo RESPUESTA en la entidad par.

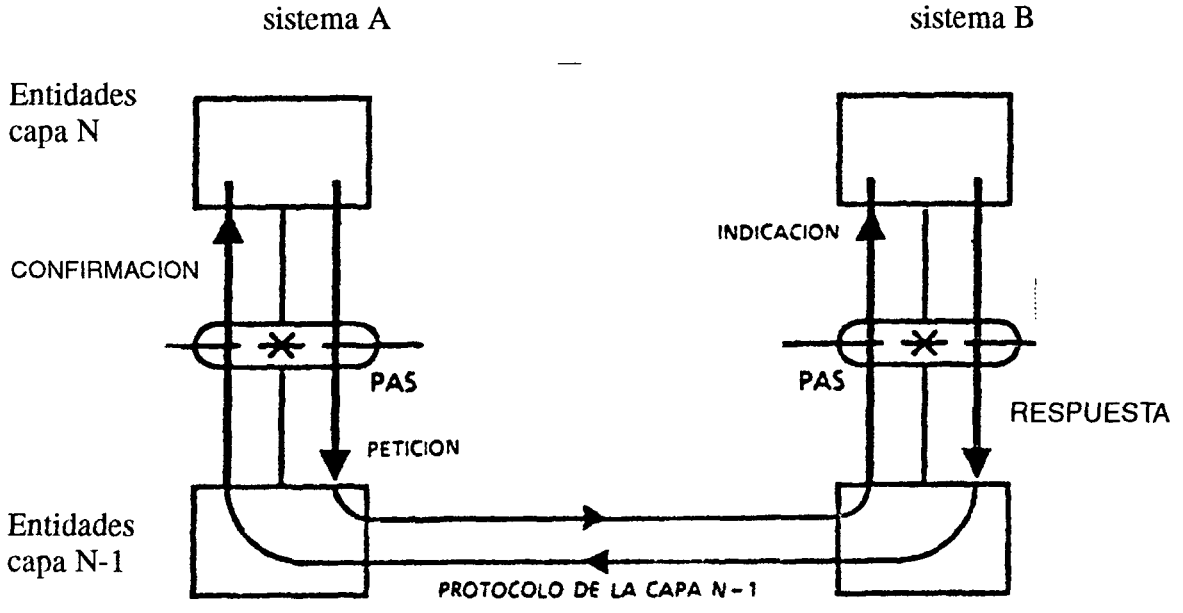


Figura 3.1 .TIPOS DE PRIMITIVAS

La sintáxis general utilizada para la notación de las primitivas responde al modelo: CAPA- NOMBRE GENERICO - TIPO donde CAPA indica la capa funcional del sistema a la que pertenece la entidad que proporciona el servicio implicado en el intercambio de primitivas. Para el caso de la capa de enlace de datos del protocolo usuario/red en la RSDI puede ser:

FI: entidad de la capa física.

GFI: entidad de gestión de capa de la capa física.

ED: entidad de la capa de enlace de datos.

GED: entidad de gestión de capa de la capa de enlace de datos

donde las primitivas GFI han sido introducidas, a pesar de no existir una comunicación directa entre la entidad de la capa de enlace de datos y la entidad de gestión de capa de la capa física, ya que existe una relación muy directa entre los servicios proporcionados por la entidad de la capa física a dicha entidad de gestión de capa y ciertas funciones de la capa de enlace de datos a través de la entidad de gestión de capa de la capa de enlace de datos.

Los diferentes NOMBRES GENERICOS están relacionados con las funciones específicas de cada una de las primitivas, es decir, con el servicio que la entidad designada deberá realizar. El TIPO de primitiva será siempre uno de los cuatro anteriormente definidos (petición, indicación, respuesta o confirmación).

Las entidades de la capa de enlace de datos como consecuencia de sus funciones características y servicios que proporcionan necesitan mantener un intercambio de primitivas con las entidades de capa superior (entidad de la capa 3 y entidad de gestión de capa) y con las entidades de capa inferior (entidad de la capa física). Estas primitivas quedan reflejadas en el cuadro de la figura 3.2 . La aplicabilidad y funciones de cada una de las primitivas se describirá en las secciones posteriores correspondientes.

Para el caso de acceso primario en el que algunas de las funciones características de la capa 2 no son soportadas, las primitivas relacionadas con la capa de enlace de datos son las reflejadas en el cuadro de la figura 3.3 que constituye una simplificación del de la figura 3.2

Figura 3.2 PRIMITIVAS RELACIONADAS CON LA CAPA DE ENLACE DE DATOS.

Capa-nombre genérico	Tipo				Parámetro		Contenido de la unidad de mensaje
	Petición	Indicación	Respuesta	Confirmación	Indicador de prioridad	Unidad de mensaje	
capa2/capa3							
ed- establecimiento	X	X		X			
ed-liberación	X	X		X			
ed-datos	X	X				X	mensaje de la capa 3 del protocolo entre pares de la capa 3
ed-unidad de datos	X	X				X	mensaje de la capa 3 del protocolo entre pares de la capa 3
capa2/entidad de gestión de capa							
ged-asignación	X	X				X	valor IET(capa2)y SEC(capa3)
ged-supresión	X					X	valor IET(capa2)y SEC(capa3)
ged-error		X	X			X	motivo del mensaje de error
ged-unidad de datos	X	X				X	mensaje entre pares con funciones de gestión
ged-xid	X	X	X	X		X	información de gestión de la conexión
capa1/capa2							
fi-datos	X	X			X(sólo para solicitud)	X	trama de la capa 2 del protocolo entre pares LAP D
fi-activación	X	X					
fi-desactivación o		X					
capa1/entidad de gestión de capa							
gfi-activación		X					
gfi-desactivación	X	X					
gfi-información		X				X	conectado/ desconectado

Figura 3.3 PRIMITIVAS RELACIONADAS CON LA CAPA DE ENLACE DE DATOS PARA EL CASO DE ACCESO PRIMARIO DE USUARIO.

Capa-Nombre genérico	Tipo				Parámetro		Contenido de la unidad de mensaje
	Petición	Indicación	respuesta	confirmación	indicador de prioridad	unidad de mensaje	
capa2/capa3							
ed- establecimiento	X	X		X			
ed-liberación	X	X		X			
ed-datos	X	X				X	mensaje de la capa 3 del protocolo entre pares de la capa 3
capa2/entidad de gestión de capa							
ged-asignación nota	X	X				X	valor IET(capa2)y SEC(capa3)
ged-supresión nota	X					X	valor IET(capa2)y SEC(capa3)
ged-error		X	X			X	motivo del mensaje de error
capa1/capa2							
fi-datos	X	X			X(sólo para petición)	X	trama de la capa 2 del protocolo entre pares LAP D
fi-activación	X	X					
fi-desactivación		X					
capa1/entidad de gestión de capa							
gfi-activación		X					

En las tablas de la figuras 3.2 y 3.3 se puede observar la existencia de los dos parámetros, asociados a las primitivas, que a continuación se definen:

Indicador de prioridad

Teniendo en cuenta la posible existencia de varios PAS asociados a la misma entidad de capa 2 del lado de red o del lado de usuario, las unidades de mensaje del

protocolo enviadas por un PAS pueden disputar la utilización de los recursos físicos (de capa 1) disponibles para la transferencia de los mensajes con las unidades de mensaje enviadas por otros PAS. El indicador de prioridad se utiliza para determinar qué unidad de mensaje tiene más prioridad para la transmisión cuando se plantea este problema de contención. En el lado de usuario, el indicador de prioridad es necesario solamente para la diferenciación de unidades de mensaje generados por PAS con IPAS de valor "0" (señalización) de cualquier otro tipo de unidad de mensaje.

Unidad de mensaje

La unidad de mensaje contiene información adicional de capa a capa concerniente a acciones y a resultados asociados a las solicitudes de servicios de capa. En el caso de las primitivas de datos, la unidad de mensaje contendrá el mensaje entre pares de la capa solicitante. Por ejemplo, la unidad de mensaje ED-DATOS contiene mensajes de la capa 3 y la unidad de mensaje FI-DATOS contendrá la trama de la capa de enlace de datos.

Las operaciones a través de la frontera entre capa 3, deben ser tales que la capa enviando ED-DATOS o ED-UNIDAD DE DATOS pueda suponer un orden temporal de los bit en la unidad de mensaje y la capa receptora pueda reconstruir el mensaje con el orden temporal supuesto.

3.4 CARACTERISTICAS DEL SERVICIO DE LA CAPA DE ENLACE DE DATOS

Mediante el intercambio de primitivas (ED-NOMBRE GENERICO-TIPO) la capa de enlace de datos proporcionará unos servicios a la capa 3, como consecuencia de los servicios que a su vez recibe de la capa física y de las funciones características que esta capa desempeña.

Adicionalmente la capa de enlace de datos proporcionará unos servicios de gestión a la entidad de gestión de capa de la capa de enlace de datos con la que mantendrá el necesario intercambio de primitivas (GED-NOMBRE GENERICO-TIPO).

3.4.1 SERVICIOS REQUERIDOS DE LA CAPA FISICA

Los servicios que la capa física proporciona a la capa de enlace de datos se describen en detalle en el capítulo correspondiente, dedicado a la especificación de la capa física del interfaz usuario/red en la RDSI. Estos servicios son los resumidos a continuación:

a) Capacidad de transmisión: Conexión de la capa física para la transmisión transparente de bits en el mismo orden en que son enviados hacia la capa física. Las primitivas relacionadas con este servicio son: FI-DATOS-PETICION/INDICACION.

b) Activación/desactivación: paso de los equipos terminales y de terminación de red al modo de bajo consumo de energía. Las primitivas relacionadas con este servicio son: FI-ACTIVACION-PETICION/INDICACION y FI-DESACTIVACION-INDICACION.

Para el caso del acceso primario de usuario, en el cual no se incluyen los procedimientos entre pares de activación y desactivación de la capa física, estas primitivas se utilizarán cuando tenga lugar una desactivación de la capa física o algún error de funcionamiento irrecuperable de dicha capa física.

c) Acceso al canal D: Control del acceso de los eguipos terminales al recurso común del canal D y transmisión de las tramas de la capa 2 de acuerdo con su prioridad de capa de enlace de datos.

Adicionalmente la capa física proporcionará unos servicios de gestión a la entidad de gestión de capa de la capa física que tendrán implicaciones en las funciones propias de la capa de enlace de datos. Estos servicios son los resumidos a continuación:

a) Activación/desactivación. Paso de los equipos terminales y de terminación de red al modo de bajo consumo de energía. Las primitivas relacionadas con este servicio son: GFI-ACTIVACION-INDICACION y GFI-DESACTIVACION-PETICION/ INDICACION.

Para el caso de acceso primario de usuario, en el cual no se incluyen los procedimientos entre pares de activación y desactivación de la capa física, estas primitivas se utilizarán cuando tenga lugar una desactivación de la capa física o algún error de funcionamiento irrecuperable de dicha capa física.

b) Gestión: Indicación del estado físico del canal D (conectado/desconectado). La primitiva relacionada con este servicio es: GFI-INFORMACION-INDICACION. Dado que en el caso de acceso primario de usuario (30B+D) la entidad funcional TR2 siempre se encuentra físicamente conectada, este servicio solamente es aplicable al caso de acceso básico de usuario (2B+D).

3.4.2 FUNCIONES DE LA CAPA DE ENLACE DE DATOS

Como ya se ha mencionado, el LAP D tiene por misión el transporte de información entre entidades de la capa 3 de sistemas diferentes a través del interfaz usuario/red, utilizando como medio el canal D. El LAP D con su estructura puede soportar tanto instalaciones multiterminales en el interfaz usuario/red como múltiples entidades de la capa 3.

Todas las unidades de mensaje de la capa de enlace de datos son transmitidas estructuradas en tramas, las cuales están claramente delimitadas mediante indicadores.

Para llevar a cabo esta tarea el LAP D debe desempeñar las funciones que se enumeran a continuación:

a) Provisión de una o más conexiones de enlace de datos sobre un mismo canal D. La discriminación entre las diferentes conexiones de enlace de datos se realiza mediante el identificador de conexión de enlace de datos (ICED) contenida en cada una de las tramas de la capa 2.

b) Delimitación, alineación y transparencia de las tramas, características que permiten la identificación unívoca de una secuencia de bits, transmitida sobre el canal D, como una trama.

c) Control de secuencia para mantener el orden secuencial de las tramas a través de la conexión de enlace de datos.

d) Detección de errores de transmisión, formato y operacionales que pudieran producirse en la conexión enlace de datos establecida.

e) Recuperación de las condiciones de error detectadas y notificación a la entidad de gestión de capa de aquellas que son irrecuperables por la propia entidad de la capa 2.

f) Control de flujo para asegurar la generación de tramas de información con una frecuencia adecuada a la capacidad de procesamiento de la entidad receptora de dichas tramas.

3.4.2.1 Tipos de enlace de datos

Las funciones de la capa 2 proporcionan los medios necesarios para la transferencia de información entre puntos extremos conectados por conexiones de enlace de datos, soportando además el establecimiento de diferentes combinaciones de dichos puntos extremos, de modo que la transferencia de información puede tener lugar mediante conexiones de enlace de datos "punto a punto", caracterizadas porque una trama está dirigida hacia un punto terminal único o mediante conexiones de enlace de datos de "difusión", caracterizadas porque las tramas están dirigidas hacia uno o más Puntos extremos.

En la figura 3.4 se muestran dos ejemplos de transferencia de información de la capa 2 punto a punto y en la figura 3.5 se muestra un ejemplo de transferencia de información tipo difusión

Para el caso de instalaciones básicas de usuario, tipo bus pasivo 2B+D (canal D de 16 kbit/s) los enlaces de datos podrán ser tanto punto a punto como de difusión. Para el caso de accesos de usuario de jerarquía primaria 30B+D (el canal D será de 64 kbit/sg) los enlaces de datos serán siempre del tipo punto a punto.

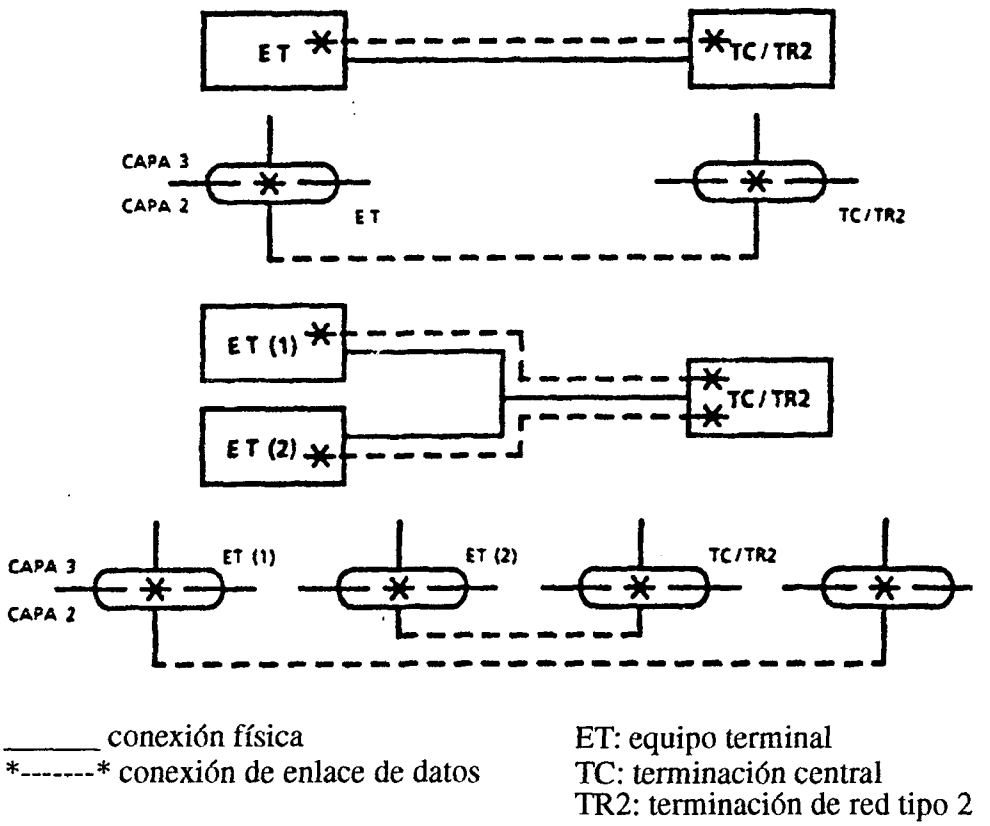


Figura 3.4 . CONEXIONES DE ENLACE DE DATOS PUNTO A PUNTO.

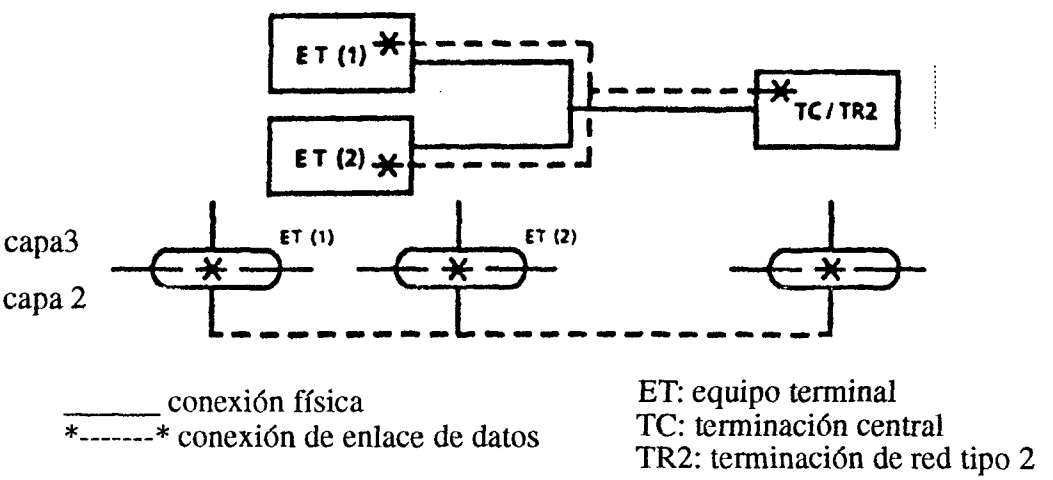


Figura 3.5 . CONEXIONES DE ENLACE DE DATOS DE DIFUSIÓN

3.4.2.2 Tipos de operación de la capa de enlace de datos

Se han definido dos tipos de operación de la capa de enlace de datos para la transferencia de información de la capa 3, los cuales deben coexistir sobre un único canal D: operación sin acuse de recibo y operación con acuse de recibo.

3.4.2.2.1 Operación sin acuse de recibo

Con este tipo de operación la información de la capa 3 es transmitida en tramas de información no numeradas (UI), tramas que no son objeto de acuse de recibo en la capa 2, de modo que aunque son detectados los errores de transmisión y de formato, no se define ningún mecanismo de recuperación de dichos errores. De una forma similar tampoco se define mecanismo de control de flujo para la transmisión de estas tramas.

Este tipo de operación es aplicable solamente para la transferencia de información de difusión. Por tanto, este modo de operación solamente será aplicable para el caso de acceso básico de usuario

3.4.2.2.2 Operación con acuse de recibo

Con este tipo de operación la información de la capa 3 es enviada en tramas que son objeto de acuse de recibo en la propia capa de enlace de datos

Para este tipo de operación se definen mecanismos de detección de errores y recuperación de los mismos mediante la retransmisión de tramas sin acuse de recibo. La entidad de gestión es informada en el caso de que se detecten errores no recuperables por la capa 2.

En este tipo de operación también se definen los mecanismos adecuados para poder llevar a cabo un control de flujo de la transmisión.

La operación con acuse de recibo es aplicable para la transferencia de información punto a punto, definiéndose un modo de transferencia en multitrama en la que la información de la capa 3 es transmitida mediante tramas de información numeradas (I). Este tipo de operación permite que en un determinado momento puedan permanecer pendientes de reconocimiento un grupo de tramas I cuyo número "k" dependerá del módulo de operación, módulo que para el caso de transferencia de información en el modo multitrama tendrá el valor 128.

Este modo de operación será aplicable tanto para el caso de acceso básico de

usuario como de acceso primario.

3.4.2.3 Estados del enlace de datos

Una entidad de la capa de enlace de datos punto a punto puede estar en uno de los tres estados básicos siguientes:

a) Estado de identificador de equipo terminal (IET) no asignado, en el que el IET aún no ha sido asignado. En este estado no es posible ningún tipo de transferencia de información de la capa 3.

Dado que en el caso de acceso primario de usuario siempre se utilizará el IET de valor cero, este estado no será aplicable para este tipo de acceso.

b) Estado de identificador de equipo terminal (IET) asignado, en el que el IET ya ha sido asignado mediante el procedimiento de asignación de IET. En este estado es posible la transferencia de información sin acuse de recibo

c) Estado de multitrama establecida, al que se ha llegado mediante el uso del procedimiento de establecimiento del modo multitrama, estado en el que es posible la transferencia de información con y sin acuse de recibo.

En el diagrama de la figura 3.6 se representan las transiciones entre estos tres estados y los procedimientos implicados en dichas transiciones.

Una entidad de la capa de enlace de datos de difusión se encontrará permanentemente en un estado de transferencia de información en el que es posible solamente la transferencia de información sin acuse de recibo, es decir, en el estado IET asignado.

3.4.3 SERVICIOS OFRECIDOS A LA CAPA 3

Como ya ha sido comentado, la capa de enlace de datos proporciona una serie de servicios a la capa 3 y a la entidad de gestión de capa de la capa 2 utilizando los servicios proporcionados por la capa física. El presente capítulo se ocupa de los servicios proporcionados a la capa 3, mientras que el 3.4.4, detalla los servicios proporcionados a la entidad de gestión de capa de la capa de enlace de datos.

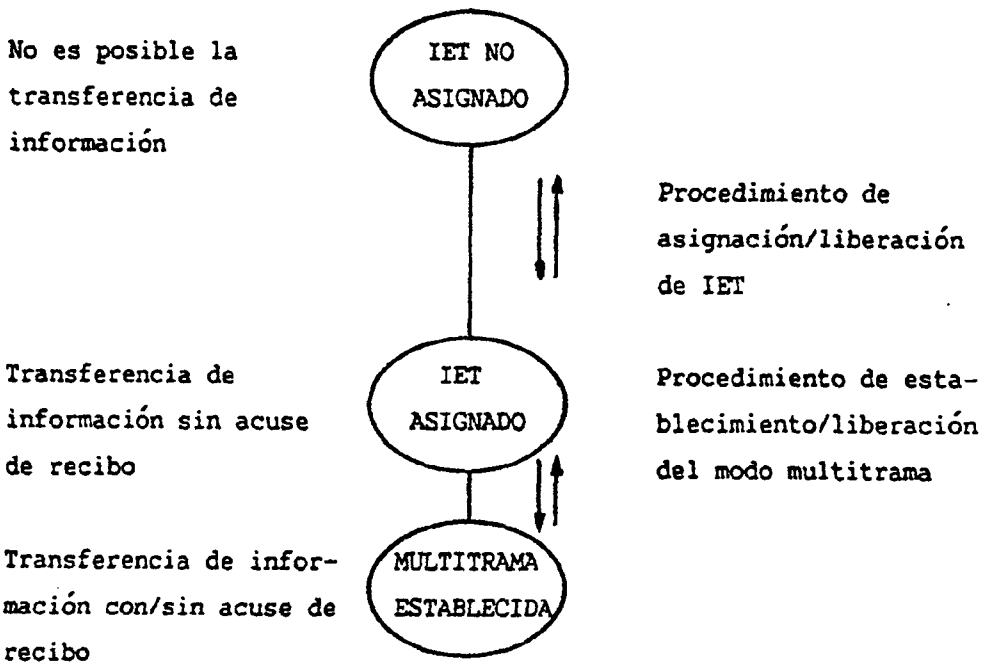


Figura 3.6 . ESTADOS DE ENLACE DE DATOS PUNTO A PUNTO

De acuerdo con el modelo de referencia de la capa de enlace de datos la prestación de estos servicios requerirá un intercambio de primitivas entre entidades de capa adyacentes.

Asociados con la capa 3 existen dos servicios diferentes de transferencia de información mediante conexiones de enlace de datos, el primero basado en la función de transferencia de información sin acuse de recibo por la capa 2 y el segundo basado en la función de transferencia de información con acuse de recibo.

3.4.3.1 Servicio de transferencia de información sin acuse de recibo

Aunque para este servicio no se establece un acuse de recibo de la información por la entidad de la capa 2, pueden existir dichos procedimientos implantados en alguna de las capas superiores.La transferencia de información sin acuse de recibo solamente será aplicable al caso de acceso básico de usuario (conexiones de enlace de datos de difusión).

Las características de este servicio pueden resumirse en las tres siguientes:

a) Provisión de conexiones de enlace de datos entre entidades de la capa 3 que permitan la transferencia de unidades de mensaje de la capa 3 (mensajes) sin acuse de recibo de los mismos.

b) Identificación de punto extremo de conexión de enlace de datos (IECED) que permita que una entidad de la capa 3 pueda ser identificada por otra de esta misma capa.

c) Las entidades de la capa de enlace de datos no llevan a cabo ningún tipo de verificación de las unidades de mensaje de la capa 2 recibidas.

Las primitivas asociadas con el servicio de transferencia de información sin acuse de recibo son las siguientes:

ED-UNIDAD DE DATOS-PETICION (ED-UDA-P): Utilizada por la entidad de la capa 3 para solicitar de la entidad de la capa 2 la transferencia del mensaje de la capa 3 en ella contenido usando los procedimientos establecidos para las transferencia de información sin acuse de recibo ED-UNIDAD DE DATOS-INDICACION (ED-UA-I): Utilizada por la entidad de la capa 2 para indicar a la entidad de la capa 3 la llegada mediante la transferencia de información sin acuse de recibo del mensaje de la capa 3 en ella contenido.

3.4.3.2 Servicio de transferencia de información con acuse de recibo

La transferencia de información con acuse de recibo se hará mediante un modo de operación en multitrama. A continuación se resumen las características de este servicio:

a) Provisión de conexiones de enlace de datos entre entidades de la capa 3 que permitan la transferencia de unidades de mensaje de la capa 3 (mensajes) con acuse de recibo.

b) Identificación de los puntos extremos de conexión del enlace de datos (IECED) que permita que una entidad de la capa 3 pueda ser identificada por otra de esta misma capa.

c) Integridad de la secuencia de las unidades de mensaje de la capa de enlace de datos cuando no existan errores de funcionamiento del sistema.

d) Notificación a la entidad par interlocutora de la capa 2 de errores de funcionamiento (por ejemplo pérdida de la secuencia).

e) Notificación a la entidad de gestión de capa de errores detectados y no recuperables por la capa de enlace de datos.

f) Control de flujo.

Las primitivas asociadas con el servicio de transferencia de información con acuse de recibo son las siguientes:

a) Transferencia de datos:

ED-DATOS-PETICION (ED-DA-P): Utilizada por la entidad de la capa 3 para solicitar de la entidad de la capa 2 la transferencia del mensaje de la capa 3 en ella contenido según los procedimientos establecidos para transferencia de información con acuse de recibo.

ED-DATOS-INDICACION (ED-DA -I): Utilizada por la entidad de la capa 2 para indicar a la entidad de la capa 3 la llegada mediante la transferencia con acuse de recibo del mensaje de la capa 3 en ella contenido.

b) Establecimiento del modo de operación multitrama:

ED-ESTABLECIMIENTO-PETICION (ED-EST-P): Utilizada por la entidad de la capa 3 para solicitar de la entidad de la capa 2 el establecimiento del modo de operación multitrama entre dos puntos de acceso al servicio mediante los procedimientos adecuados.

ED-ESTABLECIMIENTO-INDICACION (ED-EST-I): Utilizada por la entidad de la capa 2 para indicar a la entidad de la capa 3 el resultado de la solicitud de establecimiento del modo de operación.

ED-ESTABLECIMIENTO-CONFIRMACION (ED-EST-C): Utilizada por la entidad

de la capa 2 para confirmar a la entidad de la capa 3 que el establecimiento de la conexión de enlace de datos iniciado mediante la primitiva ED-ESTABLECIMIENTO-PETICION se ha llevado a cabo con éxito.

c) Finalización del modo de operación multitrama:

ED-LIBERACION-PETICION (ED-LIB-P): Utilizada por la entidad de la capa 3 para solicitar de la entidad de la capa 2 el intento de finalización del modo de operación multitrama entre dos PAS mediante los procedimientos adecuados.

ED-LIBERACION-INDICACION (ED-LIB-I): Utilizada por la entidad de la capa 2 para indicar a la entidad de la capa 3 el intento de finalización del modo operación entre dos PAS como resultado de un funcionamiento defectuoso de la capa 2.

ED-LIBERACION-CONFIRMACION (ED-LIB-C): Utilizado por la entidad de la capa 2 para confirmar a la entidad de la capa 3 que el intento de liberación de la conexión de enlace de datos iniciado mediante la primitiva ED-LIBERACION-PETICION ha sido llevado a cabo con éxito

3.4.4 SERVICIOS OFRECIDOS A LA ENTIDAD DE GESTION DE CAPA DE LA CAPA DE ENLACE DE DATOS

La entidad de la capa de enlace de datos proporciona a la entidad de gestión de capa de la capa de enlace de datos un servicio de transferencia de información sin acuse de recibo y un servicio administrativo relacionado con la gestión de los valores IET.

3.4.4.1 Transferencia de información sin acuse de recibo

La entidad de la capa de enlace de datos proporciona a la entidad de gestión de capa un servicio de transferencia de información sin acuse de recibo para permitir una comunicación entre las entidades de gestión de capa pertenecientes a sistemas diferentes y que están relacionadas por una conexión de enlace de datos. Dado que en el caso de acceso primario de usuario no se requieren procedimientos entre pares de las entidades de gestión de capa de la capa de enlace de datos, este servicio

solamente es aplicable al caso de acceso básico.

Las características de este servicio pueden resumirse en las tres siguientes:

a) Provisión de conexiones de enlace de datos entre entidades de gestión de capa de la capa 2 que permitan la transferencia de unidades de datos sin acuse de recibo.

b) Identificación de punto extremo de conexión de enlace de datos (IECED) que permita que una entidad de gestión de capa pueda ser identificada por otra de esta misma capa.

c) Las entidades de la capa de enlace de datos no llevan a cabo ningún tipo de verificación de las unidades de mensaje de capa 2 recibidas.

Las primitivas asociadas con este servicio son:

GED-UNIDAD DE DATOS-PETICION (GED-UDA-P): Utilizada por la entidad de gestión de capa de la capa 2 para solicitar de la entidad de la capa 2 la transferencia de la unidad de mensaje de gestión en ella contenida, usando los procedimientos establecidos para la transferencia de información sin acuse de recibo, para la gestión de capa.

GED-UNIDAD DE DATOS-INDICACION (GED-UDA-I): Utilizada por la entidad de la capa de enlace de datos para indicar a la entidad de gestión de capa de la capa 2 de la llegada mediante la transferencia de información sin acuse de recibo de la unidad de mensaje de gestión en ella contenida.

3.4.4.2 Servicios administrativos

La función de este tipo de servicios es ocuparse de las tareas relacionadas con la asignación, prueba y supresión de los Identificadores de Equipo Terminal (IET). Estos servicios de la capa 2 conceptualmente están pensados para ser ofrecidos por las entidades de gestión de capa tanto del lado de usuario como del lado de red.

Las primitivas relacionadas con estos servicios son las que se señalan a continuación:

a) Asignación del valor IET:

GED-ASIGNACION-PETICION (GED-ASIG-P): Utilizada por la entidad de gestión de capa para pasar el valor IET en ella contenido a la entidad de la capa 2 de modo que las entidades de la capa 2 del usuario puedan empezar a comunicarse con las entidades de la capa 2 de la red.

GED-ASIGNACION-INDICACION (GED-ASIG-I): Utilizada por la entidad de la capa 2 para indicar a la entidad de gestión de capa la necesidad de asignación de un valor IET.

b) Supresión del valor IET:

GED-SUPRESION-PETICION (GED-SUP-P): Utilizada por la entidad de gestión de capa para solicitar de la entidad de la capa 2 la supresión de un valor IET que ha sido previamente asignado mediante una primitiva GED-ASIG-P.

c) Notificación de errores:

GED-ERROR-INDICACION (GED-E-I): Utilizada por la entidad de la capa 2 para notificar a la entidad de gestión de capa que ha tenido lugar un error de procedimiento que no puede ser recuperado por la propia capa de enlace de datos.

GED-ERROR-RESPUESTA (GED-E-R): Utilizada por la entidad de gestión de capa cuando no puede obtener un valor IET requerido por una entidad de la capa 2. Esta primitiva no es aplicable al caso de acceso primario usando permanentemente el IET de valor cero (el valor IET es mantenido permanentemente por la entidad de gestión de la capa 2 aunque la propia entidad de la capa 2 lo haya perdido).

3.4.5 PROCEDIMIENTOS DE PRIMITIVAS

3.4.5.1 Modelo de servicio de enlace de datos

La capacidad de la capa de enlace de datos de proporcionar un servicio solicitado por la capa 3 dependerá del estado interno de la capa de enlace de datos. Para cada una de las entidades de capa 3, el estado interno de la capa de enlace de datos está representado por el estado del punto extremo de conexión de enlace de

datos dentro del punto de acceso al servicio que es utilizado por dicha entidad de capa 3 para solicitar el servicio. (Estos estados del punto extremo de conexión de enlace de datos, aunque son independientes, están muy relacionados y pueden derivarse de los estados de las entidades de la capa de enlace de datos descritos en el 3.4.2.3).

Por tanto, el servicio de enlace de datos puede ser definido mediante el uso de los estados del punto extremo de conexión de enlace de datos, pudiendo relacionarse con dichos estados las capacidades proporcionadas por la capa 2 y las primitivas de servicio intercambiadas.

Los posibles estados de los puntos extremos de conexión pueden ser los siguientes en función del tipo de conexión de enlace de datos.

a) Servicios de punto extremo de conexión de enlace de datos de difusión.

Una conexión de enlace de datos de difusión proporcionará un servicio de transferencia de información sin acuse de recibo. Dentro de cada punto de acceso al servicio solamente existirá un punto extremo de conexión de enlace de datos de difusión que se encontrará permanentemente en el estado "Transferencia de información".

b) Servicios de punto extremo de conexión de enlace de datos punto a punto.

Una conexión de enlace de datos punto a punto proporcionará tanto el servicio de transferencia de información sin acuse de recibo como el de transferencia de información con acuse de recibo. Dentro de cada punto de acceso al servicio existirán uno o más puntos extremo de conexión de enlace de datos cada uno de ellos identificados por un Sufijo de punto Extremo de Conexión (SEC) diferente.

Además, la transferencia de información con acuse de recibo requiere la presencia de los servicios de capa 2 relacionados con el establecimiento, restablecimiento y liberación de las conexiones de enlace de datos.

Los posibles estados de los puntos extremo de conexión de enlace de datos

punto a punto son los siguientes:

-Conexión de enlace liberada

-Establecimiento pendiente

-Liberación pendiente

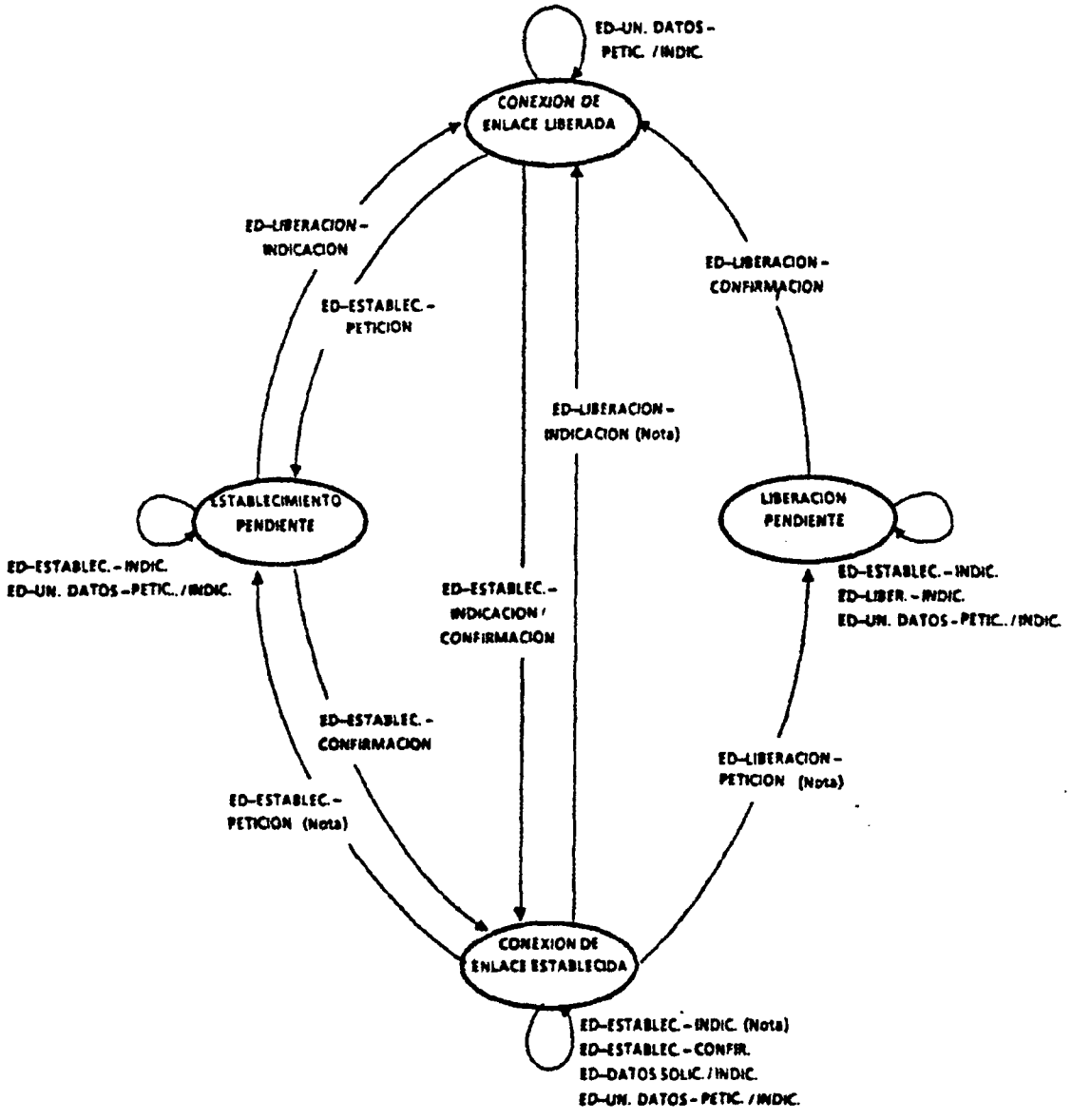
-Conexión de enlace establecida

3.4.5.2 Secuencia de primitivas en un punto extremo de conexión de enlace de datos punto a punto

Las primitivas proporcionarán los elementos y procedimientos necesarios para definir de un modo conceptual cómo un usuario de los servicios de capa 2 puede solicitarlos.

Las primitivas relacionadas con los servicios pueden ser intercambiadas de acuerdo con unas secuencias establecidas que determinan las transiciones entre los cuatro estados definidos anteriormente. Este intercambio de primitivas es el representado en el diagrama de transición entre estados de la figura 3.7 , donde los estados de conexión de enlace establecida y liberada son estados estables mientras que los estados de establecimiento y liberación pendiente son estados de transición.

El modelo mostrado en dicha figura 3.7 ilustra el comportamiento de la capa de enlace de datos tal como es observado por la capa 3. Este modelo asume que las primitivas entre las capas 2 y 3 son transmitidas a través de dicho interfaz entre capas en el mismo orden en que han sido generadas. En dicho modelo podrían tener lugar colisiones entre primitivas del tipo PETICION y del tipo INDICACION dando lugar a situaciones no previstas, de modo que dicho comportamiento deberá ser evitado por los diseños de las realizaciones práctica del protocolo del canal D.



Nota: Puede tener lugar pérdida de información.

Figura 3.7 DIAGRAMA DE TRANSICIÓN ENTRE ESTADOS DE LOS PUNTOS EXTERMOS DE CONEXIÓN DE ENLACE DE DATOS PUNTO A PUNTO

3.5 ESTRUCTURA DE LA CAPA DE ENLACE DE DATOS

De acuerdo con la estructura general mostrada en la figura 1.2 , en la figura 3.8 se muestra un diagrama funcional de bloques de la capa de enlace de datos,

diagrama que se indica solamente con carácter ilustrativo sin que ello represente ninguna limitación a las posibles realizaciones de los sistemas que las soporten.

El diagrama funcional mencionado muestra tanto las entidades de la capa 2 (C2) como la entidad de gestión de capa de la capa 2 (C2) como entidad compuesta a su vez por la entidad de gestión de capa (encargada de los procedimientos de gestión de los identificadores de equipo terminal) y por la entidad de gestión de la conexión (encargada del soporte de los parámetros del sistema así como del procesamiento de errores del protocolo). Dicho modelo funcional de la capa de enlace de datos se simplifica para el caso de acceso primario de usuario teniendo en cuenta la inexistencia del enlace de datos de difusión y la utilización de una única conexión de enlace de datos, por lo que el procedimiento múltiples no será incluido.

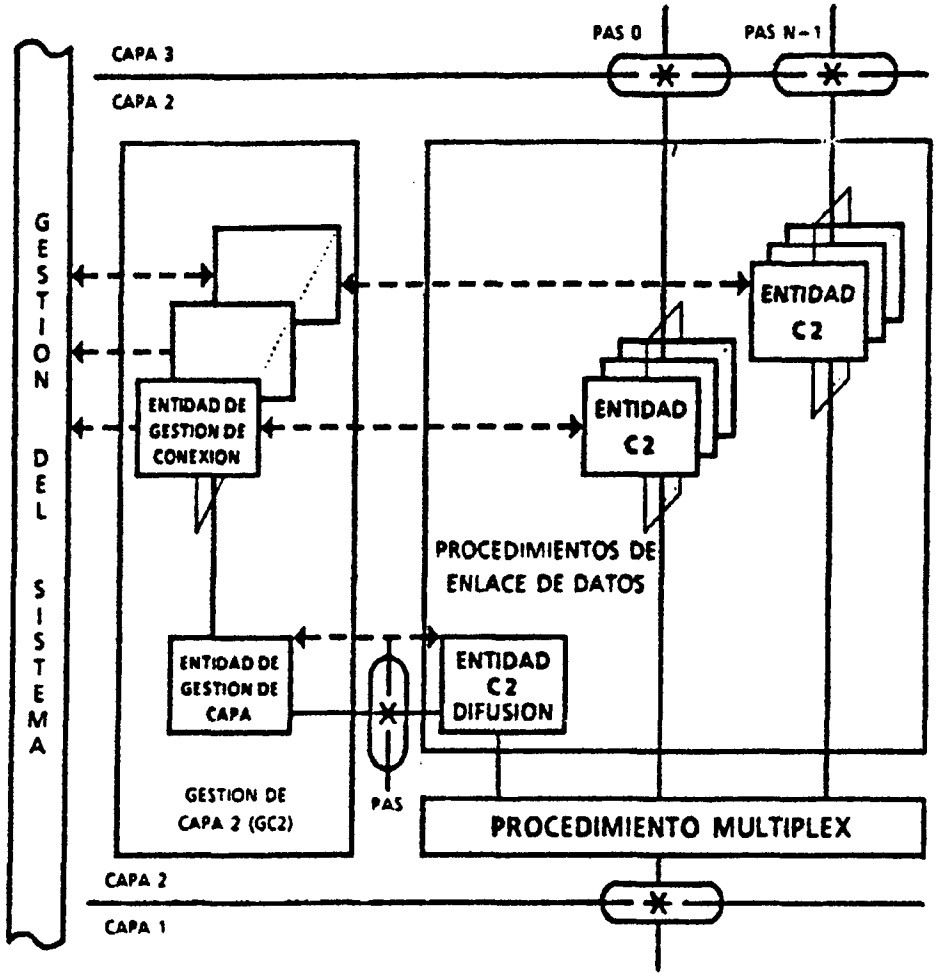


Figura 3.8 . MODELO FUNCIONAL DE LA CAPA DE ENLACE DE DATOS.

En este modelo la entidad de gestión de capa proporciona las funciones de gestión de recursos que conllevan una implicación general de la capa 2. El acceso a estas funciones requiere la utilización de un IPAS específico. Las funciones proporcionadas por la entidad de gestión de capa están relacionadas con las gestiones de IET: asignación, comprobación y supresión de valores IET.

Las entidades de gestión de la conexión proporcionan funciones para la gestión de recursos que tienen un impacto en las conexiones individuales, como la iniciación de los parámetros (en caso de que existiese procesamiento de errores y control de flujo).

El procedimiento de enlace de datos analiza el campo de control de las tramas recibidas y proporcionará las adecuadas tramas respuestas, así como las indicaciones necesarias entre capas.

El procedimiento multiplex analiza los indicadores, secuencia de verificación de trama y campo de dirección de las tramas recibidas. Si la trama es correcta, será encaminada hacia el bloque de procedimiento de enlace de datos adecuado en función del identificador de conexión de enlace de datos contenido en el campo de dirección de la trama. Durante la transmisión de tramas este procedimiento proporcionará la resolución de la contención de enlace de datos en función del IPAS dando prioridad a la información de señalización.

En la figura 3.9 , se muestra un modelo funcional de los procedimientos de enlace de datos, constituido por varios bloques funcionales para las conexiones punto a punto y las de difusión.

3.6 IDENTIFICACION DE LA CONEXION DE ENLACE DE DATOS

Un Punto de Acceso al Servicio (PAS) de la capa de enlace de datos es el medio por el cual las entidades de esta capa proporcionan los servicios a las entidades de la

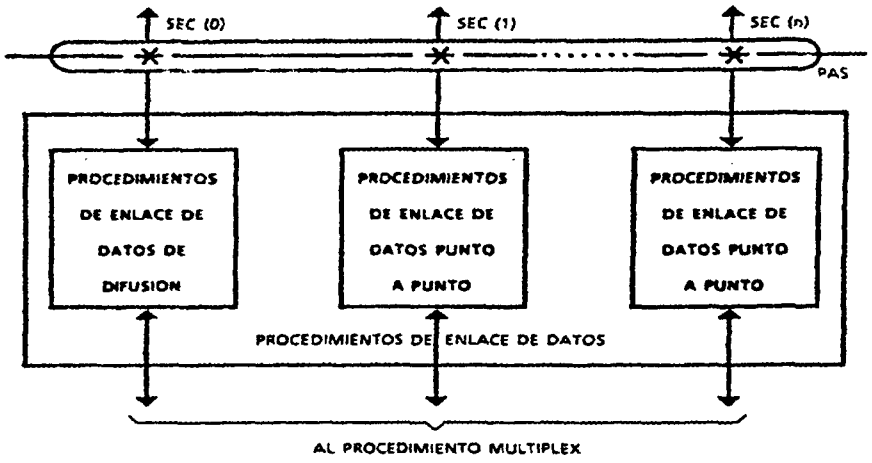


Figura 3.9 . ESTRUCTURA DE LOS PROCEDIMIENTOS DE ENLACE DE DATOS.

capa 3. Asociado con cada PAS de la capa 2 existe uno o más puntos Extremo de Conexión de Enlace de Datos (ECED).

Tal como se indica en la figura 3.10 un punto extremo de conexión de enlace de datos está identificado por un Identificador de punto Extremo de Conexión Enlace de Datos (IECED) visto desde la capa 3 y por un Identificador de Conexión de Enlace de Datos (ICED) visto desde la propia capa 2, identificador que es transportado en el campo de dirección de cada una de las tramas. El hecho de que cada punto extremo de conexión de enlace de datos esté identificado tanto desde la capa 3 como desde la capa 2, implicará que exista una relación entre ambos identificadores, que vendrá dada por la relación entre el SEC de capa 3 y el IET de capa 2.

Cada conexión de enlace de datos estará identificada por un ICED que será el mismo en los dos o más extremos de la conexión, permitiéndose así el intercambio de información entre entidades de la capa 3.

El identificador de conexión de enlace de datos (ICED) está asociado con un identificador de punto extremo de conexión en los dos extremos del enlace de datos. El identificador de punto extremo de conexión (IECED) es utilizado para la identificación de los mensajes intercambiados entre las entidades de la capa 2 y de la

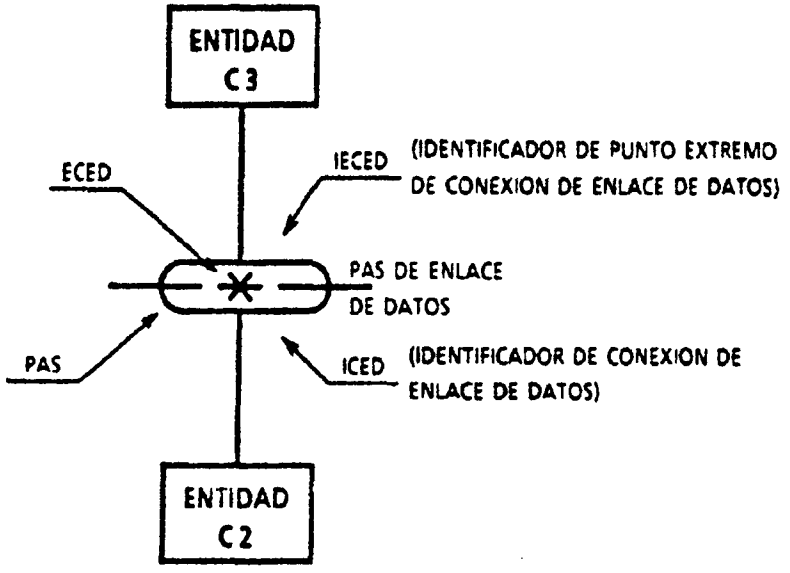


Figura 3.10 . PUNTO DE ACCESO AL SERVICIO DE LA CAPA DE ENLACE DE DATOS.

capa 3 y está compuesto por el identificador del punto de acceso al servicio (IPAS) y el Sufijo de punto Extremo de Conexión (SEC) tal como se indica en la figura 3.11 (IECED = IPAS+SEC).

El identificador de conexión de enlace de datos (ICED) se utiliza para la identificación de las tramas intercambiadas en el protocolo LAPD y está compuesto por el identificador de punto de acceso al servicio (IPAS) y el identificador de equipo terminal (IET) como se indica en la figura 3.11 (ICED= IPAS+IET).

El SEC será un valor seleccionado por la entidad de capa 3 o entidad de gestión para direccionar la entidad de la capa de enlace de datos. Cuando esta entidad conozca el valor IET utilizado por la capa de enlace de datos, se establecerá internamente una asociación entre ambos valores, y por tanto entre el IECED y el ICED.

El IPAS es utilizado para identificar el punto de acceso al servicio de enlace de datos en ambos lados del interfaz usuario/red, mientras que el IET identifica la conexión final de un equipo terminal del usuario específico a un PAS determinado, es decir este identificador (IET) tiene solamente significado asociado a su PAS, pudiendo

existir por tanto valores de IET iguales pero asociados a PAS diferentes (para el caso de terminales, de la categoría de asignación de IET). El IET puede ser asignado de un modo automático por la red mediante un procedimiento de gestión de gestión de la capa 2 o puede ser asignado previamente a la instalación del equipo de usuario quedando asignado de un modo unívoco (para el caso de terminales de la categoría de asignación no automática). En este segundo caso el IET podrá ser verificado por la capa 2 del lado de usuario antes de su utilización para asegurarse de que este valor no está siendo utilizado simultáneamente por varios equipos de usuario.

Se puede mencionar como caso particular el valor cero del IET preasignado para TR2 inteligentes en conexiones de enlace de datos punto a punto (interfaz de usuario de velocidad primaria). Este valor no usará el procedimiento de asignación de valor IET ni tendrá que ser verificado.

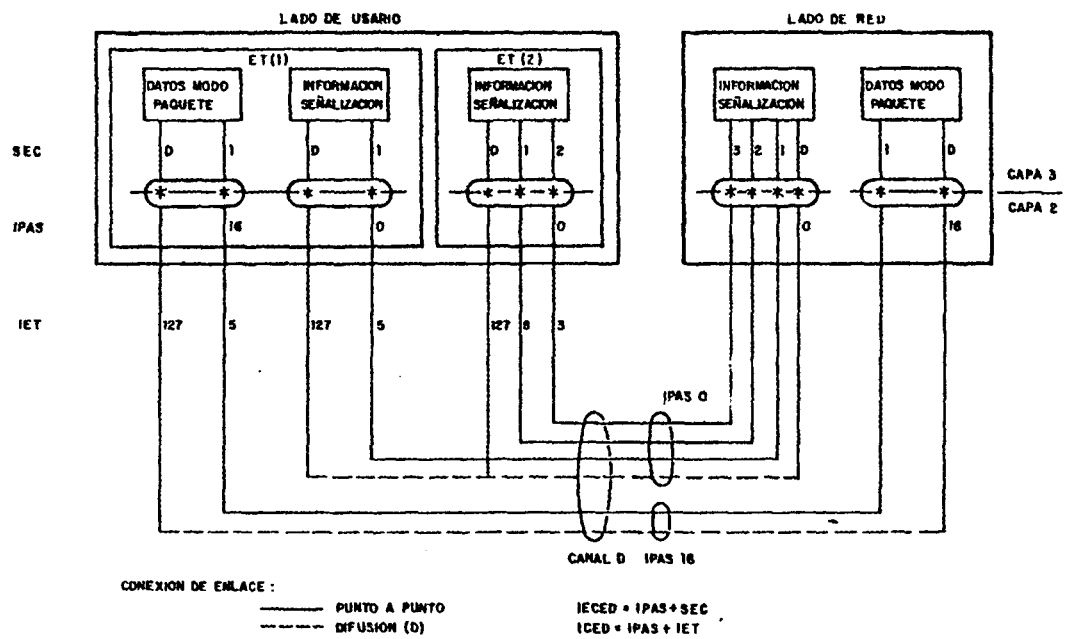
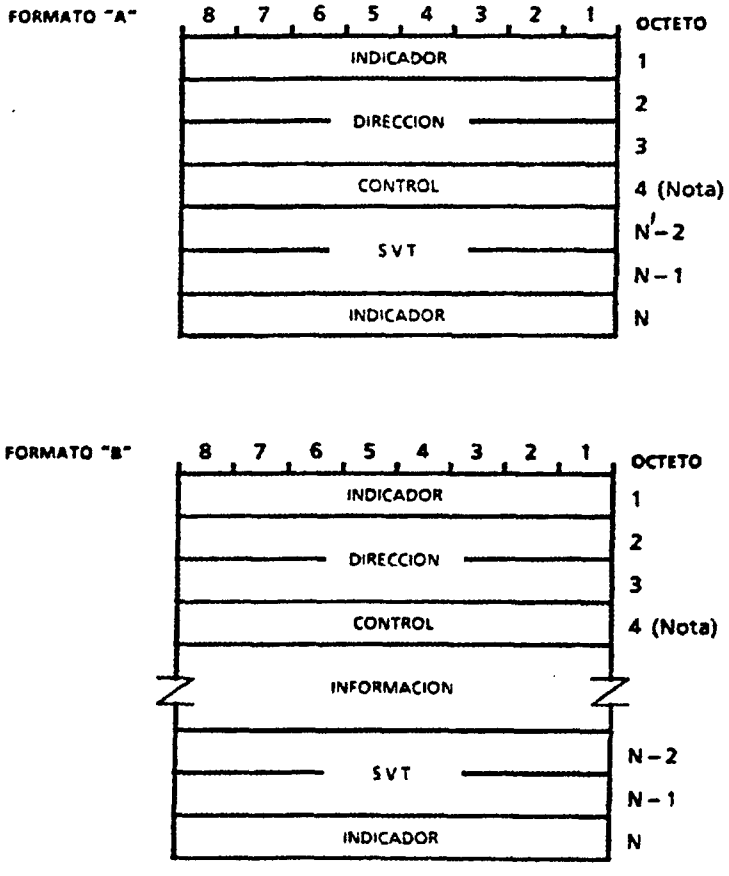


Figura 3.11 . IDENTIFICACIÓN DE LA CONEXIÓN DE ENLACE DE DATOS

3.7 ESTRUCTURA DE LA TRAMA

Todas las tramas intercambiadas entre las entidades de la capa 2 durante el protocolo LAP D se ajustarán a uno de los dos formatos que se muestran en la figura 3.12 , diferenciándose un formato "A" caracterizado por la ausencia de campo de información que sí aparece en las tramas con formato "B".



Nota: Operación sin acuse de recibo: 1 octeto
Modo multitrama (módulo 128): 2 octetos, tramas con números de secuencia.
1 octeto, tramas sin números de secuencia.

Figura 3.12 . ESTRUCTURA DE TRAMA.

Todas las tramas comenzarán y terminarán con una secuencia de bits consistente en un bit "0" seguido de seis bits "1" y cerrada por otro "0" constituyendo un octeto que se denomina indicador, siendo el indicador de apertura el que forma el primer octeto de la trama y el de cierre el que marca final de ésta. En algunos casos el indicador de cierre puede actuar como indicador de apertura de la siguiente trama. Sin embargo, todas las entidades de capa 2 deben estar preparadas para recibir uno o más indicadores consecutivos. Los dos octetos que siguen al indicador de apertura son los dedicados al campo de dirección de la trama. El campo de control de las tramas está formado por uno o dos octetos en el que se definen diferentes subcampos que permitirán la identificación de cada una de las tramas empleadas en el modo de operación sin acuse de recibo y multitrama, así como la numeración de las tramas que permite la existencia de varias tramas pendientes de reconocimiento en un momento y el control de secuencia en el modo multitrama.

-El campo de información en el caso de su existencia, estará compuesto por un número entero de octetos, N201 (el valor del parámetro N201 es 260 octetos).

Los dos octetos anteriores al indicador de cierre constituyen la secuencia de verificación de trama (SVT) consistente en un conjunto de 16 bits cuyo análisis permitirá detectar los errores de transmisión de las tramas.

3.7.1 TRANSPARENCIA

Las entidades de la capa de enlace de datos examinarán el contenido de las tramas a transmitir, entre el indicador de apertura y el de cierre (campos de dirección, control, información y SVT) e insertarán un bit "0" después de todas las secuencias de cinco bits "1" consecutivos (incluyendo los cinco últimos bits de la SVT) para asegurar que no se produce una simulación de indicador o de anulación de trama.

Las entidades de la capa de enlace de datos examinarán el contenido de las tramas recibidas entre los indicadores y eliminaran el bit "0" incluido después de cada secuencia de cinco bits "1" consecutivos.

3.7.2 SECUENCIA DE VERIFICACION DE TRAMA

La SVT será una secuencia de 16 bits generada por el transmisor de la trama tras el análisis de su contenido y se obtendrá de hacer el complemento a uno de la suma en módulo dos de:

a) El resto de la división en módulo 2 del polinomio:

$$x^K \cdot (x^{15} + x^{14} + x^{13} + x^{12} + x^{11} + x^{10} + x^9 + x^8 + x^7 + x^6 + x^5 + x^4 + x^3 + x^2 + x^1 + 1)$$

por el polinomio generador

$$G(x) = x^{16} + x^{12} + x^5 + 1$$

Donde "K" es el número de bits de la trama contenidos entre el indicador de apertura y el comienzo del campo dedicado a contener la SVT, sin incluir los bits insertados para conseguir la transparencia tal como se indica en la figura 3.13 .

Figura 3.13 GENERADOR DE LA SVT.

INDICADOR	DIRECCIÓN	CONTROL	INFORMA-CIÓN	S.V.T.	INDICADOR
	<----------	T(x), (K bits)	---------->		

De este modo:

$$\frac{x^K \cdot (x^{15} + x^{14} + \ldots + x^2 + x^1 + 1)}{G(x)} = D_1(x) + \frac{R_1(x)}{G(x)}$$

donde R_1 (x) será el polinomio resto aquí buscado.

b) El resto de la división en módulo 2 por el polinomio generador G(x) del contenido de la trama entre el indicador de apertura y el comienzo del campo dedicado a contener la SVT, sin incluir los bits insertados para conseguir la transparencia después de su multiplicación por x^{16}

De este modo:

$$\frac{x^{16} \cdot T(x)}{G(x)} = D_2 \cdot (x) + \frac{R_{2(x)}}{G(x)}$$

donde $R_2(x)$ será el polinomio resto aquí buscado. Por tanto la secuencia de verificación de trama será:

$$SVT = \overline{R_1(x) + R_2(x)}$$

Como caso práctico típico, en el transmisor, el contenido inicial del registrador del dispositivo calculando el resto de la división, se fija a "todos unos" y se modifica luego por la división por el polinomio generador G (x) (como se ha descrito anteriormente) de los campos de dirección, control e información T (x); el complemento a uno del resto resultante se transmite como SVT de 16 bits.

Como caso práctico, en el receptor, el contenido inicial del registrador del dispositivo calculando el resto de la división se fija a "todos unos". El resto final resultante después de la multiplicación por x^{16} y la división en módulo dos por el polinomio generador G (x) de la serie de bits protegidos entrantes y la SVT dará lugar a un resto "00011101 00001111" (x^{15} a x^0, respectivamente) en ausencia de errores de transmisión.

3.7.3 FORMATO DE LAS TRAMAS

3.7.3.1 Convenio de numeración

La numeración utilizada es la que se representa en la figura 3.14 , en la que se muestra como los bits se agrupan en octetos estando aquellos numerados de 1 al 8, mientras que los octetos, que se presentan verticalmente, se numeran en sentido creciente comenzando por el 1 hasta el n..

Cuando un campo ocupa un número entero de octetos, la significación de los bits decrece según se incrementa la numeración de los octetos, tal como se indica en la figura 3.15 donde se muestra un campo de dos octetos.

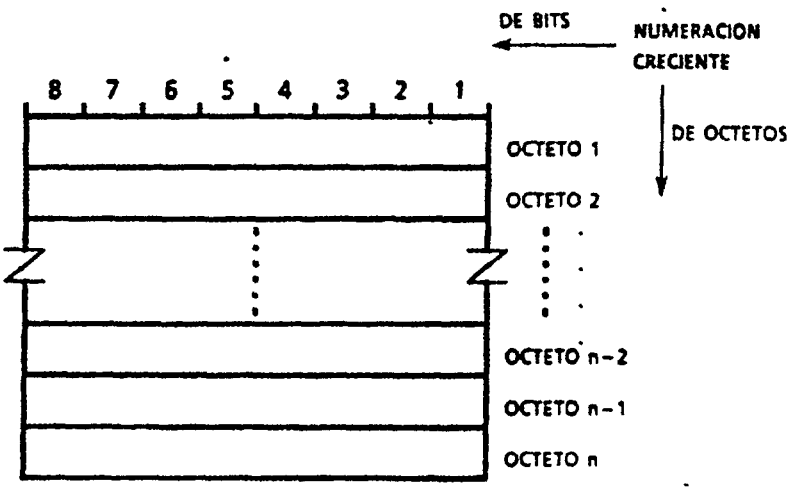


Figura 3.14 . CONVENIO DE NUMERACIÓN

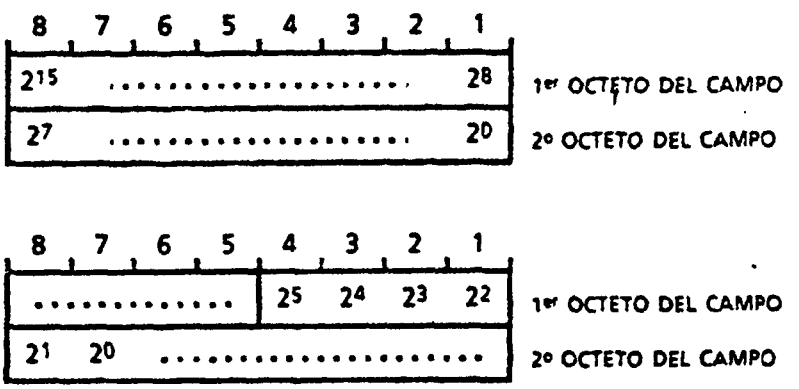


Figura 3.15 REPRESENTACIÓN DE CAMPOS.

Una excepción a este convenio es el campo de SVT que ocupa dos octetos, en este caso los bits menos significativos dentro de cada octeto son los de valor numérico superior siendo el segundo octeto menos significativo que el primero tal como se representa en la figura 3.16 .

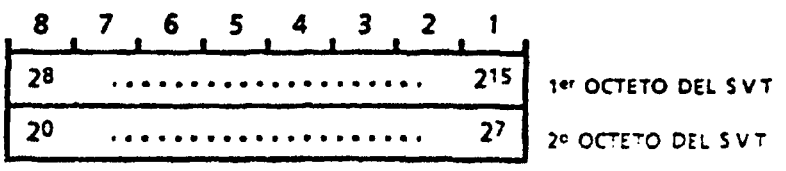


Figura 3.16 REPRESENTACIÓN DE LA SVT

3.7.4 TRAMAS NO VALIDAS

Una trama no será válida si cumple al menos una de las condiciones que se indican a continuación:

a) No está delimitada correctamente por indicadores.

b) Existen menos de seis octetos entre indicadores en el caso de tramas que contienen números de secuencia o menos de cinco octetos entre indicadores en el caso de tramas que no contienen número de secuencia.

c) No constituye un número entero de octetos antes de las inserción de ceros o después de su extracción.

d) Existe un error en la SVT.

e) Contiene un campo de dirección de un único octeto.

f) Contiene un identificador de punto de acceso al servicio (IPAS) que no es soportado por el receptor de la trama

Las tramas no válidas serán desechadas por su receptor sin que se envíe notificación alguna al transmisor de las mismas. No se llevará a cabo ninguna acción como resultado de la recepción de una trama no válida.

Las tramas recibidas que estén relacionadas con una aplicación no soportada, aunque no son tramas inválidas, serán tratadas como éstas.

3.7.5 ANULACION DE TRAMAS

La recepción de una secuencia de 7 o más unos consecutivos hará que la entidad de la capa de enlace de datos desprecie la trama que actualmente se esté recibiendo.

3.8 CODIFICACION DE LAS TRAMAS Y ELEMENTOS DEL PROTOCOLO LAP D

Los elementos del protocolo LAPD definen los comandos y respuestas que se utilizan en los procedimientos usados para el control de las conexiones de enlace de datos y que son transmitidos sobre un canal D. Estos procedimientos se describen en

secciones posteriores.

3.8.1 FORMATO Y VARIABLES DEL CAMPO DE DIRECCION

El campo de dirección contenido en todas las tramas, está constituido por dos octetos y tiene por misión la identificación del receptor de una trama comando así como el transmisor de una trama respuesta.

Este campo de dirección está estructurado en dos octetos tal como se muestra en lafigura 3.17 , donde se puede ver la existencia de un bit de extensión del campo de dirección (ED) en cada octeto, un bit indicador de comando/respuesta (C/R), el indicador del punto de acceso al servicio de la capa de enlace de datos (IPAS) y el identificador de equipo terminal (IET).

La opción de utilizar un campo de dirección de un solo octeto está reservada para la posibilidad de soportar el modo de operación LAP B (capa de enlace de datos X.25), con el fin de permitir la multiplexación de una conexión de enlace de datos LAP B con conexiones de enlace de datos LAP D..

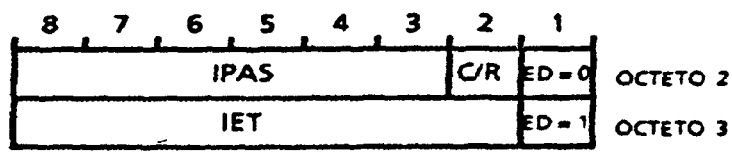


Figura 3.17 . FORMATO DEL CAMPO DE DIRECCIÓN

3.8.1.1 Bit de extensión del campo de dirección (ED)

Este bit, el primero de cada uno de los octetos, tiene por misión la delimitación del campo de dirección indicando el octeto final. El hecho de que el campo de dirección esté estructurado en dos octetos hace que el bit ED del primer octeto de este campo (segundo de la trama) esté siempre codificado con valor "0" indicando que existe un octeto más del campo de dirección, mientras que el bit ED del segundo octeto del campo se codifica con valor "1" para indicar que se trata del último octeto del campo.

3.8.1.2 Bit comando/respuesta (C/R)

Este bit del campo de dirección identifica cada una de las tramas del protocolo LAP D como un comando o una respuesta. El lado de usuario enviará los comandos con el bit C/R codificado a "0" y las respuestas con este bit puesto a "1". El lado de red hará lo contrario enviando el bit C/R codificado a "1"ó "0" según que la trama sea un comando o una respuesta respectivamente. Estas codificaciones se muestran en la tabla de la figura 3.18 .

Figura 3.18 CODIFICACIÓN DEL BIT C/R

	COMANDO	RESPUESTA
lado de red, hacia el lado de usuario	1	0
lado de usuario, hacia el lado de red	0	1

Del repertorio de tramas que se utilizan en el protocolo LAP D existen unas tramas que solamente pueden ser transmitidas como comandos y otras como respuestas, por lo que en estos casos este bit proporcionaría una información redundante sobre la proporcionada por el propio tipo de trama. En cambio, este bit será esencial en el caso de aquellas tramas que pueden ser transmitidas tanto con funciones de comando como de respuesta.

Hay que tener en cuenta que las tramas comando se caracterizan porque el contenido de su campo de dirección identifica la entidad de la capa 2 de destino de las tramas, mientras que el de las tramas respuesta identifica la entidad de capa 2 de origen.

De acuerdo con las normas establecidas en 3.6 para la identificación de las conexiones de enlace de datos, las dos entidades de la capa de enlace de datos relacionadas por una conexión de enlace de datos punto a punto utilizarán el mismo identificador de conexión de enlace de datos (ICED) compuesto por un IPAS y un IET, identificador que a su vez identificará dicha conexión de enlace de datos.

3.8.1.3 Identificador de punto de acceso al servicio (IPAS)

El IPAS identifica un punto por el cual una entidad de la capa de enlace de datos proporciona servicios a una entidad de la capa 3 o de gestión de capa. Por tanto el IPAS permite identificar la entidad de la capa 2 que procesa la trama y también la entidad de la capa 3 o de gestión de capa que recibirá la información en ella contenida.

EL campo IPAS permite identificar 64 PAS diferentes, codificado de modo que ocupa los 6 bits más significativos del primer octeto del campo de dirección siendo el bit 3 de dicho octeto el menos significativo del IPAS. Los valores de IPAS están asignados del modo siguiente:

Valor IPAS	Entidad de capa 3 o de gestión de capa relacionada
0	Procedimientos de control de llamada
1	Comunicaciones modo paquete usando los procedimientos de control de llamada de la capa 3 del protocolo de canal D
16	Comunicaciones modo paquete de acuerdo con los procedimientos de capa 3 X.25.
63	Procedimientos de gestión de capa de la capa 2.
resto	Normalización futura

3.8.1.4 Identificador de equipo terminal (IET)

El identificador de equipo terminal para una conexión de enlace de datos punto a punto estará asociado con un único equipo terminal, pero un equipo terminal puede contener uno o más identificadores de equipo terminal (del tipo punto a punto). El IET para una conexión de enlace de datos tipo difusión estará asociado a todas las entidades del enlace de datos que contengan el mismo IPAS.

Los 7 bits del segundo octeto del campo de dirección dedicados al IET permiten 128 valores diferentes, siendo el segundo bit del octeto el bit menos significativo.

La asignación de los valores del IET asociados al IPAS direccionado es la que

se indica a continuación:

valor IET	
0	Soportar una TR2 inteligente en configuración punto a punto usando una única conexión de enlace de datos punto a punto en los PAS. (Este valor IET será siempre utilizado asociado a un acceso de usuario de jerarquía primaria, 30B+D).
1-63	Conexiones de enlace de datos punto a punto. Equipos de usuario de asignación no automática de IET.
64-126	Conexiones de enlace de datos punto a punto. Equipos de usuario de asignación automática de IET
127	Conexión de enlace de datos de difusión, IET de grupo

De acuerdo con esta asignación de valores IET, todo equipo terminal conectado a un acceso básico de usuario, además de un valor o valores IET particulares, deberá soportar el valor IET de grupo para el establecimiento de conexiones de enlace de datos de difusión. De un modo similar, el lado de red deberá soportar todos los valores IET asignados o las terminales así como el valor IET de grupo.

Como se ha indicado en la tabla anterior se define un procedimiento de asignación automática de IET, caracterizado porque los valores IET son seleccionados y asignados por la red, y un procedimiento de asignación no automática de IET, en el que los valores IET son seleccionados por los propios equipos terminales siendo responsabilidad de la red su asignación definitiva.

3.8.2 FORMATO, PARAMETROS Y VARIABLES ASOCIADAS DEL CAMPO DE CONTROL.

El campo de control contenido en las tramas identifica de un modo unívoco la trama que se está transmitiendo y que podrá ser un comando o una respuesta tal como se indique mediante el bit C/R del campo de dirección. Este campo además contendrá un bit P/F y el número de secuencia de las tramas cuando sea necesario.

Se definen tres tipos diferentes de formato de campo de control:

Formato I: transferencia de información numerada

Formato S: funciones de supervisión

Formato U: transferencia de información no numerada y funciones de control

Estos formatos básicos se muestran en la figura 3.19 para el caso de funcionamiento en modo extendido multitrama (el calificativo de "extendido" surge como consecuencia del módulo de funcionamiento utilizado, 128, tal como se indica en 3.8.3.1).

Figura 3.19 FORMATOS DEL CAMPO DE CONTROL

	BITS DEL CAMPO DE CONTROL								Números De Octetos Del Campo De Control
	8	7	6	5	4	3	2	1	
formato I				N(T)				0	2
				N(R)				P	
formato S	X	X	X	X	S	S	0	1	2
				N(R)				P/F	
formato U	M	M	M	P/F	M	M	1	1	1

N(T): número de secuencia en transmisión

N(R): número de secuencia en recepción.

P/F: bit de petición cuando sea enviado en un comando y bit final cuando sea enviado en una respuesta.

S: bits de función de supervisión.

M: bits de modificación de función.

X: bits reservados puestos a "0"

a) Formato I:

Este formato de trama se utiliza para llevar a cabo la transferencia de información entre entidades de la capa 3. Estas tramas contienen los parámetros de número de secuencia en transmisión N(T) y en recepción N(R) así como el bit P cuyas funciones son independientes. En estas tramas el N(R) contenido puede representar o no el acuse de recibo de nuevas tramas I y el bit P puede estar codificado como "0" ó "1"

b) Formato S:

Este formato de trama se utiliza para llevar a cabo funciones de supervisión y control del enlace de datos tal como el acuse de recibo de tramas I, petición de retransmisión de tramas I o de petición de una suspensión temporal de la transmisión de tramas I. Estas tramas contienen los parámetros N(R), que puede representar o no el acuse de recibo de nuevas tramas I, y P/F, que puede estar codificado como "0" ó "1", cuyas funciones son independientes. Además existen dos bits de función de supervisión (notados como "S" en la figura 3.19 cuya misión es identificar de un modo unívoco el tipo particular de comando o respuesta que se está transmitiendo.

c) Formato U:

Este formato de trama se utiliza para proporcionar funciones adicionales de control del enlace de datos, así como transferencias de información no numerada relacionadas con la transferencia de información sin acuse de recibo. Estas tramas no poseen número de secuencia pero sí el bit P/F, que puede estar codificado como "0" ó "1", y una serie de bits de modificación de función (notados como "M" en la figura 3.19) que identifican de un modo unívoco el tipo particular de comando o respuesta que se está transmitiendo.

3.8.3 PARAMETROS DEL CAMPO DE CONTROL Y VARIABLES DE ESTADO ASOCIADAS

En este apartado se describen los diferentes parámetros asociados con el campo de control de las tramas de la capa de enlace de datos, así como las variables de estado asociadas a dichos parámetros. La codificación de los bits, de acuerdo con el convenio de representación de campos mostrado en 3.7.3, está hecha de modo que los bits más bajos dentro del octeto serán los menos significativos del parámetro correspondiente.

3.8.3.1 Módulo

Cada una de las tramas I está numerada de un modo secuencial desde el valor

cero hasta el valor "n-1", donde "n" es el valor del módulo de los números de secuencia.

El valor del módulo para el funcionamiento en modo multitrama en el enlace de datos es 128, por lo que los números de secuencia tomarán los valores en el margen de 0 a 127.

Todas las operaciones aritméticas contenidas en este capítulo relacionadas con el soporte de las variables de estado y números de secuencia, definidos en 3.8.3.3, se verán afectadas por este valor del módulo de operación.

3.8.3.2 Bit Petición/Final (P/F)

Todas las tramas contienen un bit Petición/Final (P/F) que desempeña una función en todas ellas, tanto comando donde se utiliza como bit P como respuesta donde se utiliza como bit F. El bit P puesto a"1" es utilizado por una entidad de la capa 2 para solicitar una trama respuesta de la entidad par de capa 2 remota que la transmitirá con el bit F puesto a "1" para indicar que es el resultado de la recepción del comando de petición.

En el cuadro de la figura 3.20 se muestra la asociación posible entre comandos y respuestas transmitidas con el bit P/F puesto a "1". En el apartado 3.10 se describe el procedimiento de capa 2 definido para la utilización del bit P/F.

Figura 3.20 UTILIZACIÓN DEL BIT P/F

Comandos recibidos con P= 1	Respuestas transmitidas con F = 1
SABME, DISC	UA, DM
I, RR, RNR, REJ	RR, RNR, REJ, DM

3.8.3.3 Operación en modo multitrama. Variables de estado y números de secuencia

a) Variable de estado en transmisión V (T)

Cada punto extremo de conexión de enlace de datos punto a punto mantendrá asociada una variable de estado en transmisión V(T) cuando se están transmitiendo

tramas comando I. Esta variable indica el número de secuencia de la próxima trama I a ser transmitida y podrá tener un valor comprendido entre cero y el módulo menos uno (127). Este contador se incrementar en una unidad después de la transmisión de cada trama I consecutiva y no podrá exceder del valor indicado por la variable de estado del acuse de recibo V(A) en más del número de tramas I que pueden estar pendientes de acuse de recibo (ventana) y que se denota por "k". El valor de "k" deberá estar contenido en el margen $1< k < 127$ para la operación en módulo 128.

b) Variable de estado del acuse de recibo V(A):

Cada punto extremo de conexión de enlace de datos punto a punto mantendrá asociada una variable de estado del acuse de recibo V(A), cuando se están transmitiendo tramas (comandos o respuestas) de supervisión y tramas comando I. Esta variable identifica en cada momento el número de la trama, de las transmitidas por la propia entidad, de la que ya se ha acusado recibo por la entidad par remota de modo que el N(T) de la última trama I de la que se ha acusado recibo será igual a V(A)-1. Esta variable puede tomar cualquiera de los valores comprendidos entre cero y el valor del módulo menos uno (127). La variable V(A) se actualiza continuamente con los valores N(R) válidos recibidos en las tramas procedentes de la entidad par remota. Un valor N(R) recibido será válido si cumple la condición $V(A)< N(R) < V(T)$.

c) Variable de estado en recepción V(R):

Cada punto extremo de conexión de enlace de datos punto a punto mantendrá asociada una variable de estado en recepción V(R) cuando se están transmitiendo tramas comando I y tramas (comandos o respuestas) de supervisión. Esta variable indica el número de secuencia de la próxima trama I a recibir y puede tomar un valor comprendido entre cero y el módulo menos uno (127). El valor de V(R) se incrementará en una unidad con la recepción de una trama I libre de errores cuyo número de secuencia en transmisión N(T) coincide con el valor actual de la variable de estado en recepción V(R).

d) Número de secuencia en transmisión N(T):

Solamente las tramas I contienen este parámetro en el campo de control que indica el número de secuencia de las tramas transmitidas. Cada vez que una trama I se designa para ser transmitida, su valor N(T) es puesto al valor indicado por la variable de estado en transmisión V(T).

e) Número de secuencia en recepción N(R):

Todas las tramas I y de supervisión contienen este parámetro en el campo de control que indica el número de secuencia de la próxima trama I que se espera recibir. Cuando una trama de las citadas se designa para la transmisión, el valor N(R) es puesto al valor de la variable de estado en recepción V(R). El parámetro N(R) indica que la entidad de la capa 2 que lo transmite ha recibido correctamente todas las tramas I numeradas secuencialmente hasta el valor N(R)-1 inclusive.

3.8.4 COMANDOS Y RESPUESTAS

Los comandos y respuestas que se definen a continuación, cuyo campo de control y codificación se reflejan en la tabla de la figura 3.21 , constituyen el conjunto de tramas utilizadas en el protocolo entre pares de la capa de enlace de datos LAP D que se mantendrá entre las entidades de capa 2. Cada entidad de la capa de enlace de datos debe soportar el conjunto total de dichas tramas.Las tramas recibidas y que estén relacionadas con una aplicación no soportada serán despreciadas y no se llevará a cabo acción alguna (caso de las tramas XID y de las tramas UI en el acceso primario de usuario con IET de valor cero); Cualquier trama recibida que no corresponda a una de las identificadas en el cuadro de la figura 3.21 será considerada como no definida.

a) Comando de información I

La función del comando de información (I) es la transferencia, a través de las conexiones de enlace de datos, de tramas numeradas de un modo secuencial conteniendo un campo de información, campo destinado al transporte de información

proporcionada y con destino a las entidades de capa 3. El campo de información de estas tramas será de una longitud máxima de N201 octetos (el valor del parámetro N201 es 260).

Estas tramas son utilizadas en el modo de funcionamiento multitrama (módulo 128) sobre conexiones de enlace de datos punto a punto.

Figura 3.21 COMANDOS Y RESPUESTAS

Aplicación	Formato	Comando	Respuesta	Codificación del campo de control
				8---7---6---5---4---3---2---1
	transferencia de información	I		N(T) ----------0
				N(R) ----------P
transferencia de información con/sin acuse de recibo		RR	RR	0---0---0---0---0---0---0---0---1
				N(R)----------P/F
	supervisión(S)	RNR	RNR	0---0---0---0---0---0---1---0---1
				N(R)----------P/F
		REJ	REJ	0---0---0---0---0---1---0---0---1
				N(R)----------P/F
		SABME		0---1---1---P---1---1---1---1---1
	no numeradas(U)		DM	0---0---0---F---1---1---1---1---1
		UI		0---0---0---P---0---0---0---1---1
		DISC		0---1---0---P---0---0---0---1---1
			UA	0---1---1---F---0---0---0---1---1
			FRMR	1---0---0---F---0---0---1---1---1
gestión de conexión		XID	XID	1---0---1---P/F---1---1---1---1

b) Comando de establecimiento del modo equilibrado asíncrono (extendido), SABME:

El comando no numerado de establecimiento del modo equilibrado asíncrono SABME es utilizado para establecer o restablecer el modo de funcionamiento multitrama con acuse de recibo y módulo de funcionamiento 128 entre las entidades de capa de enlace de datos del lado de red y del lado de usuario direccionadas, entre las que se establece la conexión de enlace de datos. Este comando carece de campo

de información.

Una entidad de capa 2 comunica la aceptación de este comando mediante la transmisión de una respuesta UA. Tras la aceptación de este comando, se pondrán a cero las variables de estado en transmisión V(T), en recepción V(R) y del acuse de recibo V(A) de la entidad de capa 2. La transmisión de este comando SABME indicará la anulación de toda posible condición de excepción indicada con anterioridad.

Las posibles tramas I que se hallan transmitido anteriormente y que se encuentren pendientes del acuse de recibo, cuando se procesa el comando SABME (transmitido o recibido), serán despreciadas y quedarán sin acuse de recibo, siendo responsabilidad de las entidades de capa superior (por ejemplo de capa 3) la recuperación de la posible pérdida de información que puede tener lugar.

c) Comando de desconexión, DISC:

Este comando no numerado se transmitirá para finalizar la operación en el modo multitrama. El comando DISC tampoco contiene campo de información en su estructura de trama.

La entidad de la capa de enlace de datos receptora de este comando confirma su aceptación mediante la transmisión de una respuesta UA. La entidad de la capa 2 que ha enviado el comando DISC dará por finalizado el modo de funcionamiento multitrama cuando recibe una trama respuesta UA o DM generado por la entidad par como acuse de recibo del comando DISC.

Las tramas I previamente transmitidas y pendientes del acuse de recibo serán despreciadas y quedarán sin acuse de recibo cuando se procesa el comando DISC (transmitido o recibido) y es responsabilidad de las entidades de capa superior (por ejemplo de capa 3) la recuperación de la posible pérdida de información que puede tener lugar.

d) Comando de información no numerada, UI:

Cuando la entidad de la capa 3 o la entidad de gestión de capa solicita una transferencia de información sin acuse de recibo se usará el comando no numerado UI, con lo cual se enviará la información, contenida en el campo de información de la trama, al otro extremo sin afectar a las variables de la capa de enlace de datos.

Las tramas UI no contienen numero de secuencia, por tanto, y de acuerdo con el procedimiento de transferencia de información sin acuse de recibo, estas tramas pueden perderse sin envío de notificación alguna.

Teniendo en cuenta que los procedimientos basados en el intercambio de tramas UI en el caso de conexiones de enlace de datos punto a punto del acceso primario de usuario no están soportadas, cualquier entidad de la capa 2 asociada a un acceso primario (IET de valor cero) que reciba una trama UI deberá despreciarla sin llevar a cabo acción alguna.

e) Comando/respuesta de receptor dispuesto RR:

Las tramas de supervisión (comando o respuesta) RR son utilizadas por una entidad de la capa de enlace de datos para:

a) Indicar que la entidad de la capa 2 está dispuesta para la recepción de una nueva trama I.

b) Acuse de recibo de las tramas I previamente recibidas y numeradas secuencialmente hasta el valor N(R)-1.

c) Anular la condición de ocupado que había sido indicada mediante la transmisión previa de una trama RNR por la misma entidad de la capa de enlace de datos.

Además de utilizarse para indicar el estado de la entidad de la capa 2 que lo transmite, el comando RR con el bit P puesto a "1" puede ser utilizado por la entidad de la capa 2 para preguntar por el estado de la entidad par de la capa de enlace de datos.

f) Comando/respuesta de rechazo, REJ:

Esta trama de supervisión es utilizada por la entidad de la capa 2 para solicitar la retransmisión de tramas I empezando por la trama indicada por el valor N(R) contenido en el campo de control de la trama REJ. El valor N(R) de la trama REJ acusa recibo de todas las tramas I previamente recibidas incluyendo la numerada con el valor N(R)-1. Las nuevas tramas pendientes de la transmisión inicial pueden ser transmitidas después de la retransmisión de las tramas I solicitadas.

En un momento y para un sentido de transmisión dados, solamente se puede establecer una condición de excepción de rechazo (REJ); estado que se considera anulado (reiniciación) con la recepción de una trama I con un N(T) igual al N(R) indicado previamente en la trama REJ.

La transmisión de una trama REJ puede también indicar la anulación de cualquier posible estado de ocupado preexistente en la entidad que la transmite, y que había sido indicado previamente mediante la transmisión de una trama RNR.

Además de indicar el estado de la entidad de la capa 2 que lo transmite, un comando REJ con el bit P puesto a "1" puede utilizarse para preguntar por el estado de la entidad par de la capa de enlace de datos.

g) Comando/respuesta de receptor no dispuesto, RNR:

La trama de supervisión RNR será utilizada por una entidad de la capa de enlace de datos para indicar su estado de ocupado, es decir, su incapacidad temporal para aceptar nuevas tramas I entrantes.

El valor N(R) contenido en las tramas RNR acusa recibo de las tramas I recibidas y numeradas secuencialmente hasta el valor N(R)-1, inclusive. Además de indicar el estado de la entidad de la capa 2 que lo transmite, el comando RNR con el bit P puesto a "1" puede utilizarse por la entidad de la capa 2 para preguntar por el estado de la entidad par de la capa de enlace de datos.

h) Respuesta de acuse de recibo no numerado, UA:

La respuesta no numerada UA es utilizada por las entidades de la capa de

enlace de datos como acuse de recibo de la recepción y aceptación de las tramas comando SABME o DISC, comandos que no serán procesados hasta que se transmita la respuesta UA. Las tramas UA no contendrán campo de información. La transmisión de un comando SABME exige la espera de la respuesta UA para considerar el modo de funcionamiento establecido. La transmisión de una respuesta UA indica la anulación de cualquier condición de ocupado previamente indicada por la transmisión de una trama RNR, por la misma entidad de la capa 2.

i) Respuesta de modo desconectado, DM:

La respuesta no numerada DM se utiliza para indicar a la entidad par de la capa 2 que la entidad que la transmite se encuentra en un estado tal que no puede mantenerse una comunicación multitrama. Una entidad de la capa de enlace de datos en este estado contestará con la trama DM al posible comando SABME o DISC recibido. Estas tramas no contendrán campo de información.

j) Respuesta de rechazo de trama, FRMR:

Una entidad de la capa de enlace de datos nunca generará una trama respuesta FRMR. Sin embargo, si la entidad de capa 2 del lado de red recibiese una trama FRMR la trataría de acuerdo con el procedimiento descrito en 3.17.1.6.

La respuesta no numerada FRMR puede ser recibida por una entidad de la capa 2 como indicación de la existencia de un error no recuperable mediante la retransmisión de las tramas implicadas. Esto supone la existencia de al menos una de las condiciones de error que se enumeran a continuación:

a) Recepción de una trama comando o respuesta, cuyo campo de control no está definido o no está soportado por la entidad receptora de la trama (bit W).

b) Recepción de una trama no numerada o de supervisión con una longitud incorrecta (bit X).

c) Recepción de una trama con un campo de información que supera la máxima longitud permitida (bit Y).

d) Recepción de la trama con un número de secuencia N(R) no válido (bit Z).

Un campo de control se considera no definido cuando su codificación es diferente de las previstas en el repertorio de tramas de la figura 3.21 . Un valor N(R) es válido si se encuentra en el margen V(A) <= N(R) <= V(T).

Esta trama posee un campo de información de una longitud de 5 octetos, colocados a continuación del campo de control de la trama, en el que se especifica la razón por la cual ha sido generada esta respuesta FRMR. Este campo de información se representa en la figura 3.22 .

En los dos primeros octetos de este campo de información se reproduce el campo de control de la trama que causó la respuesta FRMR. Cuando la trama que causa el rechazo es una trama no numerada (con campo de control de un sólo octeto), el campo de control de la trama se reproduce en el octeto 5 y el octeto 6 se codificará como todo "0".

Los valores V(T) y V(R), octetos 7 y 8 respectivamente, reproducen el estado actual de las variables de estado de la entidad que está transmitiendo la trama FRMR.

El bit C/R indicará si la trama rechazada fue un comando o una respuesta tomando los valores "0" o "1" respectivamente.

Los bits W, X, Y y Z, bits 1 a 4 del octeto 9, sirven para especificar la puesta a "1", la condición de error detectada mediante su dichos bits están asociados respectivamente con las cuatro condiciones indicadas anteriormente.

De acuerdo con las definiciones y condiciones descritas el bit W deberá ser puesto a "1" siempre que el bit X se ponga a "1"

En la estructura del campo de información de las tramas FRMR, que es la reproducida en la figura 3.22 se muestra como los bits no utilizados (bits 5 a 8 del octeto 9) se codifican permanentemente a "0"

k) Comando/respuesta de intercambio de identificación, XID:

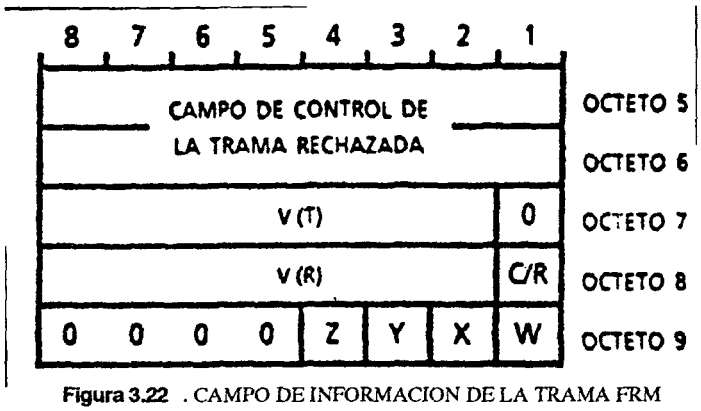


Figura 3.22 . CAMPO DE INFORMACION DE LA TRAMA FRM

Las tramas XID pueden contener un campo de información en el cual se transportará la información de identificación. El intercambio de tramas XID tiene lugar entre las entidades de gestión de la conexión de modo que cuando una entidad recibe una trama XID debe responder con otra trama XID lo antes posible. Estas tramas no contienen números de secuencia en su campo de control.

El campo de información de las tramas XID es opcional. Sin embargo, si se recibe una trama XID válida con un campo de información que el receptor puede interpretar la trama XID de respuesta deberá contener también campo de información. Si se recibe una trama XID con un campo de información que no se puede interpretar o de longitud nula, la trama XID de respuesta tendrá un campo de información de longitud nula. La longitud máxima del campo de información de una trama XID será N201 octetos.

La transmisión o recepción de una trama XID no afectará al modo de operación de la capa de enlace de datos ni a los usuarios de estado asociadas a las entidades de la capa 2.

Teniendo en cuenta que el procedimiento de negociación automática de parámetros de la capa de enlace de datos, basado en el intercambio de tramas XID, no se soporta, dichas tramas no son aplicables por lo que cualquier entidad de la capa de enlace de datos que reciba una trama XID deberá despreciarla sin llevar a cabo acción alguna.

3.9 INTRODUCCION A LOS PROCEDIMIENTOS DEL PROTOCOLO LAP-D

En 3.8 se han definido los elementos que constituyen la herramienta básica para el establecimiento del conjunto de procedimientos que constituyen el protocolo entre pares de la capa de enlace de datos (LAP D) basado en el intercambio de tramas de la capa de enlace de datos sobre el canal D. Los procedimientos que constituyen este protocolo son los que se enumeran a continuación:

Procedimiento para la utilización del bit P/F.

Procedimiento para la transmisión sin acuse de recibo.

Procedimientos de gestión de los identificadores de equipo terminal (IET).

Procedimiento de establecimiento del modo multitrama.

Procedimiento de liberación del modo multitrama.

Procedimientos para la transferencia de información en el modo multitrama.

Procedimiento para el restablecimiento del modo multitrama.

Procedimiento para la supervisión de la conexión de enlace de datos.

Estos procedimientos son tratados individualmente en las secciones posteriores de este capítulo. La descripción de estos procedimientos se complementa con unos diagramas de transición entre estados en los que se representa el intercambio de tramas entre las entidades de la capa de enlace de datos del lado de red y del lado de usuario.

Es de señalar el hecho de que el protocolo en el modo en que se describe se considera desde el punto de vista de la entidad de la capa 2 del lado de red aunque esta consideración en la mayoría de las ocasiones pierde sentido debido a la simetría de los procedimientos, pero no ocurre así cuando se trata de la interpretación de los diagramas de transición entre estados, los cuales corresponden a los estados posibles en que se puede encontrar la entidad de la capa 2 del lado de red, aunque podrían establecerse unos estados semejantes en la entidad del lado de usuario.

Dentro del conjunto de estados posibles existen tres que pueden considerarse fundamentales (IET no asignado, IET asignado y multitrama establecida) cuyas definiciones coinciden con las establecidas en la sección 5 para los estados del propio enlace de datos punto a punto.

3.10 PROCEDIMIENTO PARA LA UTILIZACION DEL BIT P/F

Como se ha visto, la utilización del bit P/F y su interpretación será diferente en función del tipo de trama que se está transmitiendo o recibiendo. A continuación se describen las diferentes particularidades del procedimiento.

3.10.1 TRANSFERENCIA DE INFORMACION SIN ACUSE DE RECIBO

En este modo de operación, cuyo procedimiento de transferencia de información se basa en el intercambio de tramas comando UI, el bit P/F, puesto a "0" permanentemente, no es utilizado.

3.10.2 TRANSFERENCIA DE INFORMACION CON ACUSE DE RECIBO. MODO MULTITRAMA

En este modo de operación este bit se utiliza como P en las tramas comando y como F en las tramas respuesta.

Una entidad de la capa de enlace de datos transmitirá un comando con el bit P puesto a "1" cuando desee obtener una respuesta inmediata de la entidad par extrema, respuesta que será identificada por ser transmitida con el bit F puesto a "1"

En el cuadro de la figura 3.23 se muestra la asociación posible entre comandos y respuestas transmitidas con el bit P/F puesto a "1"

Figura 3.23 UTILIZACIÓN DEL BIT P/F

COMANDOS RECIBIDOS CON P=1	RESPUESTAS TRANSMITIDAS CON F=1
SABME, DISC	UA, DM
I, RR, RNR,REJ	RR,RNR,REJ,DM

3.11 PROCEDIMIENTO PARA LA TRANSFERENCIA DE INFORMACION SIN ACUSE DE RECIBO

Este procedimiento solamente será aplicable en el caso de instalaciones de usuario tipo bus pasivo utilizando un acceso básico de usuario (2B+D), por tanto no será utilizado en el caso de accesos de usuario de jerarquía primaria (30B+D).

Como ya ha sido comentado en secciones anteriores, en este modo de transferencia de información no se definen procedimientos para la recuperación de errores por la capa de enlace de datos.

3.11.1 TRANSMISION DE INFORMACION SIN ACUSE DE RECIBO

El término "transmisión" de una trama UI se refiere al envío de la trama UI por la entidad de la capa de enlace de datos hacia la de la capa física.

La información sin acuse de recibo puede ser ofrecida a la entidad de la capa de enlace de datos para su transmisión tanto por la entidad de la capa 3 como por la entidad de gestión de capa utilizando las primitivas ED-UNIDAD DE DATOS-PETICION o GED-UNIDAD DE DATOS-PETICION, respectivamente.

La información procedente de la capa 3 o de la entidad de gestión de capa será transmitida mediante tramas comando UI. Para la operación sobre un enlace de difusión el valor de IET contenido en el campo de dirección del comando UI será 127, el valor de grupo, (en binario "111 1111"). Para la operación sobre un enlace punto a punto se utilizará el valor de IET apropiado, correspondiente al enlace de datos.

La codificación del campo de control de las tramas UI ya se ha expuesto en 3.8, debiendo codificarse el bit P con valor "0" permanentemente por no ser necesaria su utilización.

La entidad de gestión del sistema deberá asegurar que la capa física del sistema no sea desactivada antes de que se haya completado la transmisión de todas las tramas UI.

En el caso de que se produzca una desactivación de la capa 1 no recuperable,

la entidad de la capa 2 será informada mediante la indicación apropiada, FI-DESACTIVACION-INDICACION. Tras la recepción de dicha indicación, todas las posibles colas de transmisión de tramas UI serán desechadas. En el lado de red la entidad de gestión del sistema asegurará que la primitiva FI-DESACTIVACION-INDICACION sea enviada cuando ocurra la situación de desactivación persistente de la capa física.

3.11.2 RECEPCION DE INFORMACION SIN ACUSE DE RECIBO

La información contenida en el campo de información de un comando UI recibido por una entidad de la capa 2 con un IPAS y un IET soportados por la entidad receptora será enviado hacia la entidad de la capa 3 o de gestión mediante las primitivas de la capa de datos ED-UNIDAD DE DATOS-INDICACION o GED-UNIDAD DE DATOS-INDICACION, respectivamente. En caso contrario el comando UI recibido será desechado.

3.12 PROCEDIMIENTOS DE GESTION DE LOS IDENTIFICADORES DE EQUIPO TERMINAL

Los procedimientos descritos en esta sección, como procedimientos entre pares mantenidos entre las entidades de gestión de capa del lado de usuario y del lado de red solamente serán aplicables en el caso de instalaciones de abonado tipo bus pasivo utilizando en acceso básico de usuario (2B+D), que es a la que hacemos referencia en este trabajo, en cuyo caso los procedimientos deben ser soportados tanto por las entidades de gestión de capa del lado de usuario como del lado de red, salvo en el caso del procedimiento de verificación de valores IET descrito en el apartado 13.5 cuyo soporte por las entidades del lado de usuario es opcional dependiendo de las características de fabricación del equipo.

Los procedimientos entre pares de gestión de los identificadores de equipo terminal son los que se enumeran a continuación:

-Procedimiento de asignación de IET

-Procedimiento de comprobación de IET

-Procedimiento de verificación de IET

-Procedimiento de supresión de IET

Para el caso de equipos terminales conectados a un acceso básico de usuario pero utilizando valores IET de asignación no automática (valores IET en la gama 1 a 63) el procedimiento entre pares de asignación de IET no será aplicable quedándose reducido al procedimiento entre capas que tiene lugar entre la entidad de gestión de capa, que es la que contiene el valor IET de asignación no automática, y la entidad de la capa de enlace de datos, que es la que necesita dicho valor para las comunicaciones entre pares de la capa 2. Dicho procedimiento entre capas se basa en el intercambio de las primitivas GED-ASIGNACION-PETICION/INDICACION.

Para el caso de acceso primario de usuario dichos procedimientos entre pares no serán aplicables ya que siempre se utilizará el IET de valor "cero" de asignación no automática en cuyo caso también se mantendrá el protocolo entre capas, tal como se indicó para el caso de equipos terminales usando valores IET de asignación no automática, basado en el intercambio de las primitivas GED-ASIGNACION-PETICION-INDICACION.

3.12.1 GENERAL

Un equipo de usuario en el estado de "IET no asignado" utilizará los "procedimientos de asignación de IET" para entrar en el estado de "IET asignado". Estos procedimientos residen conceptualmente en las entidades de gestión de capa de la capa de enlace de datos tanto del lado de usuario como de red recibiendo esta última la denominación "Punto Fuente de Asignación" (PFA).

El objetivo general de este procedimiento se resume en los cuatro puntos siguientes:

a) Permitir que los equipos de usuario de gestión automática de IET soliciten de la red la asignación de un valor IET, que las entidades de la capa de enlace de datos

dentro del equipo de usuario solicitante utilizarán en las comunicaciones posteriores.

b) Permitir que la red suprima un valor IET previamente asignado de uno o de todos los equipos de usuario.

c) Permitir que la red compruebe si un determinado valor IET está o no en uso o si se ha producido una asignación múltiple de valores IET.

d) Permitir que un equipo de usuario solicite que la red inicie el procedimiento de comprobación de valores IET.

La entidad de gestión de capa del lado de usuario indicará a la entidad de la capa de enlace de datos la supresión de todos los valores IET cuando reciba notificación de la desconexión del terminal del interfaz.

Además, la entidad de gestión de capa del lado de usuario podría indicar a la entidad de la capa de enlace de datos la supresión de un valor IET por razones internas (por ejemplo, pérdida de la capacidad de comunicación con la red). La entidad de gestión de capa llevará a cabo esta acción mediante el uso de la primitiva GED-SUPRESION-PETICION.

En general, un equipo de usuario debería utilizar un único valor IET, el cual sería utilizado en asociación con todos los IPAS que dicho equipo pueda soportar. Si fuese necesario un equipo de usuario podría solicitar múltiples valores IET mediante la utilización del procedimiento de asignación de IET varias veces, en cuyo caso sería responsabilidad del equipo de usuario el mantener la asociación necesaria entre los valores IET y los valores IPAS.

El procedimiento de asignación de IET será iniciado con la recepción de una solicitud de establecimiento de la conexión de enlace de datos (ED-ESTABLECIMIENTO-PETICION) o una solicitud de transferencia de información sin acuse de recibo (ED-UNIDAD DE DATOS-PETICION). La entidad de la capa de enlace de datos informará a la entidad de gestión de capa de la necesidad de asignación de un valor IET mediante la primitiva GED-ASIGNACION-INDICACION.

En el caso de iniciación de los procedimientos de la capa 2 después de una interrupción del suministro de energía, el equipo de usuario debería posponer el arranque del procedimiento de asignación de IET hasta el momento en que se detecte la necesidad de proporcionar un servicio de capa 2 que precise de un valor de IET.

Todos los mensajes relacionados con los procedimientos de gestión de IET son transmitidos o recibidos por la entidad de gestión de capa (hacia o desde la entidad de la capa de enlace de datos) mediante las primitivas GED-UNIDAD DE DATOS-PETICION y GED-UNIDAD DE DATOS-INDICACION respectivamente. La entidad de la capa de enlace de datos transmitirá los mensajes de la entidad de gestión de capa relacionados con la gestión de los valores IET mediante tramas comando UI. Estas tramas UI serán transmitidas utilizando el valor IPAS 63 (procedimientos de gestión de capa de la capa 2), y el valor IET 127 (valor de grupo)

En el diagrama de transición entre estados que se muestra en la figura 3.24 se ilustran los procedimientos de gestión de los identificadores de equipo terminal. En dicha figura además de los estados de enlace de datos de "IET no asignado" e "IET asignado" se ha introducido el estado intermedio de transición "Comprobación de IET" con fines ilustrativos del procedimiento.

A continuación se enumeran los diferentes mensajes de gestión transmitidos durante el desarrollo de estos procedimientos así como la notación abreviada empleada en los diagramas de transición entre estados.*

TIPO DE MENSAJE	NOTACIÓN
SOLICITUD DE IDENTIDAD	ID SOL
IDENTIDAD ASIGNADA	ID ASIG
IDENTIDAD DENEGADA	ID DEN
SOLICITUD DE COMPROBACION DE IDENTIDAD	ID COMP SOL

TIPO DE MENSAJE	NOTACIÓN
RESPUESTA DE COMPROBACION DE IDENTIDAD.	ID COMP RES
VERIFICACION DE IDENTIDAD	ID VER
SUPRESION DE IDENTIDAD	ID SUPR

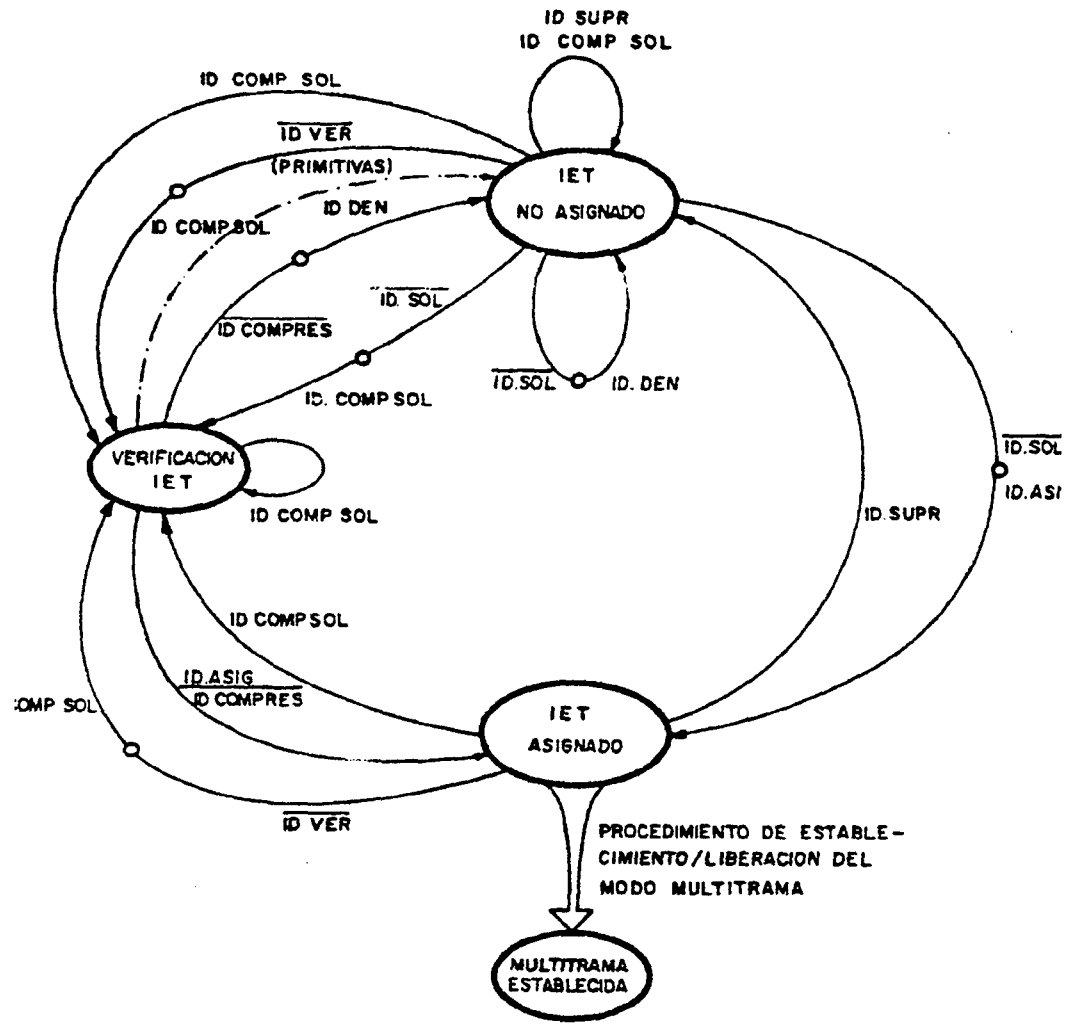


Figura 3.24 DIAGRAMA DE TRANSICIÓN ENTRE ESTADOS PARA LOS PROCEDIMIENTOS DE GESTIÓN DE LOS IET.

3.12.2 FORMATOS Y CÓDIGOS.

Todos los mensajes utilizados en los procedimientos de asignación, verificación y supresión de valores IET son transmitidos en el campo de información (cinco octetos) de comandos UI intercambiados entre las entidades de gestión de capa de capa de ambos lados del interfaz usuario/red. Estas tramas son transmitidas utilizando como elementos del campo de dirección de las tramas los valores IPAS = 63 ("11 1111" en binario) y IET = 127 ("111 1111" en binario) correspondientes a los procedimientos de gestión y grupo de usuarios respectivamente. En la figura 3.25 se muestra el formato de estos comandos.*

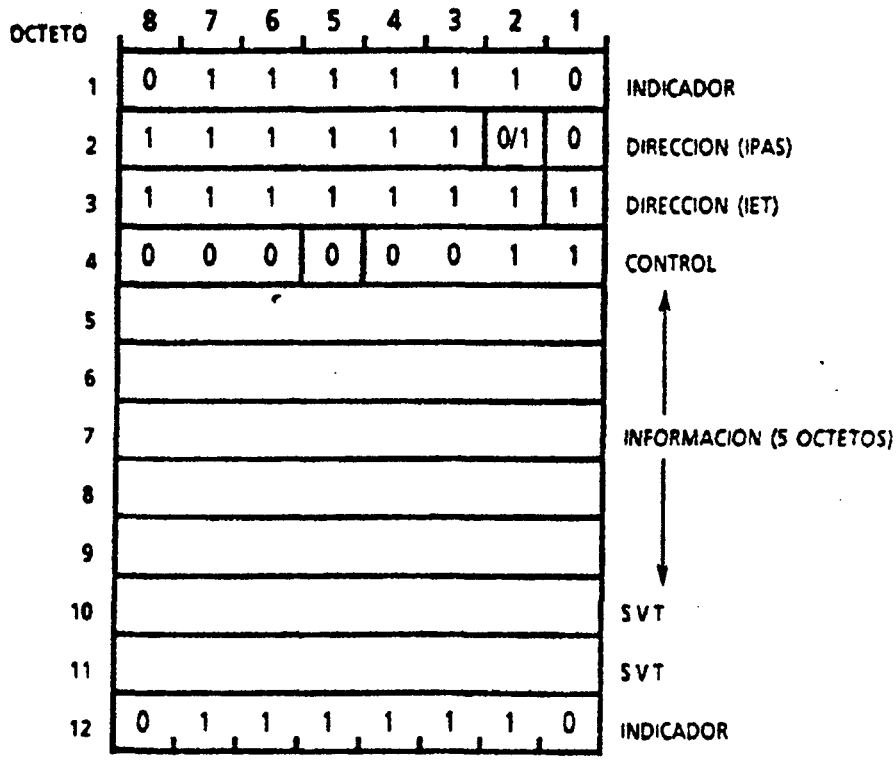


Figura 3.25 . ESTRUCTURA DE LOS COMANDOS UI.

Los cinco octetos del campo de información de estas tramas que componen los mensajes utilizados en los procedimientos de gestión de los IET se dividen en varios subcampos tal como se indica en la figura 3.27 .

Los campos que no son utilizados en un determinado mensaje se codifican

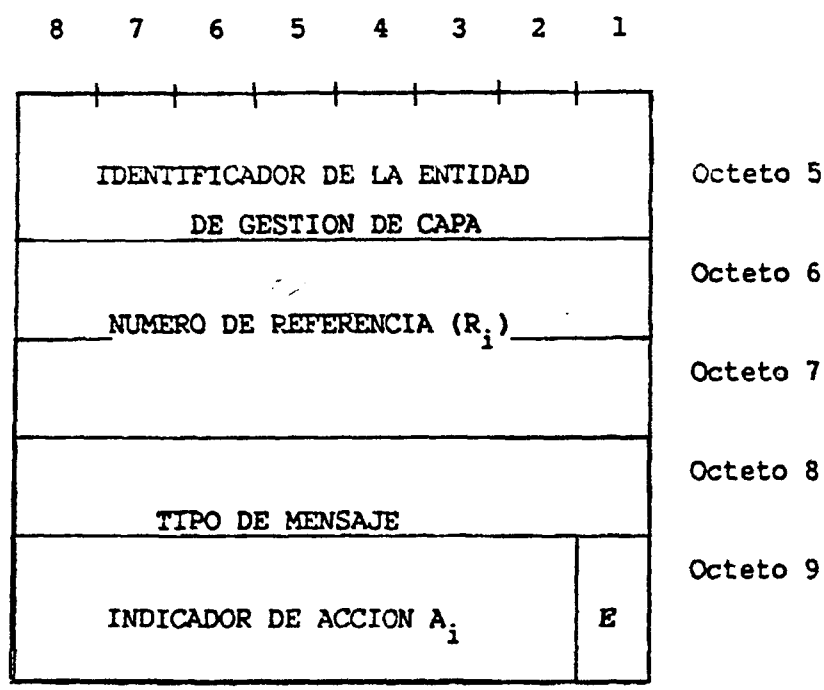


Figura 3.26 ESTRUCTURA DEL CAMPO DE INFORMACIÒN DE LOS COMANDOS UI

como todo "0" y no son procesados por las entidades receptoras.

El campo "identificador de la entidad de gestión" está pensado para su posible utilización futura cuando puedan existir diferentes procedimientos que impliquen la utilización de varias entidades de gestión. Para el caso de los procedimientos de gestión de IET se codifica como "0000 1111".

El campo de dos octetos del "número de referencia" (R_i) se utilizará para diferenciar entre un determinado número de equipos de usuario que posiblemente envíen simultáneamente un mensaje hacia la red. Este número será generado aleatoriamente por el equipo de usuario para cada uno de los mensajes de solicitud pudiendo tomar cualquier valor entre 0 y 65535. Este amplio margen de valores junto con un diseño adecuado del generador aleatorio minimizará la posibilidad de repetición del valor R_i en dos terminales que envíen un mensaje de solicitud simultáneamente.

El campo de un octeto "tipo de mensaje" identificará mediante codificaciones diferentes cada uno de los mensajes utilizados en estos procedimientos de gestión.

El campo "indicador de acción" A_i tiene una longitud efectiva de siete bits (los más significativos del octeto) que permiten contener los diferentes valores de IET posibles y que en determinados momentos se necesitan manejar de un modo u otro en función del tipo de mensaje de que se trate.

El campo A_i puede ser extendido mediante el uso del primer bit del octeto como "bit de extensión (E)" el cual indicará cuál es el último octeto del campo A_i mediante su codificación al valor "1" binario, el bit E puesto a "0" indicara la existencia de octetos adicionales del campo A_i.

La codificación de los diferentes campos en función del tipo de mensaje se muestra en el cuadro de la figura 3.27 .

Figura 3.27 CODIFICACIÓN DE LOS MENSAJES UTILIZADOS EN LOS PROCEDIMIENTOS DE GESTIÓN DE IET

Nombre de mensaje	Identificador de la entidad de gestión	Número de referencia R_i	Tipo de mensaje	Indicador de acción A_i
solicitud de identidad, usuario a red	00001111	0-65535	00000001	Ai=127, cualquier valor IET es aceptable
identidad asignada, red a usuario	00001111	0-65535	00000010	A_i=64-126, valor IET asignado
identidad denegada, red a usuario	00001111	0-65535	00000011	A_i=64-126, valor IET denegado A_i=127, ningún valor IET disponible
solicitud de comprobación de identidad, red a usuario	00001111	no utilizado, codificado"0"	00000100	A_i=127, comprobación de todos los valores IET A_i=0-126, valor IET a ser comprobado
respuesta de comprobación de identidad, usuario a red	00001111	0-65535	00000101	A_i=0-126, valor IET en uso
verificación de identidad, usuario a red	00001111	no utilizado, codificado"0"	00000111	Ai=0-126, valor IET a ser verificado
supresión de identidad, red a usuario	00001111	no utilizado, codificado"0"	00000110	A_i=127,supresión de todos los valores IET A_i=0-126, valor IET a ser suprimido

3.12.3 PROCEDIMIENTO DE ASIGNACION DE IET

En el caso de que el equipo usuario (terminal) no sea de la categoría de asignación automática de valores IET (valores IET preasignados, en el rango 1-63), la entidad de gestión de capa enviará hacia la entidad o entidades de la capa de enlace

de datos el valor IET a ser utilizado mediante la primitiva GED-ASIGNACION-PETICION.

En el caso de que el equipo de usuario (terminal) sea de la categoría de asignación automática de valores IET (valores IET en el rango 64 a 126), tras la iniciación del procedimiento de asignación automática de IET con la recepción de la primitiva GED-UNIDAD DE DATOS-INDICACION, la entidad de gestión de capa del lado de usuario transmitirá hacia su entidad par (del lado red) un mensaje de gestión conteniendo los elementos siguientes:

I) Tipo de mensaje: SOLICITUD IDENTIDAD

II) R_i: NUMERO DE REFERENCIA

III) A_i: INDICADOR DE ACCION

El número de referencia permitirá diferenciar entre varios terminales que envíen simultáneamente una solicitud de un valor IET.

El valor Ri tendrá una longitud de dos octetos y será generado por los terminales aleatoriamente en el margen de cero a 65535 para cada mensaje de solicitud.

El indicador de acción A_i se utilizará para indicar al punto fuente de asignación de cualquier valor de IET disponible. El campo A_i tendrá una longitud de un octeto y será codificado al valor 127 (IET de grupo).

Con la transmisión del mensaje SOLICITUD DE IDENTIDAD se arrancará el temporizador T202 (el valor del temporizador T202 es de 2 segundos).

El punto fuente de asignación al recibir el mensaje SOLICITUD DE IDENTIDAD llevará a cabo una de las tres acciones siguientes:

a) Seleccionar un valor IET,

b) Denegar las solicitudes de identidad conteniendo un valor A_i en el margen de 64 a 125 e ignorar las solicitudes de identidad conteniendo un valor A_i en el margen de 0 a 63.

c) Ignorar el mensaje SOLICITUD DE IDENTIDAD recibido si previamente se ha recibido otro mensaje de solicitud conteniendo el mismo valor R_i y aún no se ha enviado una respuesta. En este caso no se efectuará una asignación de valor IET para ninguna de las solicitudes.

La asignación de los valores IET podrá ser llevada a cabo sobre la base de la información almacenada en el PFA, que podría consistir en:

-Una tabla completa de la gama de valores IET que pueden ser asignados automáticamente y su estado de asignación.

-Una lista actualizada de todos los valores IET que aún están disponibles para su asignación utilizando el procedimiento automático de asignación.

El PFA, después de la selección del valor IET, informará a las entidades de la capa de enlace de datos mediante la primitiva GED-ASIGNACION-PETICION y transmitirá un mensaie hacia la entidad par remota del lado de usuario conteniendo los elementos siguientes:

I) Tipo de mensaje: IDENTIDAD ASIGNADA

II) R_i: NUMERO DE REFERENCIA

III) A_i: VALOR IET ASIGNADO (de 64 a 126)

Si la información disponible por el PFA indica que no existen valores IET disponibles, el PFA deberá iniciar el procedimiento de comprobación de valores IET

Las entidades de gestión de capa de la capa 2 del lado de usuario al recibir el mensaje IDENTIDAD ASIGNADA, analizarán su contenido y compararán el contenido de su campo A_i con sus propios valores de IET (en el caso de que existan) para determinar si ya ha sido asignado un valor IET en el caso de que existiese una solicitud pendiente. Adicionalmente las entidades de gestión podrán comparar los valores IET contenidos en el campo A_i de todos los mensajes IDENTIDAD ASIGNADA con su propio valor IET. Si el valor IET recibido coincide con el valor ya asignado a un equipo terminal, la entidad de gestión de capa de dicho terminal

deberá llevar a cabo una de las acciones siguientes:

a) Iniciar el procedimiento de supresión de IET.

b) Iniciar el procedimiento de verificación de IET.

Si el valor IET recibido no coincide con un valor ya asignado las entidades de gestión de capa del lado de usuario deberán llevar a cabo una de las acciones siguientes:

a) Si existe una petición de identidad IET pendiente, comparar el valor R_i recibido con el generado en el mensaje SOLICITUD DE IDENTIDAD y si son iguales considerar el valor IET asignado a dicho equipo terminal tras lo cual el valor R_i deberá ser anulado. En estas condiciones la entidad de gestión de capa deberá informar a la entidad de la capa de enlace de datos la asignación del valor IET mediante la primitiva GED-ASIGNACION-PETICION y el temporizador T202 deberá ser detenido.

b) Si existe una petición de identidad IET pendiente y el valor Ri recibido no coincide con el previamente generado la entidad de gestión de capa no llevará a cabo acción alguna.

c) Si no existe una petición de identidad IET pendiente y el valor IET recibido no coincide con uno ya asignado la entidad de gestión de capa de la capa 2 no llevará a cabo acción alguna.

Cuando la entidad de la capa de enlace de datos reciba la primitiva GED-ASIGNACION-PETICION, procedente de la entidad de gestión de capa, llevara a cabo las dos acciones siguientes:

-Entrar en el estado "IET asignado".

-Iniciar el procedimiento de establecimiento de enlace de datos si se encuentra pendiente una primitiva ED-ESTABLECIMIENTO-PETICION o iniciar la transmisión de una trama comando UI si se encuentra pendiente una primitiva ED-UNIDAD DE DATOS-PETICION.

Para denegar un valor IET el PFA rechazará el mensaje SOLICITUD DE

IDENTIDAD mediante la transmisión de un mensaje conteniendo los elementos siguientes:

I) Tipo de mensaje: IDENTIDAD DENEGADA

II) R_i: NUMERO DE REFERENCIA

III) A_i: IET DENEGADO (El valor 127 indica que no existe ningún valor IET disponible)

Si tras la transmisión del mensaje SOLICITUD DE IDENTIDAD el temporizador T202 expira y no se ha recibido respuesta o se ha recibido un mensaje IDENTIDAD DENEGADA, el temporizador será rearrancado y el mensaje de SOLICITUD DE IDENTIDAD será retransmitido utilizando un nuevo valor R_i.

Después de N202 intentos (el valor del parámetro N202 es 3 veces) de obtener un valor IET, la entidad de gestión de capa deberá informar a la entidad de la capa de enlace de datos de este evento mediante la primitiva GED-ERROR-RESPUESTA. La entidad de la capa de enlace de datos deberá responder a dicha primitiva con el envío de la primitiva ED-LIBERACION-INDICACION si se encuentra pendiente una primitiva ED-ESTABLECIMIENTO-PETICION y deberán despreciarse todas las primitivas ED-UNIDAD DE DATOS-PETICION pendientes de atención.

La figura 3.28 ilustra el procedimiento de asignación de valores IET.

3.12.4 PROCEDIMIENTO DE COMPROBACION DE IET.

El procedimiento de comprobación de IET será utilizado para la comprobación del estado de los valores IET y la recuperación de condiciones de error. Este procedimiento permite a la entidad de sesión de capa del lado de red determinar si un valor de IET está en uso y verificar una posible asignación múltiple de valores IET.

El procedimiento de comprobación para la verificación de una asignación múltiple de valores IET puede también iniciarse como consecuencia de la recepción de un mensaje VERIFICACION DE IDENTIDAD procedente del equipo de usuario solicitando esta verificación.

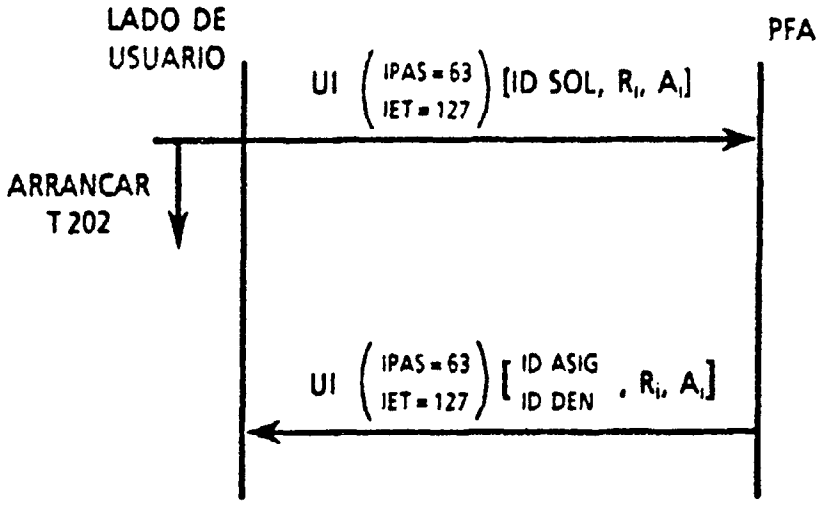


Figura 3.28 . PROCEDIMIENTO DE ASGNACIÓN DE IET.

El procedimiento de comprobación de IET será iniciado por el PFA mediante la transmisión de un mensaje conteniendo los elementos siguientes:

I) Tipo de mensaje: SOLICITUD DE COMPROBACION DE IDENTIDAD

II) Ai: VALOR IET A COMPROBAR (el valor 127 indicará que todos tienen que ser comprobados)

Con la transmisión de este mensaje la red arrancará el temporizador T201 (el valor del temporizador T201 es de 1 segundo).

Si cualquier equipo terminal tiene asignado el valor IET indicado en el mensaje SOLICITUD DE COMPROBACION DE IDENTIDAD, éste deberá responder con un mensaje conteniendo los elementos siguientes:

I) Tipo de mensaje: RESPUESTA DE COMPROBACION DE IDENTIDAD

II) A_i: VALOR IET

III) R_i: NUMERO DE REFERENCIA

El valor R_i generado aleatoriamente y contenido en el mensaje RESPUESTA DE COMPROBACION DE IDENTIDAD tiene por misión asegurar que cuando más de un terminal transmite la respuesta al mismo tiempo produce la colisión de capa 1 en el

acceso al canal D. La resolución de esta colisión resulta en la transmisión de múltiples mensajes RESPUESTA DE COMPROBACION DE IDENTIDAD.

En el caso de que el procedimiento de comprobación de valores IET se esté utilizando para la verificación de una posible asignación múltiple de valores IET deberá tenerse en cuenta los eventos siguientes:

-Si se reciben más de un mensaje RESPUESTA DE COMPROBACION DE IDENTIDAD durante el período de tiempo T201 deberá asumirse la existencia de dicha asignación múltiple, en caso contrario la solicitud deberá repetirse y el temporizador rearrancado.

-Si se reciben más de un mensaje RESPUESTA DE COMPROBACION DE IDENTIDAD durante el segundo período de tiempo T201 deberá asumirse la existencia de dicha asignación múltiple.

-Si no se recibe ningún mensaje RESPUESTA DE COMPROBACION DE IDENTIDAD, después de la segunda expiración del temporizador T201, el valor IET se considerará libre y por tanto disponible para su asignación o reasignación.

-Si solamente se ha recibido un mensaje RESPUESTA DE COMPROBACION DE IDENTIDAD en el primero o en ambos períodos de tiempo T201 el valor IET se considera en uso pero sin la existencia de asignación múltiple.

Cuando el procedimiento de comprobación de IET es utilizado para determinar si un valor particular de IET está en uso o no, el procedimiento se completa tras la recepción del primer mensaje RESPUESTA DE COMPROBACION DE IDENTIDAD, la cual determina que dicho valor IET se encuentra en uso. Para el desarrollo de este procedimiento deberán tenerse en cuenta los dos casos siguientes:

-Si durante el período de tiempo T201 no se recibe ninguna respuesta, el mensaje SOLICITUD DE COMPROBACION DE IDENTIDAD debe ser retransmitido y el temporizador T201 rearrancado.

-Si tras la segunda expiración del temporizador T201 no se ha recibido

respuesta alguna el valor IET se considerará libre y por tanto disponible para su asignación o reasignación.

Si el valor A_i contenido en el mensaje SOLICITUD DE COMPROBACION DE IDENTIDAD es 127, es preferible que la entidad de gestión de capa receptora del lado de usuario responda con un único mensaje RESPUESTA DE COMPROBACION DE IDENTIDAD conteniendo todos los valores IET soportados por dicho equipo terminal. Si el mensaje SOLICITUD DE COMPROBACION DE IDENTIDAD se ha transmitido con el valor A_i de 127, y se recibe un mensaje RESPUESTA DE COMPROBACION DE IDENTIDAD haciendo uso de la facilidad de extensión (inclusión de varios valores IET en un único mensaje de respuesta), cada variable A_i contenida en el campo A_i deberá ser procesada como si se hubieran recibido en mensajes separados RESPUESTA DE COMPROBACION DE IDENTIDAD cada uno de ellos correspondiente a distintos mensajes SOLICITUD DE COMPROBACION DE IDENTIDAD.

La figura 3.29 ilustra el procedimiento de comprobación de valores IET.

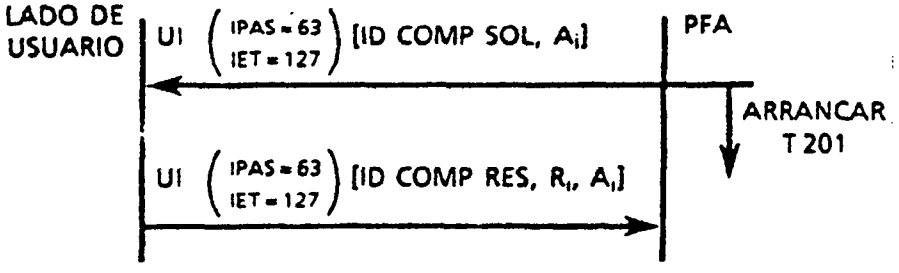


Figura 3.29 . PROCEDIMIENTO DE COMPROBACIÓN DE IET.

3.12.5 PROCEDIMIENTO DE VERIFICACION DE IET

El procedimiento aquí descrito debe ser soportado en el caso de instalaciones básicas de usuario (buses pasivos) por el lado de red de la conexión de enlace de datos y opcionalmente podría también ser soportado por los equipos terminales; esta opción dependerá de las características de fabricación de los terminales conectados

en el bus pasivo).

El procedimiento de verificación de IET permite a la entidad de gestión de capa del lado de usuario solicitar de la red el inicio del procedimiento de comprobación de IET para llevar a cabo la verificación de una asignación múltiple de IET.

Este procedimiento será iniciado por el equipo de usuario mediante la transmisión del mensaje VERIFICACION DE IDENTIDAD mensaje que contendrá los elementos siguientes:

I) Tipo de mensaje: VERIFICACION DE IDENTDAD

II) A_i: IET A SER VERIFICADO (0-126, el valor 127 no es permitido)

Con la transmisión de este mensaje el equipo de usuario arrancará el temporizador T202.

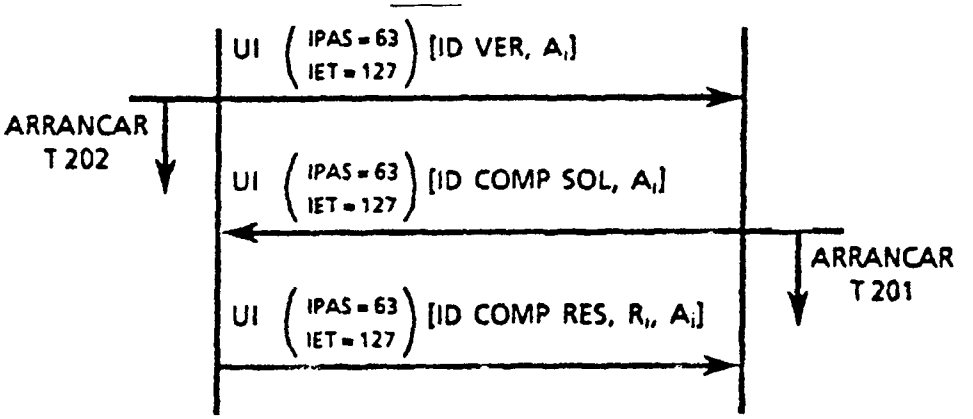
El PFA tras la recepción del mensaje VERIFICACION DE IDENTIDAD, deberá iniciar el procedimiento de comprobación de IET tal como ha sido descrito en 3.12.4, transmitiendo un mensaje SOLICITUD DE COMPROBACION DE IDENTIDAD.

El equipo de usuario al recibir el mensaje SOLICITUD DE COMPROBACION DE IDENTIDAD verificará si el valor IET

contenido en el campo A_i coincide con el valor IET que ha solicitado verificar o con 127, deteniendo el temporizador T202 en caso de que coincida. En cualquier caso el terminal deberá contestar al mensaje SOLICITUD DE COMPROBACION DE IDENTIDAD de acuerdo con lo descrito en 3.12.4.

Si el equipo de usuario no recibe un mensaje SOLICITUD DE COMPROBACION DE IDENTIDAD, indicando en el campo A_i el IET particular del terminal o el valor 127, antes de la expiración del temporizador T202, la entidad de gestión de capa del lado de usuario deberá retransmitir el mensaje VERIFICACION DE IDENTIDAD y rearrancar el temporizador T202. Si el terminal no recibe un mensaje SOLICITUD DE COMPROBACION DE IDENTIDAD adecuado antes de la segunda expiración del temporizador T202 el valor IET será suprimido.

La figura 2.30 ilustra el procedimiento de verificación de valores IET.



( ): contenido del campo de dirección del comando UI.
< >: contenido del campo de información del comando UI.

Figura 3.30 . PROCEDIMIENTO DE VERIFICACIÓN DE IET.

3.12.6 PROCEDIMIENTO DE SUPRESION DE IET

En el lado de red, los valores IET deben ser suprimidos cuando ocurra una de las condiciones siguientes:

-Tras la finalización del procedimiento de comprobación de valores IET cuando se determine que un valor IET ya no está siendo utilizado o cuando se detecte una asignación múltiple de valores IET.

-Tras la recepción de una primitiva GED-ERROR-INDICACION indicando una posible asignación múltiple de un valor IET, evento que será confirmado mediante la utilización del procedimiento de comprobación de IET.

En el lado de usuario, los valores IET de la gama de asignación automática (64-126) deben ser suprimidos cuando ocurra una de las condiciones que se enumeran a continuación. Para el caso de equipos terminales soportando valores IET correspondientes a la gama de asignación no automática (1-63) cuando ocurra una de las condiciones que se enumeran a continuación, deberá enviarse una indicación adecuada hacia el usuario:

-Tras la recepción de un mensaje SUPRESION DE IDENTIDAD procedente del

PFA.

-Tras la recepción de una primitiva GFI-INFORMACION-INDICACION indicando la condición "desconectado".

-Tras la recepción de una primitiva GED-ERROR-INDICACION indicando que la entidad de la capa de enlace de datos ha asumido una posible asignación múltiple de un valor IET. Esta acción se hará en lugar de solicitar un procedimiento de comprobación de IET mediante la transmisión de un mensaje SOLICITUD DE VERIFICACION DE IDENTIDAD.

-Tras la recepción de un mensaje IDENTIDAD ASIGNADA conteniendo en el campo A_i un valor IET que está siendo utilizado por el equipo de usuario.

Cuando la entidad de gestión de capa de la capa de enlace de datos determina la necesidad de supresión de un valor de IET, el PFA transmitirá una primitiva GED-SUPRESION-PETICION y un mensaje conteniendo los elementos siguientes:

I) Tipo de mensaje: SUPRESION DE IDENTIDAD

II) A_i: VALOR IET A SUPRIMIR (el valor 127 indica que todos los equipos de usuario deberían suprimir todos sus valores IET)

Para incrementar la seguridad del procedimiento y teniendo en cuenta la posible pérdida del mensaje SUPRESION DE IDENTIDAD este mensaje será enviado por duplicado, es decir, dos veces consecutivas.

Cuando la entidad de gestión de capa del lado de usuario determina la necesidad de suprimir un valor IET, indicará a la entidad de la capa de enlace de datos la necesidad de pasar al estado "IET no asignado" mediante la primitiva GED-SUPRESION-PETICION. Esta acción será llevada a cabo para todos los valores IET cuando el campo A_i del mensaje SUPRESION DE IDENTIDAD contenga el valor 127.

Tras completar el procedimiento de supresión IET las acciones a llevar a cabo serán, la iniciación del procedimiento de asignación automática de un nuevo valor IET o la notificación al equipo de usuario del inicio de una acción correctiva cuando el

equipo de usuario no soporte los procedimientos automáticos de asignación de valores IET.

Cuando, como consecuencia del desarrollo del procedimiento de supresión de IET, una entidad de la capa de enlace de datos recibe la primitiva GED-SUPRESION-PETICION deberá llevar a cabo una de las acciones siguientes:

a) Si no existe una primitiva ED-LIBERACION-PETICION pendiente y el equipo de usuario no se encuentra en el estado "IET asignado", se debe enviar una primitiva ED-LIBERACION-INDICACION

b) Si existe una primitiva ED-LIBERACION-PETICION pendiente, se debe enviar la primitiva ED-LIBERACION-CONFIRMACION.

Después de estas acciones la entidad de la capa de enlace de datos debe pasar al estado "IET no asignado" después de la anulación de las colas de tramas I y UI.

La figura 3.31 ilustra el procedimiento de supresión de valores IET.

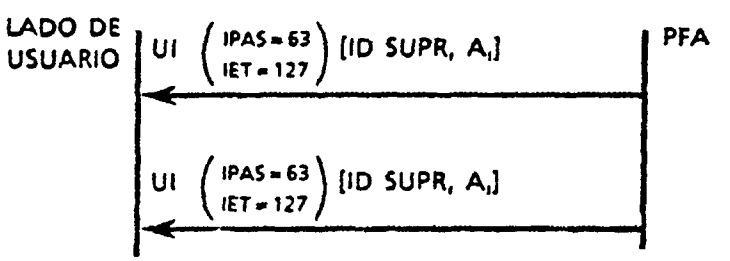


Figura 3.31 . PROCEDIMIENTO DE SUPRESIÓN DE IET.

3.13 PROCEDIMIENTOS DE ESTABLECIMIENTO Y LIBERACION DEL MODO MULTITRAMA.

De acuerdo con las definiciones contenidas en 3.4, el modo de funcionamiento multitrama permite la transferencia de información entre entidades de la capa 3 mediante la transmisión de tramas I numeradas secuencialmente en módulo 128.

En este apartado se describen tanto el procedimiento de establecimiento del modo de funcionamiento cuando la entidad de la capa de enlace de datos se

encuentra ya en el estado "IET asignado" como el de liberación, procedimientos basados en el intercambio de tramas no numeradas (formato U). Estos procedimientos serán usados entre las entidades de la capa de enlace de datos de la red y de un usuario.

En el diagrama de transición entre estados de la figura 3.32 se representan, con fines ilustrativos, estos procedimientos apareciendo los estados "IET asignado", "multitrama establecida", "establecimiento pendiente" y "liberación pendiente" propios de los puntos extremos de una conexión de enlace de datos.

En el caso de acceso primario de usuario la conexión de enlace de datos punto a punto se encontrará habitualmente en el estado "multitrama establecida" de modo que la utilización de los procedimientos de establecimiento y liberación se utilizarán excepcionalmente como consecuencia de eventos de la capa1,2 ó 3.

3.13.1 PROCEDIMIENTO DE ESTABLECIMIENTO DEL MODO MULTITRAMA

Una entidad de la capa 3 indicará la solicitud del establecimiento del modo de funcionamiento multitrama entre la red y un determinado equipo de usuario mediante la transmisión de la primitiva ED-ESTABLECIMIENTO-PETICION.

Durante el procedimiento de establecimiento serán ignoradas todas las tramas que no sean del formato no numerado (formato U).

Una entidad de la capa de enlace de datos indicará la solicitud para el establecimiento del modo multitrama mediante la transmisión del comando de establecimiento del modo equilibrado asíncrono SABME, con el bit P Puesto a "1".

Todas las posibles condiciones de excepción existentes serán anuladas y el contador de retransmisiones y el temporizador T200 (el valor del temporizador T200 es 1 segundo) serán iniciados con la transmisión del comando SABME.

Cuando el procedimiento de establecimiento se inicia como consecuencia de

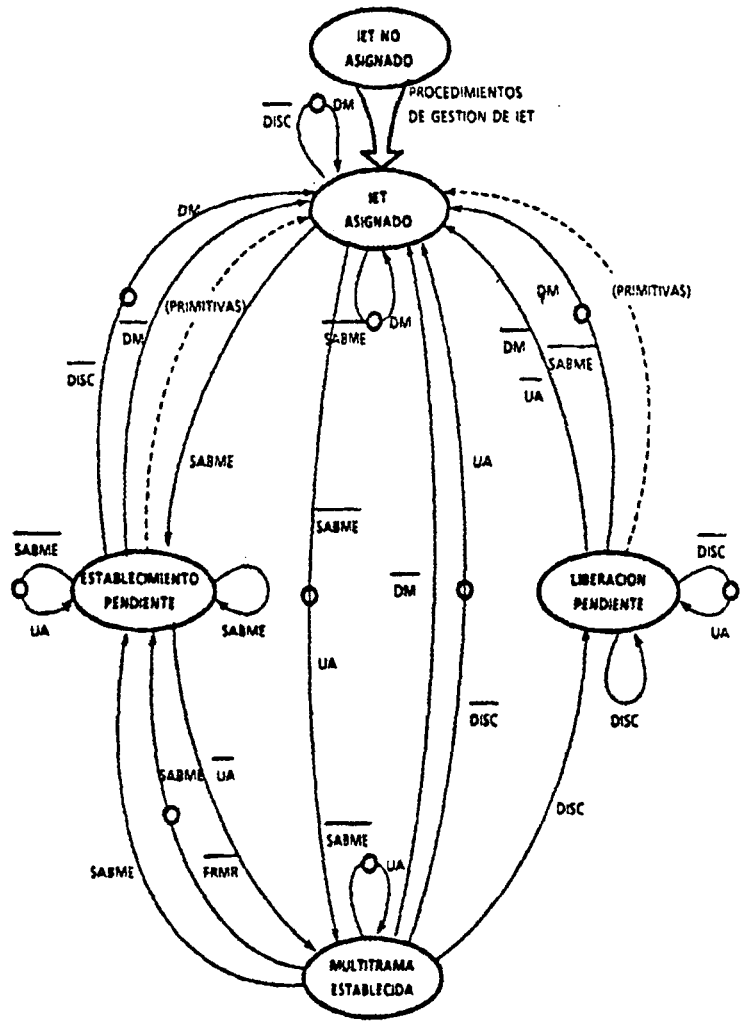


Figura 3.32 . DIAGRAMA DE TRANSICIÓN ENTRE ESTADOS PARA LOS PROCEDIMIENTOS DE ESTABLECIMIENTO Y LIBERACIÓN DEL MODO MULTITRAMA

una acción de la capa 3 todas las posibles primitivas ED-DATOS-PETICION y colas de tramas I pendientes serán despreciadas.

La entidad de la capa de enlace de datos en condiciones de establecer el modo multitrama, al recibir un comando SABME llevará a cabo las acciones siguientes:

-Responder con una trama de acuse de recibo no numerado UA con el bit F puesto a "1" (bit F del mismo valor binario que el bit P del comando SABME).

-Iniciar las variables de estado en transmisión, V(T); en recepción, V(R); y del acuse de recibo, V(A), al valor cero.

-Entrar en el estado "multitrama establecida" e informar de ello a la entidad de la

capa 3 mediante la primitiva ED-ESTABLECIMIENTO-INDICACION.

-Anular todas las posibles condiciones de excepción existentes.

-Anular la condición de ocupado en recepción de la entidad extrema si existe.

-Arrancar el temporizador T203 (el valor del temporizador T203 es 10 segundos).

Si la entidad de la capa 2 que recibe el comando SABME no está en condiciones de aceptarlo y entrar en el estado "multitrama establecida", transmitirá una respuesta DM con el bit F puesto a "1"

Al recibir la respuesta UA con el bit F puesto a "1" la entidad generadora del comando SABME llevará a cabo las acciones siguientes:

-Anular el temporizador T200.

-Arrancar el temporizador T203.

-Iniciar las variables de estado en transmisión, V(T); en recepción, Y(R); y del acuse de recibo, V(A), al valor cero.

-Entrar en el estado "multitrama establecida" e informar de ello a la entidad de la capa 3 mediante la primitiva ED-ESTABLECIMIENTO-CONFIRMACION.

Si la entidad generadora del comando SABME recibe una respuesta DM con el bit E puesto a "1", indicará este hecho a la entidad de la capa 3 mediante la primitiva ED-LIBERACION-INDICACION. En esta condición se anulará el temporizador T200 y entrará de nuevo en el estado "IET asignado". En este caso las respuestas DM, con el bit F puesto a "0", recibidas serán ignoradas.

Si durante el procedimiento de establecimiento se recibe una primitiva ED-LIBERACION-PETICION, ésta no será atendida hasta que se complete el procedimiento en curso.

Si el temporizador T200 expira antes de recibir una respuesta DM o UA con el bit F puesto a "1", la entidad de la capa 2 transmisora del comando SABME llevará a

cabo las acciones siguientes:

-Retransmisión del comando SABME con el bit P puesto a "1"

-Rearrancar el temporizador T200.

-Incrementar el contador de retransmisiones.

Después de la retransmisión del comando SABME N200 veces, (el valor del parámetro N200 es 3), la entidad de la capa de enlace de datos indicará esta incapacidad para el establecimiento del modo de funcionamiento a las entidades de la capa 3 y de gestión de la conexión mediante las primitivas ED-LIBERACION-INDICACION y GED-ERROR-INDICACION, respectivamente. En estas condiciones la entidad de la capa 2 entrará de nuevo en el estado "IET asignado".

Una vez en el estado "multitrama establecida", es decir después de haber recibido una respuesta UA a un comando SABME transmitido o después de responder con una trama UA a un comando SABME recibido, la entidad de la capa 2 puede comenzar la transmisión de tramas de supervisión y tramas I de acuerdo con los procedimientos de transferencia de información en modo multitrama que se describen en la sección 15 posterior.

Si se recibe un comando SABME en el estado de multitrama establecida, la entidad de la capa de enlace de datos llevará a cabo el procedimiento de restablecimiento del modo de funcionamiento tal como se describe en 3.15.

La recepción de un comando UI será tratada de acuerdo con el procedimiento para la transmisión de información sin acuse de recibo descrito en 3.11.

3.13.2 PROCEDIMIENTO DE LIBERACION DEL MODO MULTITRAMA

Una entidad de la capa 3 indicará su solicitud para la terminación del modo de funcionamiento multitrama entre la red y un determinado equipo de usuario mediante la primitiva ED-LIBERACION-PETICION.

Durante la ejecución de este procedimiento no se admitirán más que tramas no

numeradas (formato U), ignorándose cualquier otra.

Cuando se inicia el procedimiento de liberación del modo de funcionamiento todas las posibles primitivas ED-DATOS-PETICION y colas de tramas I pendientes serán despreciadas.

Cuando se detecte una situación no recuperable de desactivación de la capa física, la entidad de la capa de enlace de datos despreciará todas las posibles colas de tramas I pendientes e informará de ello a la capa 3 mediante una primitiva ED-LIBERACION-CONFIRMACION si existe una primitiva ED-LIBERACION-PETICION pendiente, en caso contrario lo hará mediante la primitiva ED-LIBERACION-INDICACION.

Una entidad de la capa 2 solicita la iniciación del procedimiento de liberación del modo de funcionamiento mediante la transmisión del comando de desconexión DISC, con el bit P puesto a "1". Con la transmisión de este comando se arrancará el temporizador T200 y se pondrá a cero el contador de retransmisiones.

Una entidad de la capa 2 al recibir un comando DISC estando en el estado "multitrama establecida" o de "recuperación del temporizador" transmitirá una respuesta UA con el bit F puesto a "1" (bit F del mismo valor que el bit P del comando DISC). La entidad de la capa 3 recibirá notificación de este evento mediante la primitiva ED-LIBERACION-INDICACION. La entidad de la capa de enlace de datos retornará al estado "IET asignado".

La entidad generadora del comando DISC, al recibir una trama UA con el bit F puesto a "1" como respuesta al comando transmitido o cuando recibe una respuesta DM con el bit F puesto a "1" generada por la entidad par de la capa 2 para indicar que ya se encuentra en el estado de "IET asignado", también regresará al estado "IET asignado" y anulará el temporizador T200, dando así por finalizado el modo de funcionamiento. En estas condiciones, la entidad de la capa de enlace de datos que transmitió el comando DISC, se encuentra en el estado "IET asignado" y se le

indicará a la entidad de la capa 3 mediante la primitiva ED-LIBERACION-CONFIRMACION.

Si el temporizador T200 arrancado en el lado generador del comando DISC, expira antes de recibir una respuesta UA o DM con el bit F puesto a "1" se llevarán a cabo las acciones siguientes:

-Retransmitir el comando DISC con el bit P puesto a "1".

-Rearrancar el temporizador T200.

-Incrementar el contador de retransmisiones.

Si la entidad de la capa 2 solicitante de la desconexión no recibe una respuesta correcta después de N200 intentos, informará de este hecho a la entidad de gestión de la conexión mediante la primitiva GED-ERROR-INDICACION y retornará al estado "IET asignado", notificándoselo a la entidad de la capa 3 mediante la primitiva ED-LIBERACION-CONFIRMACION.

3.13.3 EVENTOS EN EL ESTADO IET ASIGNADO

Cuando una entidad de la capa de enlace de datos se encuentra en el estado "IET asignado" está dispuesto para llevar a cabo el establecimiento del modo multitrama mediante la transmisión o recepción de una trama comando SABME de acuerdo con el procedimiento ya descrito anteriormente.

En este estado también pueden tener lugar otros eventos que requerirán una actuación específica tal como se indica en los puntos siguientes:

-La recepción de un comando DISC dará lugar a una respuesta DM, con el bit F puesto al mismo valor que el bit P contenido en el comando DISC.

-Tras la recepción de una respuesta no solicitada DM con el bit F puesto a "0", la entidad de la capa de enlace de datos, si está en condiciones de hacerlo, iniciará el establecimiento del modo de funcionamiento mediante la transmisión del comando SABME. En caso contrario la respuesta DM recibida será ignorada.

-Tras la recepción de cualquier respuesta no solicitada UA se transmitirá una primitiva GED-ERROR-INDICACION, indicando la posibilidad de una múltiple asignación de valores IET.

-Las tramas UI recibidas serán tratadas de acuerdo con el procedimiento de transferencia de información sin acuse de recibo descrito en la sección 12.

-Cualquier otro tipo de trama recibida en este estado será despreciada sin llevar a cabo acción alguna.

3.13.4 SITUACIONES DE COLISION

El desarrollo de los procedimientos anteriores puede dar lugar a situaciones de colisión de comandos y respuestas, es decir, a la transmisión de tramas no numeradas simultáneamente y en sentidos diferentes sobre la misma conexión de enlace de datos. En los puntos siguientes se describe la resolución de dichas situaciones por las entidades de la capa de enlace de datos.

3.13.4.1 Colisión de comandos idénticos

Si la colisión se produce entre comandos no numerados, de modo que el transmitido y el recibido es el mismo (SABME o DISC) las entidades de la capa de enlace de datos transmitirán una trama respuesta UA en la oportunidad más próxima. La recepción de la respuesta UA hará entrar a la entidad de la capa 2 en el estado correspondiente, "multitrama establecida" en el caso de colisión entre comandos SABME o "IET asignado" en el caso de colisión entre comandos DISC las entidades de la capa 2 de ambos lados del interfaz usuario/red avisarán a sus respectivas entidades de la capa 3 mediante la transmisión de la primitiva confirmación adecuada tras la resolución de la colisión y haber alcanzado el estado correspondiente, ED-ESTABLECIMIENTO-CONFIRMACION Y ED-LIBERACION-CONFIRMACION en los casos de colisión entre comandos SABME O DISC respectivamente.

3.13.4.2 Colisión de comandos diferentes

Cuando los comandos no numerados transmitido y recibido que dan lugar a la

colisión son diferentes (SAMBE o DISC), las entidades de la capa de enlace de datos enviará una respuesta DM en la oportunidad más próxima.

Tras la recepción de la respuesta DM con el bit F puesto a "1", la entidad de la capa de enlace de datos entrará en el estado "IET asignado", e informará a la entidad de la capa 3 mediante la primitiva adecuada, es decir, ED-LIBERACION-INDICACION para la entidad receptora del comando DISC Y ED-LIBERACION-CONFIRMACION para la entidad transmisora del comando DISC.

3.13.4.3 Colisión de una respuesta DM no solicitada a un comando SABME o DISC

Cuando una entidad de la Capa 2 recibe una respuesta DM con el bit F puesto a "0" puede producirse una colisión de un comando SABME o DISC transmitido con la respuesta DM no solicitada. Para evitar un error de interpretación de la respuesta DM recibida, los comandos SABME y DISC se envían siempre con el bit P puesto a"1". Las entidades de la capa 2 transmisora de los comandos ignorarán estas respuestas DM con el bit F puesto a "0" con lo que se alcanza la resolución de la colisión.

3.14 PROCEDIMIENTOS PARA LA TRANSFERENCIA DE INFORMACION EN EL MODO MULTITRAMA

El término "transmisión" de una trama I se refiere al envío de una trama I por la entidad de la capa de enlace de datos hacia la de la capa física.

Las entidades de la capa de enlace de datos que han alcanzado el estado "multitrama establecida" tras la transmisión o recepción de la respuesta UA, están en condiciones de comenzar la transmisión de tramas I y tramas de supervisión de acuerdo con los procedimientos que se describen en los apartados siguientes. Además del estado mencionado de "multitrama establecida" se incluyen tres nuevos estados que se definen seguidamente:

Ocupación local: estado en el cual la entidad de la capa 2 no tiene capacidad para recibir nuevas tramas I, condición de ocupado en recepción de la entidad propia,

condición que ha sido indicada a la entidad remota mediante la transmisión de una trama RNR. En este estado la entidad de la capa 2 aún tiene capacidad para la transmisión de tramas I y tramas de supervisión.

Ocupación remota: estado en el cual la entidad de la capa 2 remota no tiene capacidad para recibir nuevas tramas I, condición de ocupado en recepción de la entidad par remota, condición que ha sido conocida por la recepción de una trama RNR. En este estado la entidad de la capa 2 aún tiene capacidad para la transmisión de tramas I y tramas de supervisión.

Ocupación local y remota: estado en el cual ninguna de las entidades de la capa 2 tiene capacidad para la recepción de más tramas I, condición que habrá sido indicada por ambas partes a la entidad par remota mediante la transmisión de una trama RNR. En este estado solamente existe la capacidad para la transferencia de tramas de supervisión.

Durante el intercambio de tramas I las entidades de la capa 2 se encontrarán en el estado "multitrama establecida". Al recibir un comando SABME en este estado, la entidad de la capa 2 deberá continuar el procedimiento de restablecimiento del modo multitrama descrito en 3.15.

3.14.1 TRANSMISION DE TRAMAS I

La información que las entidades de la capa de enlace de datos reciben procedente de la capa 3 mediante la primitiva ED-DATOS-PETICION deberá ser transmitida mediante tramas I. Los parámetros número de secuencia en transmisión, N(T), y número de secuencia en recepción, N(R), del campo de control de la trama tomarán los valores actuales de las variables de estado en transmisión y recepción V(T) y V(R), respectivamente. El valor de la variable de estado de transmisión V(T) se incrementará en una unidad al final de la transmisión de cada nueva trama I.

Si la variable V(T) es igual a V(A) más "k" (donde "k" es el máximo número de tramas I pendientes da reconocimiento, que será "1" en el caso de accesos básicos

de usuario, canal D de 16 kbit/s, y "7" en el caso de accesos de usuario de jerarquía primaria, canal D de 64 kbit/s), la entidad de la capa de enlace de datos detendrá la transmisión de nuevas tramas I, pero en cambio sí tendrá capacidad para retransmitir tramas I como consecuencia de un procedimiento de recuperación de errores.

Cuando una de las entidades de la capa 2 de la conexión de enlace de datos (lado de red o lado de usuario) se encuentre en la condición de ocupado en recepción de la entidad propia, por lo que no tendrá capacidad para recibir nuevas tramas I, aún conservará la capacidad de transmisión de tramas I siempre y cuando su entidad par no se encuentre en la condición de ocupado en recepción de la entidad par remota.

Todas las primitivas ED-DATOS-PETICION que se reciban durante la condición de recuperación del temporizador serán puestas en cola de espera para ser atendidas posteriormente cuando dicha condición haya sido recuperada.

3.14.2 RECEPCION DE TRAMAS

Independientemente de la posible existencia de una condición de recuperación de la temporización, cuando una entidad de la capa 2 no se encuentra en la condición de ocupado en recepción de la entidad propia y recibe una trama I cuyo número de secuencia en transmisión N(t) coincide con el valor de la variable de estado en recepción V(R) llevará a cabo las acciones siguientes:

-Enviar el campo de información contenido en la trama I a la capa 3 utilizando para ello la primitiva ED-DATOS-INDICACION.

-Incrementar en una unidad la variable de estado de recepción V(R), actuando posteriormente tal como se describe seguidamente:

1. Si el bit P contenido en el campo de control de la trama I recibida estaba puesto a "1", la entidad de la capa de enlace de datos responderá a su entidad par remota en uno de los modos siguientes:

-Si la entidad de la capa 2 receptora de la trama I no ha pasado a la condición de ocupado en recepción de la entidad propia enviará una trama respuesta RR con el

bit F puesto a "1".

-Si la entidad de la capa 2 receptora de la trama I ha pasado a la condición de ocupado en recepción de la entidad propia, al recibir la nueva trama I. enviará una respuesta RNR con el bit F puesto a "1".

2. Si el bit P contenido en el campo de control de la trama I recibida estaba puesto a "0", suponiendo que la entidad de la capa 2 no ha pasado a la condición de ocupado en recepción de la entidad propia, se llevará a cabo una de las acciones siguientes:

-Si no existe una trama I disponible para ser transmitida o existiendo, si se ha detectado la condición de ocupado en recepción de la entidad par remota, la entidad de la capa 2 transmitirá una respuesta RR con el bit F puesto a "0".

-Si existiese una trama I disponible para la transmisión y no se ha detectado la condición de ocupado en recepción de la entidad par remota, la entidad de la capa 2 transmitirá la trama I con el valor N(R) actualizado con el valor V(R).

3. Si al recibir una trama I con el bit P puesto a "0" la entidad de la capa 2 se encuentra en la condición de ocupado en recepción de la entidad propia, transmitirá una respuesta RNR con el bit F puesto a "0".

Cuando la entidad de la capa de enlace de datos se encuentra en la condición de ocupado en recepción de la entidad propia procesará las tramas I recibidas de acuerdo con el procedimiento descrito en 3.14.6 (condición de ocupado en recepción de la entidad propia de la capa de enlace de datos).

3.14.3 ACUSE DE RECIBO DE TRAMAS

3.14.3.1 Envio del acuse de recibo de tramas

Siempre que una entidad de la capa de enlace de datos transmita una trama I o una trama de supervisión (RR, RNR o REJ), el número de secuencia en transmisión, N(R), contenido en el campo de control de dicha trama será puesto al valor actual de la variable de estado en recepción V(R), indicando de este modo el acuse de recibo

de todas las tramas I recibidas con un número de secuencia en transmisión, N(T), hasta el valor N(R)-1 (incluido éste).

3.14.3.2 Recepción del acuse de recibo de tramas

Al recibir una trama I válida o una trama de supervisión (RR, RNR o REJ), incluso en la condición de ocupado en recepción de la entidad propia o en la condición de recuperación de la temporización, la entidad de la capa de enlace de datos considerará el valor N(R) contenido en la trama como un acuse de recibo de todas las tramas I que han sido transmitidas con un N(T) hasta el valor N(R)-1 (incluido éste). El valor de la variable de estado de acuse de recibo V(A) será actualizada con el valor N(R) recibido.

La entidad de la capa 2 habrá anulado el temporizador T200 con la recepción de una trama I o de supervisión válida con un valor N(R) superior al valor actual V(A), realizándose realmente una función de acuse de recibo de alguna trama I, o una trama REJ con un N(R) igual al V(A).

Si previamente había sido transmitida una trama de supervisión con el bit P puesto a "1" y aún no se ha recibido su acuse de recibo, en las condiciones anteriores el temporizador T200 no se detendrá.

El temporizador T200 tampoco se detendrá tras la recepción de una trama I válida, si la entidad de la capa de enlace de datos conoce la condición de ocupado en recepción de la entidad par remota.

Si con la recepción de una trama I, RR o RNR ha sido anulado el temporizador T200 y existen aún tramas I pendientes de acuse de recibo, el temporizador debe ser rearrancado de nuevo. Si en estas condiciones el temporizador expira, la entidad de la capa 2 deberá seguir el procedimiento de recuperación de las tramas pendientes de acuse de recibo, definido en 3.14.7.

Si el temporizador T200 se ha anulado como consecuencia de la recepción de una trama REJ, la entidad de la capa 2 debe seguir los procedimientos de

retransmisión descritos en el apartado 15.4 de esta sección.

3.14.4 RECEPCION DE TRAMAS REJ

Tras la recepción de una trama REJ válida, la entidad de la capa de enlace de datos debe actuar tal como se indica a continuación:

a) Si no se encuentra en la condición de recuperación de la temporización:

-Anular, si existiese, la condición de ocupado en recepción de la entidad par remota.

-Actualizar sus variables de estado V(T) y V(A) con el valor N(R) contenido en el campo de control de la trama REJ recibida.

-Detener el temporizador T200.

- Arrancar el temporizador T203.

-Si se trataba de una trama comando REJ con el bit P puesto a "1", transmitir la respuesta de supervisión adecuada con el bit F puesto a "1" de acuerdo con los cuatro principios siguientes:

. Si la entidad de la capa de enlace de datos no se encuentra en la condición de ocupado en recepción de la entidad propia y se encuentra en la condición de excepción de rechazo de trama, es decir, se ha recibido un error de secuencia N(T) y se ha transmitido una trama REJ, pero la trama I solicitada aún no se ha recibido, la trama de supervisión adecuada es RR.

. Si la entidad de la capa de enlace de datos no se encuentra en la condición de ocupado en recepción de la entidad propia pero se encuentra en una condición de excepción de error en el numero de secuencia N(T), es decir, se ha recibido un error de secuencia N(T) pero aún no se ha transmitido la trama REJ, la trama de supervisión adecuada es REJ.

. Si la entidad de la capa de enlace de datos se encuentra en la condición de ocupado en recepción de la entidad propia, la trama de supervisión adecuada es

RNR.

. En todos los demás casos, la trama de supervisión adecuada es RR.

-Transmitir la trama I correspondiente lo antes posible teniendo en cuenta los cuatro principios siguientes:

. Si la entidad de la capa de enlace de datos se encuentra transmitiendo una trama de supervisión cuando se recibe la trama REJ, debe completarse la transmisión de dicha trama antes de comenzar la retransmisión de las tramas I solicitadas.

. Si la entidad de la capa de enlace de datos se encuentra transmitiendo una trama SABME, DISC, UA o DM cuando se recibe la trama REJ, debe ignorarse dicha solicitud de retransmisión de tramas I.

. Si la entidad de la capa de enlace de datos no se encuentra transmitiendo ninguna trama cuando se recibe la trama REJ, debe iniciarse inmediatamente la retransmisión de tramas I solicitada.

. Cuando se inicia la retransmisión de tramas I, deberán retransmitirse todas las tramas I pendientes del acuse de recibo comenzando con la trama I identificada por el N(R) contenido en el campo de control de la trama REJ. Todas las tramas I pendientes de ser transmitidas serán transmitidas después de la retransmisión de tramas I solicitada.

-Si se trataba de una trama respuesta REJ con el bit F puesto a "1" notificar la violación del protocolo a la entidad de gestión de la conexión mediante la primitiva GED-ERROR-INDICACION.

b) Si se encuentra en la condición de recuperación de la temporización y se trataba de una trama respuesta REJ con el bit F puesto a "1":

- Anular, si existiese, la condición de ocupado en recepción de la entidad par remota.

-Actualizar las variables de estado V(T) y V(A) con el valor N(R) contenido en el

campo de control de la trama REJ recibida.

-Detener el temporizador T200.

-Arrancar el temporizador T203.

-Pasar al estado "multitrama establecida".

-Transmitir la trama I correspondiente lo antes posible teniendo en cuenta los cuatro principios citados anteriormente (bajo el condicionante "a").

c) Si se encuentra en la condición de recuperación de la temporización y se trataba de una trama comando REJ o de una respuesta con el bit F puesto a "0":

-Anular, si existiese, la condición de ocupado en recepción de la entidad par remota.

-Actualizar el valor de la variable de estado V(A) con el valor N(R) contenido en el campo de control de la trama REJ recibida.

-Si se trataba de una trama comando REJ con el bit P puesto a "1", transmitir la respuesta de supervisión adecuada con el bit F puesto a "1" de acuerdo con los cuatro principios citados anteriormente (bajo el condicionante "a").

3.14.5 RECEPCION DE TRAMAS RNR

Este procedimiento cubre la consulta de la condición de ocupado o disponible de las entidades remotas conocido como proceso de consulta.

Después de la recepción de una trama RNR válida, ya sea comando o respuesta y si la entidad de la capa de enlace de datos no está implicada en un procedimiento de establecimiento del modo de funcionamiento ésta debe considerar a la entidad de la capa 2 par en la condición de ocupado en recepción de la entidad par remota, llevando a cabo una de las acciones siguientes:

-Si la trama RNR recibida fue un comando con el bit P puesto a "1", la entidad de la capa 2 debe responder con una trama RR con el bit F puesto a "1", si no se encuentra también en la condición de ocupado en recepción de la entidad propia en

cuyo caso transmitiría una respuesta RNR con el bit F puesto a "1"

-Si la trama RNR recibida fue una respuesta con el bit F puesto a "1" la posible condición existente de recuperación de la temporización debe ser anulada y el número de secuencia N(R) contenido en la respuesta RNR debe ser utilizado para actualizar la variable de estado en transmisión V(T).

Debe tenerse en cuenta que el valor N(R) contenido en cualquier comando RR o RNR, independientemente del valor del valor del bit P, no podrá ser utilizado para actualizar la variable de estado V(T).

La entidad de la capa de enlace de datos al recibir una trama RNR llevará a cabo las acciones siguientes:

-Considerar el valor N(R) contenido en la trama RNR recibida como un acuse de recibo de todas las tramas I previamente transmitidas con un valor N(T) hasta el valor N(R)-1, incluido éste. La variable de estado del reconocimiento V(A) por tanto deberá ser actualizada con el valor N(R) contenido en la trama RNR.

-Rearrancar el temporizador T200, salvo en el caso que se esté esperando la recepción de una respuesta de supervisión con el bit F puesto a "1".

Si el temporizador T200 expira, la entidad de la capa de enlace de datos llevará a cabo las acciones siguientes:

-Si no está ya implicada en la condición de recuperación de la temporización entrar en la condición de recuperación de la temporización. El contador de retransmisión deberá ser puesto a cero.

-Si ya se encuentra en la condición de recuperación de la temporización incrementar en una unidad el contador de retransmisiones.

Seguidamente la entidad de la capa de enlace de datos llevará a cabo las acciones siguientes:

-Si el contador de retransmisiones aún no ha alcanzado el valor N200 transmitirá el comando de supervisión RR, RNR o REJ adecuado con el bit P puesto

a "1" de acuerdo con los cuatro principios descritos en 3.14.4 (bajo el condicionante "a") y rearrancará el temporizador T200.

-Si el contador de retransmisiones ya ha alcanzado el valor N200, la entidad de la capa 2 debe iniciar el procedimiento de restablecimiento del modo multitrama que se describe en 3.15. En este caso la entidad de gestión de la conexión deberá ser notificada mediante la primitiva GED-ERROR-INDICACION.

La entidad de la capa de enlace de datos par al recibir la trama de supervisión RNR con el bit P puesto a "1" deberá responder en la primera oportunidad disponible con la respuesta de supervisión RR, RNR o REJ adecuada con el bit F puesto a "1" de acuerdo con los cuatro principios descritos en 3.14.4 (bajo el condicionante "a"), indicando si existe o no la condición de ocupado en recepción de la entidad propia.

Al recibir una respuesta de supervisión con el bit F puesto a "1" la entidad de la capa 2 debe anular el temporizador T200 y llevar a cabo una de las acciones siguientes:

-Si la respuesta es una trama RR o REJ, anular la condición de ocupado en recepción de la entidad par remota por lo que la entidad de la capa 2 ya puede transmitir nuevas tramas I o retransmitirlas en su caso.

-Si la respuesta es una trama RNR, la entidad par aún permanece en la condición de ocupado en recepción de la entidad par remota, por lo que debe procederse de acuerdo con el procedimiento descrito en 3.14.5.

Si se recibe una trama comando de supervisión (RR, RNR o REJ) o una respuesta de supervisión (RR, RNR o REJ) con el bit F puesto a "0" durante el proceso de consulta de estado, la entidad de la capa de enlace de datos llevará a cabo una de las acciones siguientes:

-Si la trama de supervisión es un comando RR o REJ o una respuesta RR o REJ con el bit F puesto a "0", la condición de ocupado en recepción de la entidad par remota debe ser anulada; y si la trama de supervisión fue un comando con el bit P

puesto a "1" deberá transmitirse la respuesta de supervisión adecuada con el bit F puesto a "1" de acuerdo con los cuatro principios descritos en 3.14.4 (bajo el condicionante "a"). Sin embargo, la transmisión o retransmisión de tramas I no deberá iniciarse hasta el momento en que se reciba una respuesta de supervisión con el bit F puesto a "1" o hasta que expire el temporizador T200.

-Si la trama de supervisión es un comando RNR o una respuesta RNR con el bit F puesto a "0" debe mantenerse la condición de ocupado en recepción de la entidad par remota; y si la trama recibida fue un comando RNR con el bit P puesto a "1", deberá transmitirse la respuesta de supervisión adecuada de acuerdo con los cuatro principios descritos anteriormente en 3.14.4 (bajo el condicionante "a").

Tras la recepción de una trama comando SABME, la entidad de la capa de enlace de datos deberá anular siempre la condición de ocupado en recepción de la entidad par remota.

3.14.6 CONDICION DE OCUPADO EN RECEPCION DE LA ENTIDAD PROPIA DE LA CAPA DE ENLACE DE DATOS

Cuando una entidad de la capa de enlace de datos pasa a la condición de ocupado en recepción de la entidad propia debe transmitir una trama RNR en la primera oportunidad disponible.

La trama RNR podrá ser una de las siguientes:

-Una trama respuesta RNR con el bit F puesto a "0"

-Si se ha pasado a esta condición con la recepcion de una trama comando con el bit P puesto a "1" la respuesta será una trama RNR con el bit F puesto a "1"

-Si se ha pasado a esta condición como consecuencia de expirar el temporizador T200 se transmitirá una trama comando RNR con el bit P puesto a "1".

Bajo la condición de ocupado en recepción de la entidad propia, todas las tramas I que se reciban con el bit P puesto a "0" serán despreciadas tras la actualización de la variable de estado V(A), y todas las tramas de supervisión que se

reciban con el bit P/F puesto a "0" serán procesadas incluyendo la actualización de la variable de estado V(A).

Bajo la condición de ocupado en recepción de la entidad propia, todas las tramas I que se reciban con el bit P puesto a "1" serán despreciadas tras la actualización de la variable de estado V(A), sin embargo se deberá transmitir una trama respuesta RNR con el bit F puesto a "1" al igual que en el caso en que se hubiese recibido una trama de supervisión con el bit P puesto a "1" que será también. procesada incluyendo la actualización de la variable de estado V(A).

Para indicar a la entidad de la capa de enlace de datos par remota la anulación de la condición de ocupado en recepción de la entidad propia deberá transmitirse una trama RR, o una trama REJ con el valor N(R) puesto al valor actual de la variable V(R) si aún no se hubiese informado de un error de secuencia N(T) previamente detectado.

La transmisión de un comando SABME o una respuesta UA como consecuencia de un comando SABME recibido, también indicará la anulación de la condición de ocupado en recepción de la entidad propia.

3.14.7 ACUSE DE RECIBO PENDIENTE

Este procedimiento cubre el tratamiento del temporizador T200 conocido como procedimiento de recuperación de la temporización.

La entidad de la capa de enlace de datos mantendrá una variable interna de cuenta de las retransmisiones de trama efectuadas.

Cuando el temporizador T200 expira, la entidad de la capa 2 llevará a cabo una de las acciones siguientes:

-Si no se encuentra aún en el estado de recuperación de la temporización entrar en él e iniciar el contador de retransmisiones.

-Si ya se encuentra en el estado de recuperación de la temporización se incrementará en una unidad el valor del contador de retransmisiones.

Seguidamente la entidad de la capa 2 en función del valor del contador de retransmisiones llevará a cabo una de las acciones siguientes:

a) Si el valor del contador de retransmisiones es inferior a N200:

-Rearrancar el temporizador T200.

-Transmitir un comando de supervisión adecuado (RR, REJ o RNR) de acuerdo con los cuatro principios descritos anteriormente en 3.14.4 (bajo el condicionante "a") con el bit P puesto a "1" o retransmitir la última trama I (V(T)-1) con el bit P puesto a "1"

b) Si el contador de retransmisiones alcanza el valor N200, se iniciará el procedimiento de restablecimiento del modo multitrama descrito en 3.15, indicando este evento a la entidad de gestión de la conexión mediante la primitiva GED-ERROR-INDICACION.

El estado de recuperación de la temporización es anulado cuando la entidad de la capa 2 recibe una respuesta de supervisión valida con el bit F puesto a "1". Si el valor N(R) recibido en la trama de supervisión está contenido en el margen comprendido entre el valor actual de la variable V(A) y el valor de la variable V(T), la entidad de la capa 2 actualizará el valor de la variable V(T) con el valor del número de secuencia en recepción N(R) recibido.

El temporizador T200 se detendrá si la respuesta de supervisión recibida es una trama RR o REJ comenzándose de nuevo la transmisión o retransmisión, según proceda, de tramas I. Si la respuesta es una trama RNR el temporizador T200 deberá ser detenido y rearrancado, para llevar a cabo el proceso de consulta tal como se indicó en 3.14.5.

3.15 PROCEDIMIENTO PARA EL RESTABLECIMIENTO DEL MODO MULTITRAMA

El procedimiento para el restablecimiento del modo de funcionamiento multitrama es iniciado como consecuencia de la aparición de una de las condiciones

siguientes:

-La recepcion de un comando SABME durante la fase de operación normal en el modo multitrama (entidades de la capa de enlace de datos en el estado "multitrama establecida").

-La recepción de una primitiva ED-ESTABLECIMIENTO-PETICION procedente de la entidad de la capa 3.

-Cuando ocurren N200 fallos de retransmisión mientras la entidad de la capa de enlace de datos se encuentra en la condición de recuperación de la temporización.

-Aparición de una de las condiciones de rechazo de trama.

. Recepcion de una trama no definida.

. Recepción de una trama no numerada o de supervisión con una longitud incorrecta.

. Recepción de un número de secuencia en recepción N(R) incorrecto.

. Recepcion de una Trama I con un campo de información que excede la máxima longitud permitida.

-Recepción, durante el modo de operación multitrama establecida, de una trama respuesta FRMR.

-Recepción, durante el modo de operación de multitrama establecida, de una trama respuesta DM no solicitada con el bit F puesto a "0".

-Recepción durante la condición de recuperación de la temporización de una trama respuesta DM con el bit F puesto a "1"

Bajo cada una de las condiciones de restablecimiento. la entidad de la capa de enlace de datos que la detecta deberá seguir los procedimientos descritos en 3.13.1 (procedimiento de establecimiento del modo multitrama), por tanto deberá ser transmitida la trama SABME.

En el caso de que la entidad de la capa de enlace de datos remota sea la que

inicie el restablecimiento del modo de funcionamiento, la entidad de la capa de enlace de datos además deberá:

-Transmitir una primitiva GED-ERROR-INDICACION hacia la entidad de gestión de la conexión.

-Si se cumple la condición V(T) > V(A) antes de llevar a cabo al restablecimiento se transmitirá la primitiva ED-ESTABLECIMIENTO-INDICACION hacia la entidad de la capa 3 y deberán despreciarse todas las posibles colas de tramas I pendientes de transmisión.

En el caso de que sea la entidad de la capa 3 la que inicie el restablecimiento, o en el caso de que se reciba una primitiva ED-ESTABLECIMIENTO-PETICION durante la fase de restablecimiento se transmitirá la primitiva ED-ESTABLECIMIENTO-CONFIRMACION hacia la entidad de la capa 3.

3.16 PROCEDIMIENTO PARA LA SUPERVISION DE LA CONEXION DE ENLACE DE DATOS

El procedimiento aquí descrito debe ser soportado en el caso de instalaciones básicas de usuario (buses pasivos) por el lado de red de la conexión de enlace de datos y opcionalmente podría también ser soportado por los equipos terminales. En el caso de accesos de usuario de jerarquía primaria este procedimiento debe ser soportado por las entidades de la capa de enlace de datos tanto del lado de usuario como del lado de red.

Este procedimiento de supervisión proporciona los medios para llevar a cabo una supervisión del enlace de datos cuando las entidades de la capa de enlace de datos se encuentran en el estado "multitrama establecida".

La verificación de estas conexiones de enlace de datos es un servicio que las entidades de la capa de enlace de datos proporcionan a las de la capa 3, esto implica que la entidad de la capa 3 será avisada únicamente en el caso en que sea detectado algún fallo por la entidad de la capa de enlace de datos.

La supervisión de la capa de enlace de datos quedará incorporada al intercambio normal de información entre entidades de la capa 2.

El procedimiento que aquí se describe se basa en el intercambio de tramas comando de supervisión RR y RNR y en la utilización del temporizador T203 (el valor del temporizador T203 es de 10 segundos), permaneciendo las entidades de la capa de enlace de datos en el estado "multitrama establecida".

Si no existe intercambio de tramas sobre la conexión de enlace de datos (ni tramas I, ni tramas de supervisión con el bit P puesto a "1") no existe posibilidad alguna de detectar la existencia de un fallo en dicha conexión de enlaces de datos o si alguno de los equipos de usuario se ha desconectado. El temporizador T203, implicado en el procedimiento, indica el máximo tiempo permisible sin intercambio de tramas en el enlace de datos.

Si el temporizador T203 expira se transmitirá un comando de supervisión con el bit P puesto a "1". Este procedimiento está protegido contra los posibles errores de transmisión mediante el temporizador normal T200 incluyendo las N200 retransmisiones.

El procedimiento para la supervisión de la conexión del enlace de datos se basa en los siguientes principios:

a) Arrancar el temporizador T203

El temporizador T203 es arrancado:

-Cuando la entidad de la capa de enlace de datos entra en el estado "multitrama establecida".

-En el estado "multitrama establecida" siempre que el temporizador T200 es detenido.

Tras la recepción de una Trama I o de supervisión el temporizador T203 será arrancado si el temporizador T200 no va a ser arrancado.

b) Detener el temporizador T203

El temporizador T203 es detenido:

-En el estado "multitrama establecida" cuando el temporizador T200 es arrancado.

-Al abandonar el estado "multitrama establecida"

Estas dos condiciones significan que el temporizador T203 se arranca solamente cuando el temporizador T200 es detenido pero no rearrancado.

c) El temporizador T203 expira:

Si el temporizador T203 expira, la entidad de la capa de enlace de datos llevará a cabo las acciones que se indican a continuación (en este momento el temporizador T200 no está corriendo ni ha expirado):

-Poner el contador de retransmisiones a cero.

-Entrar en la condición de recuperación de la temporización.

-Transmitir un comando de supervisión con el bit P puesto a "1" teniendo en cuenta las condiciones siguientes:

a) Si la entidad transmisora no se encuentra en la condición de ocupado en recepción de la entidad propia transmitirá un comando RR.

b) Si la entidad transmisora se encuentra en la condición de ocupado en recepción de la entidad propia transmitirá un comando RNR.

-Arrancar el temporizador T200.

-Enviar la primitiva GED-ERROR-INDICACION hacia la entidad de gestión de la conexión después de efectuar N200 retransmisiones.

3.17 ERRORES DE PROCEDIMIENTO DE LA CAPA DE ENLACE DE DATOS

Las condiciones consideradas como de excepción pueden ocurrir como consecuencia de la aparición de errores de la capa física o de errores de procedimiento de la capa de enlace de datos.

A continuación se describen las diferentes condiciones de excepción posibles así como los procedimientos necesarios para su recuperación por la entidad de capa de enlace de datos

3.17.1 CONDICIONES DE RECEPCION, NOTIFICACION Y RECUPERACION

3.17.1.1 Error en el número de secuencia en transmisión, N(T)

La condición de excepción definida como error en el número de secuencia en transmisión N(T) contenido en el campo de control de las tramas I tiene lugar cuando se recibe una trama válida I pero cuyo valor N(T) no coincide con el valor esperado y que está indicado en el contenido de la variable de estado en recepción V(R) de la entidad de la capa 2 receptora. En este caso, cuando el valor N(R) no coincide con el V(R), el contenido del campo de información de la trama T no será tenido en cuenta.

La entidad de capa 2 receptora no efectuará el acuse de recibo de la trama I causando el error de secuencia, es decir, no incrementará el contenido de la variable V(R), actuando del mismo modo con todas las posibles tramas I posteriores hasta que se reciba una trama con un valor correcto del número de secuencia N(T).

Una entidad de la capa de enlace de datos que reciba una o más tramas I con errores de secuencia pero libres de otro tipo de errores, o tramas de supervisión posteriores (RR, REJ o RNR), utilizará la información de control contenida en el subcampo N(R) de dichas tramas así como el bit P/F para llevar a cabo las acciones de control características de la capa 2, por ejemplo para recibir acuse de recibo de las tramas I previamente transmitidas y transmitir tramas respuesta adecuadas cuando el bit P recibido está puesto a "1". Por ello, las tramas I retransmitidas pueden contener un valor N(R) y un bit P actualizado y por tanto, diferentes de los contenidos en las tramas I transmitidas originalmente.

La trama REJ es la utilizada por la entidad de la capa de enlace de datos receptora para iniciar la recuperación de la condición de excepción (retransmisión)

después de haber detectado un error de secuencia N(T).

En un determinado momento y para una determinada dirección solamente puede establecerse una condición de rechazo (REJ).

Una entidad de la capa de enlace de datos al recibir un comando o respuesta REJ comenzará la transmisión secuencial (retransmisión) de las tramas I, comenzando por la trama I indicada con el valor N(R) contenido en la trama REJ.

Se considera que una condición de excepción REJ ha desaparecido cuando se recibe la trama I solicitada o cuando se recibe un comando SABME o DISC.

3.17.1.2 Error en el número de secuencia en recepción, N(R)

Existe una condición de excepción de número de secuencia en recepción en el transmisor de una trama, cuando se recibe una trama I o de supervisión válidas pero cuyo valor N(R) no es valido. Un valor N(R) será válido si se encuentra contenido en el margen V(A) < N(R) <= V(T).

El campo de información contenido en una trama I recibida de un modo correcto, en secuencia y formato, será entregado a la capa 3 mediante la primitiva ED-DATOS-INDICACION.

La entidad de la capa 2 al detectar esta condición de excepción del número de secuencia N(R) la notificará a la entidad de gestión de la conexión mediante la primitiva GED-ERROR-INDICACION y comenzará el restablecimiento del modo de funcionamiento mediante la transmisión del comando SABME tal como se ha descrito en 3.15.

3.17.1.3 Recuperación de la temporización (T200)

Si una entidad de la capa de enlace de datos, debido a un error de transmisión no recibe una trama I simple o la última de una secuencia de tramas I, no detectará, por tanto, la condición de excepción de fuera de secuencia y por tanto no podrá transmitir la trama REJ.

La entidad transmisora de la trama (o tramas) I pendiente del acuse de recibo

deberá llevar a cabo las acciones adecuadas de recuperación de la condición de excepción cuando expire el temporizador T200, con el fin de detectar en qué trama I debe comenzar la retransmisión. El procedimiento que debe ser llevado a cabo en este caso es el descrito en 3.14.7 donde se describe el procedimiento de acuse de recibo de tramas pendientes.

3.17.1.4 Condición de tramas no válidas

Cualquier trama recibida que se considere inválida, tal como se ha definido en 3.7.4, no será tenida en cuenta y por tanto no se llevará a cabo acción alguna como consecuencia de dicha trama.

3.17.1.5 Condición de rechazo de trama

Una condición de rechazo de trama resulta como consecuencia de la aparición de una de las siguientes condiciones:

-Recepción de una trama no definida, es decir, una trama cuya codificación no coincide con una de las especificadas en el cuadro de la figura 3.21 .

-La recepción de una trama de supervisión o no numerada con una longitud incorrecta.

-La recepción de un valor de número de secuencia en recepción N(R) no válido.

-La recepción de una trama con un campo de información que excede la máxima longitud permitida.

Tras la detección de una condición de rechazo de trama la entidad de la capa de enlace de datos llevará a cabo las acciones siguientes:

-Transmitir una primitiva GED-ERROR-INDICACION.

-Iniciar el procedimiento de restablecimiento del modo de funcionamiento de acuerdo con lo descrito en 3.15.

Una vez detectado el evento de rechazo de trama durante el establecimiento o liberación del modo de funcionamiento multitrama, o cuando el enlace de datos no

está establecido la entidad de la capa de enlace de datos llevará a cabo las acciones siguientes:

-Transmitir una primitiva GED-ERROR-INDICACION

-Despreciar la trama.

Para un funcionamiento correcto es fundamental que el receptor de tramas pueda diferenciar sin ambigüedad entre tramas no válidas y tramas con un campo de información que supera el máximo tamaño permitido. Se puede asumir que existe una trama ilimitada, y por tanto será despreciada, si se recibe el doble de los octetos correspondientes a la máxima longitud de trama permisible más dos octetos, sin haberse detectado la existencia de indicadores de trama.

3.17.1.6 Recepción de una trama respuesta FRMR

Eventualmente, la entidad de la capa de enlace de datos del lado de red podría recibir una trama respuesta FRMR (estructurada tal como se ha definido en 3.8.4), mientras se encuentra en el modo de funcionamiento multitrama, en ese caso se llevarán a cabo las acciones siguientes:

-Transmitir una primitiva GED-ERROR-INDICACION hacia la entidad de gestión de la conexión.

-Iniciar el procedimiento de restablecimiento del modo de funcionamiento de acuerdo con lo descrito en 3.15.

3.17.1.7 Tramas respuesta no solicitadas

En la tabla de la figura 3.33 se indican las acciones que deberán ser llevadas a cabo por una entidad de capa de enlace de datos tras la recepción de una trama respuesta no solicitada teniendo en cuenta los posibles estados de la entidad de la capa 2 (y la condición de recuperación de la temporización)

La entidad de la capa de enlace de datos asumirá la existencia de una asignación múltiple de valores IET tras la recepción de una respuesta UA no solicitada.

Una entidad de la capa de enlace de datos al asumir la existencia de una asignación múltiple de IET iniciara los procedimientos de recuperación que correspondan tal como están descritos en la tabla de la figura 3.33 cuando:

- Reciba una respuesta UA mientras se encuentra en el estado "multitrama establecida".

- Reciba una respuesta UA mientras se encuentra en la condición de recuperación del temporizador.

- Reciba una respuesta UA en el estado "IET".

Una entidad de la capa de enlace de datos tras detectar una asignación múltiple de IET lo notificara a la entidad de gestión de la conexión mediante la primitiva GED-ERROR-INDICACION.

		ESTADO DE LA ENTIDAD DE LA CAPA DE ENLACE DE DATOS				
		IET ASIGNADO	ESTABLECIMIENTO PENDIENTE	LIBERACION PENDIENTE	MULTITRAMA ESTABLECIDA	RECUPERACION DE LA TEMPORIZACION
TRAMA RESPUESTA NO SOLICITADA	UA (F = 1)	GED-ERROR-INDIC.	Válido	Válido	GED-ERROR-INDIC.	GED-ERROR-INDIC.
	UA (F = 0)	GED-ERROR-INDIC.	GED-ERROR-INDIC.	GED-ERROR-INDIC.	GED-ERROR-INDIC.	GED-ERROR-INDIC.
	DM (F = 1)	Ignorar	Válido	Válido	GED-ERROR-INDIC.	Válido
	DM (F = 0)	ESTABLECIMIENTO	Ignorar	Ignorar	GED-ERROR-INDIC. RESTABLECIMIENT.	GED-ERROR-INDIC. RESTABLECIMIENT.
	RR (F = 1) REJ (F = 1) RNR (F = 1)	Ignorar	Ignorar	Ignorar	GED-ERROR-INDIC.	Válido
	RR (F = 0) REJ (F = 0) RNR (F = 0)	Ignorar	Ignorar	Ignorar	Válido	Válido

Figura 3.33 TRATAMIENTO DE LAS TRAMAS RESPUESTA NO SOLICITADAS

La detección de alguna de las condiciones de error descritas anteriormente dará lugar a la generación de una primitiva GED-ERROR-INDICACION desde la entidad de la capa de enlace de datos hacia la entidad de gestión de la conexión de la capa 2.

La tabla que se muestra en la figura 3.34 indica las situaciones en las que la primitiva GED-ERROR-INDICACION será generada, primitiva que indicará a la entidad de gestión de la conexión el código de error que identificará el error que se ha producido. La tabla mencionada también indica las acciones que serán llevadas a

cabo por la entidad de gestión de la conexión tanto en el lado de usuario como en el lado de red en función del tipo de error que se ha indicado.

Teniendo en cuenta que en el caso de acceso primario del usuario (conexión de enlace de datos punto a punto) no se soportan los procedimientos entre pares de gestión de los valores IET las acciones de gestión deberán ser diferentes tal como se indica en el cuadro de la figura 3.35 .

En los cuadros mencionados se incluyen diferentes columnas con diferentes informaciones tal como se indica a continuación:

-La columna "código de error" indica el valor de identificación de cada situación de error, el cual será incluido como parámetro en la primitiva GED-ERROR-INDICACION.

-La columna "condición de error" describe el evento de error de protocolo que causa la generación de la primitiva GED-ERROR-INDICACION.

-La columna "acción de gestión del lado de red" indica la acción que deberá ser llevada a cabo por la entidad de gestión de la conexión del lado de red.

-La columna "acción de gestión del lado de usuario" indica la acción que deberá ser llevada a cabo por la entidad de gestión de la conexión del lado de usuario.

Cuando en los cuadros se indica como acción de gestión "almacenamiento del error", quiere decir que el evento incrementará un contador de errores, la longitud y el tratamiento de dicho contador será dependiente de las condiciones de fabricación del equipo de que se trate. Cuando para el lado de usuario se indica "dependiente del equipo" quiere decir que la acción a llevar a cabo dependerá de las condiciones de fabricación del equipo de usuario de que se trate, el cual podrá o no incluir el contador de errores mencionado.

Figura 3.34 ACCIONES DE GESTIÓN TRAS LA RECEPCIÓN DE LA PRIMITIVA GED-ERROR-INDICACIÓN

Tipo de error	Código de error	Condición de error	Acción de gestión del lado de red	Acción de gestión del lado de usuario
	A	trama de supervisión (F=1)	almacenamiento de error	dependiente del equipo
	B	DM (F=1)	almacenamiento del error	dependiente del equipo
recepción de una respuesta no solicitada	C D	UA (F=1) UA (F=0)	procedimiento de supresión de IET, ó procedimiento de comprobación de IET, después si el IET: - está libre:suprimir el IET. - es único: no se actua. - es múltiple: procedimiento de supresión de IET	Supresión de IET ó procedimiento de verificación de identidad IET
	E	DM(F=0)	almacenamiento del error	dependiente del equipo
restablecimiento iniciado por la entidad par	F	SABME	almacenamiento del error	dependiente del equipo
retransmisión infructuosa (N 200 veces)	G H	SABME DISC	procedimiento de supresión de IET, ó procedimiento de comprobación de IET, después si el IET: - está libre: suprimir el IET - es único: no se actua - es múltiple: procedimiento de supresión de IET	supresión de IET ó procedimiento de verificación de identidad IET
	I	consulta de estado	almacenamiento del error	dependiente del equipo
	J	error N (R)	almacenamiento del error	dependiente del equipo
	K	recepción de la respuesta FRMR	almacenamiento del error	dependiente del equipo
otras	L	recepción de una trama con un campo de control no definido	almacenamiento del error	dependiente del equipo
	M			
	N	recepción de una trama con una tamaño no excesivo	almacenamiento del error	dependiente del equipo
	O	error N201	almacenamiento del error	dependiente del equipo

Figura 3.35 ACCIONES DE GESTIÓN PARA EL CASO DE ACCESO PRIMARIO DE USUARIO TRAS LA RECEPCIÓN DE LA PRIMITIVA GED-ERROR-INDICACIÓN

Tipo de error	Código de error	Condición de error	Acción de gestión del lado de red	Acción de gestión del lado de usuario
recepción de una respuesta no solicitada	A	trama de supervisión (F=1)	almacenamiento del error	almacenamiento del error
	B	DM(F=1)	almacenamiento del error	almacenamiento del error
	C	UA(F=1)	almacenamiento del error	almacenamiento del error
	D	UA(F=0)		
	E	DM(F=0)	almacenamiento del error	almacenamiento del error
restablecimiento iniciado por la entidad par	F	SABME	almacenamiento del error	almacenamiento del error
retransmisión infructuosa (N 200 veces)	G	SABME	indicación de que es necesaria una acción de mantenimiento a que la capa 2 noppuede proporcionar servicios	indicación de que es necesaria una acción de mantenimiento a que la capa 2 no puede proporcionar servicios
	H	DISC		
	I	consulta de estado	almacenamiento del error	almacenamiento del error
	J	error N (R)	almacenamiento del error	almacenamiento del error
	K	recepción de respuesta FRMR	almacenamiento del error	almacenamiento del error
OTRAS	L	recepción de una trama con un campo de control no definido	almacenamiento del error	almacenamiento del error
	M			
	N	recepción de una trama con un tamaño excesivo	almacenamiento del error	almacenamiento del error
	O	error N 201	almacenamiento del error	almacenamiento del error

3.18 PARÁMETROS DEL SISTEMA

Los parámetros del sistema que son enumerados a continuación están asociados a cada uno de los puntos de acceso al servicio (PAS).

Estos valores de los parámetros aquí indicados son considerados como

aplicables tanto en el lado de usuario como en el lado de red del enlace de datos.

a) Temporizador T200

Indica el tiempo que una vez transcurrido debe efectuarse la retransmisión de una trama a la vez que se incrementa el contador de retransmisiones.

Para una operación correcta de los procedimientos de capa 2 se requiere que el temporizador T200 sea superior al máximo tiempo requerido entre la transmisión de un comando y la recepción de su correspondiente respuesta o trama de acuse recibo. El valor asignado a este parámetro es de un 1 segundo.

T200 = 1 segundo

b) Temporizador T201

Este temporizador representa el tiempo que debe transcurrir entre las retransmisiones del mensaje de gestión SOLICITUD DE COMPROBACION DE IDENTIDAD. El valor asignado a este parámetro es 1 segundo.

T201 = 1 segundo

c) Temporizador T202

Este temporizador representa el tiempo que debe transcurrir entre retransmisiones de los mensajes de gestión SOLICITUD DE ASIGNACION DE IDENTIDAD y VERIFICACION DE IDENTIDAD. El valor asignado a este parámetro es 2 segundos.

T202 = 2 segundos

d) Temporizador T201

Este temporizador soporta el mecanismo de supervisión de la conexión de enlace de datos, representando al máximo tiempo permitido sin tener lugar un intercambio de tramas sobre la conexión de enlace de datos. El valor asignado a este parámetro es 10 segundos.

T203 = 10 segundos

e) Máximo número de retransmisiones. N200

Este parámetro representa el máximo número de veces que una trama puede ser retransmitida antes de considerar la situación como un fallo que requiere actuaciones posteriores. El valor asignado a este parámetro es 3 veces.

N200 = 3 veces

f) Máximo número de octetos en el campo de información de una trama

Este parámetro representa la capacidad máxima, en octetos, del campo de información en una trama (UI ó I) y por tanto el tamaño máximo del mensaje de la capa 3 transportado en una primitiva hacia la capa de enlace de datos. El valor asignado a este parámetro es 260 octetos.

N201 = 260 octetos

g) Máximo número de retransmisiones del mensaje de gestión SOLICITUD DE IDENTIDAD, N202

Este parámetro representa el máximo número de veces que el mensaie de gestión SOLICITUD DE IDENTIDAD será transmitido por la entidad de gestión de capa del lado de usuario cuando éste está solicitando un valor IET. El valor asignado a este parámetro es 3 veces.

N202 = 3 veces.

h) Ventana, k

Este parámetro representa el máximo número de tramas I numeradas secuencialmente que en un determinado momento pueden estar pendientes del acuse de recibo. Este es un parámetro cuyo valor máximo vendrá limitado por el módulo de funcionamiento en el modo multitrama cuyo valor 128 no puede ser igualado ni superado. El valor asignado a este parámetro es:

Para el caso de PAS soportando señalización sobre un acceso básico de usuario (canal D de 16 kbit/s) el valor de la ventana (k_{16}) es 1 trama.

Para el caso de PAS soportando señalización sobre un acceso de usuario de jerarquía primaria (canal D de 64 kbit/s) el valor de la ventana (k_{64}) es 7 tramas.

k_{16} = 1 trama

k_{64} = 7 trama

Capítulo 4: Descripción del Entorno Global

4.1 OBJETO

El presente capítulo tiene por objeto la descripción del entorno global que dividimos en dos apartados:

El primero tratará las características generales del Microcontrolador de Comunicaciones MC68302 y aquellas relacionadas con el protocolo LAP-D (controlador HDLC).

El segundo apartado describe de forma general el módulo Kernel (núcleo).

4.2 MICROCONTROLADOR DE COMUNICACIONES 68302 DE MOTOROLA

4.2.1 Descripción General

El Microcontrolador de Comunicaciones 68302 de Motorola (MC68302) es un Procesador Multiprotocolo Integrado (IMP o Integrated Multiprotocol Processor) que constituye un dispositivo VLSI que incorpora los bloques de construcción principales requeridos para una gran variedad de controladores. Este dispositivo IMP, fuertemente acoplado, es el primero que ofrece los beneficios de un núcleo basado en el standard del microprocesador M68000/M68008 y una arquitectura de comunicaciones flexible.

Este Procesador Multiprotocolo Integrado puede ser configurado para soportar una serie de interfaces que incluyen aquellos para el acceso básico a la RDSI y

aplicaciones de adaptador de terminal.

Se puede obtener la operación concurrente de diferentes protocolos gracias a la combinación de las características de la arquitectura y de la programación. Ejemplos de aplicaciones apropiadas a este dispositivo son las siguientes: concentradores de datos, tarjetas de línea, puentes (bridges) y pasarelas (gateways).

Este IMP es un dispositivo de tecnología HCMOS que está formado por un procesador núcleo M68000, un bloque de integración de sistema (SIB o System Integration Block) y un procesador de comunicaciones (CP).

4.2.2 Diagrama de bloques

En la figura 4.1 se muestra el diagrama de bloques del MC68302.

El IMP es capaz de manejar tareas complejas como las tareas de acceso a un acceso básico RDSI (2B+D). Por ejemplo, la arquitectura del IMP y los puertos del controlador de comunicaciones serie (SCC o Serial Communication Controller) pueden funcionar como interfaces de un chip transmisor/receptor S/T y realizar la parte baja de las funciones de la capa 2 OSI de ISO (manejo de los bits).

Puede soportar también aplicaciones de adaptador de terminal usando la estructura de buffer basada en memoria flexible transformando y compartiendo la información de los buffer entre los tres puertos SCC y el puerto de comunicaciones serie (SCP o Serial Communication Port).

El MC68302 utiliza una arquitectura microprocesador en la que los dispositivos periféricos están conectados al sistema a través de una memoria de doble puerto. Algunos parámetros, contadores y toda las tablas de descriptores de buffer de memoria residen en la memoria RAM de doble puerto.

Los buffers de datos de transmisión y recepción pueden estar localizados en esta memoria RAM interna o en una memoria RAM externa del sistema.

Hay seis canales DMA dedicados a los seis puertos serie (recepción y transmisión de cada uno de los tres canales SCC).

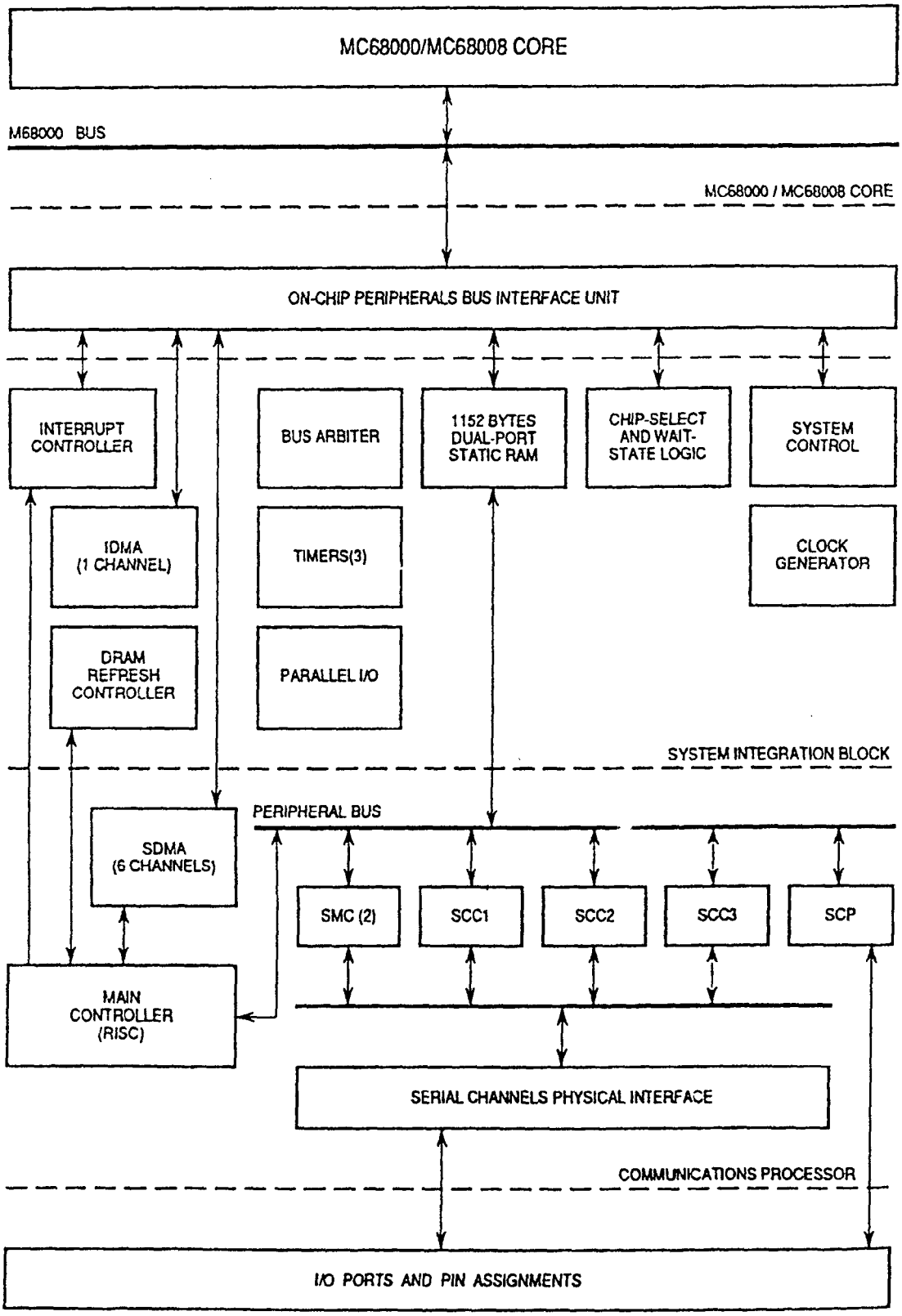


Figura 4.1 Diagrama de bloques del Microcontrolador MC68302

4.2.3 Arquitectura del sistema MC68302

La mayoría de los sistemas de propósito general basados en un microprocesador utilizan una arquitectura con un sólo bus al que están conectados todos los dispositivos periféricos. Una arquitectura de este tipo se representa en la figura 4.2.

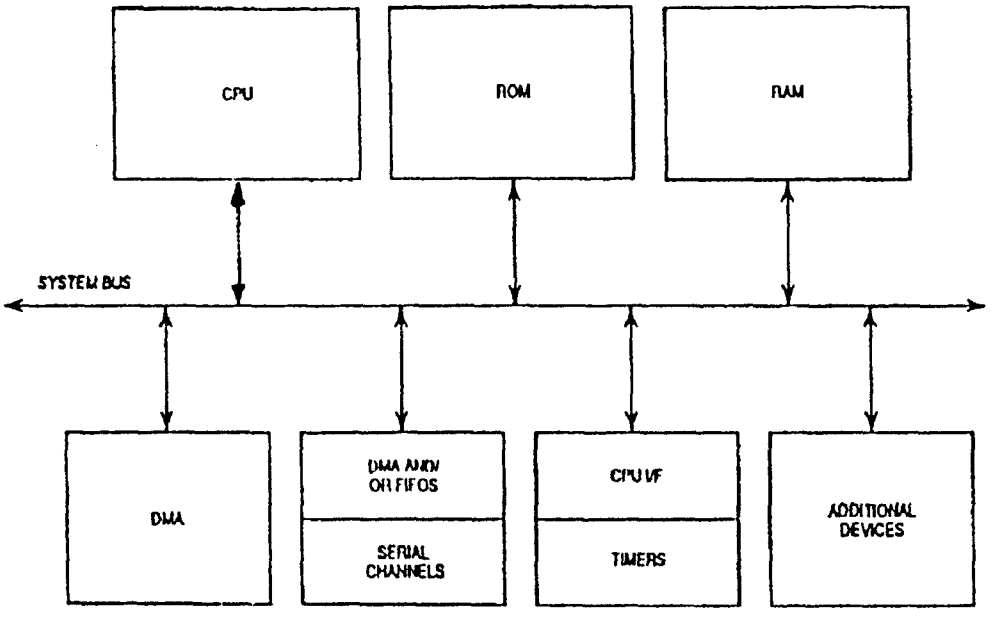


Figura 4.2 Diseño de un sistema microprocesador de propósito general

La arquitectura del sistema MC68302 se muestra en la figura 4.3. En esta arquitectura los dispositivos periféricos están aislados del bus del sistema a través de una memoria RAM de doble puerto.

El uso de un esquema de arbitraje único y transferencias asíncronas entre el microprocesador y la memoria RAM de doble puerto da al núcleo basado en el M68000, la apariencia de operación en estado de espera nulo.

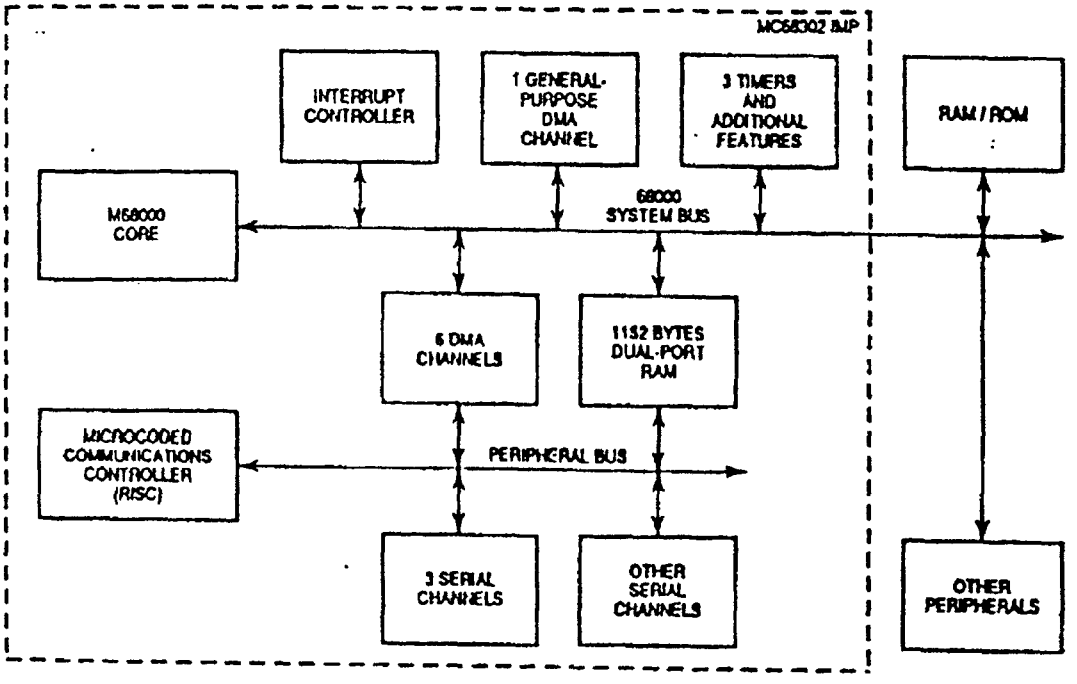


Figura 4.3 Diseño del sistema MC68302

4.2.4 Componentes principales del MC68302

• Microprocesador núcleo MC6800/MC68008 que soporta la familia M68000 de 8 ó 16 bits.

• Bloque de Integración del Sistema (SIB) que incluye

Controlador de Acceso Directo a Memoria Independiente (IDMA)

Controlador de Interrupciones con dos modos de operación

Puertos paralelos de entrada/salida; algunos con capacidad de interrupción

1152 bytes de Memoria de Acceso Aleatorio de Doble Puerto integrados en el chip

Tres relojes, incluido un reloj guardián o "watchdog timer"

Cuatro líneas programables Chip-Select con lógica de estado en espera

Mapeado de direcciones de la RAM de doble puerto y de los registros del IMP programables

Generador de reloj interno con señal de salida de reloj

Control del sistema:

Lógica de arbitraje del bus que soporta permanencia de interrupciones baja

Hardware guardián para monitoreo de la actividad del bus

Modos de bajo consumo (Standby)

Capacidad de deshabilitar la lógica de la CPU (M68000)

Controlador de refresco de la DRAM (RAM de doble puerto)

- Procesador de Comunicaciones que incluye:

Controlador Principal (Procesador RISC)

Tres Controladores de Comunicaciones Serie Independientes que soportan varios protocolos:

Control de Enlace de Datos Síncrono / Alto Nivel (HDLC / SDLC)

Transmisor Receptor Asíncrono Universal (UART)

Comunicación Síncrona Binaria (BISYNC)

Protocolo de Mensajes de Comunicaciones de Datos Síncrono / Asíncrono (DDCMP)

Modos Transparentes

Norma V.110

Seis canales DMA para los tres SCC

Interface físico flexible accesible a los SCC para Enlace de Datos Interchip (IDL), Interface Cicuito General (GCI), Modulación por Código de Pulsos (PCM), y Operación Interface Serie No multiplexado (NMSI)

Puerto de Comunicaciones Serie (SCP) para Comunicación Síncrona

Controladores de Manejo Serie (SMC) para los canales IDL y GCI

4.2.4.1 Bloque de Integración del Sistema (SIB)

El MC68302 posee un SIB que simplifica el diseño hardware y software. El controlador IDMA elimina la necesidad de un controlador DMA externo. Además hay un controlador de interrupciones que puede usarse en modo dedicado para generar

señales de reconocimiento de interrupciones sin necesidad de una lógica externa.

4.2.4.2 Procesador de Comunicaciones

El Procesador de Comunicaciones incluye el controlador principal, seis canales DMA serie, los tres Controladores de Comunicación Serie (SCC), el Puerto de Comunicaciones Serie (SCP) y los dos Controladores de Manejo Serie SMC.

Los datos son transmitidos y recibidos usando los descriptores de buffer apropiados y el espacio de datos de buffer para un canal dado.

4.2.4.3 Controlador principal

El controlador principal es un procesador RISC que proporciona servicios a todos los canales serie. Este contolador principal transfiere los datos entre los canales serie y la RAM interna / externa, ejecuta los comandos del host (comandos software) y genera interrupciones para el controlador de interrupciones.

Los datos son transferidos desde el canal serie a la memoria RAM de doble puerto o a la memoria externa a través del bus de los periféricos. Si los datos son transferidos entre los canales SCC y una memoria externa, el controlador principal utilizará hasta seis canales DMA para la transferencia. El controlador principal controla la comparación de direcciones y la generación y comprobación del CRC.

La unidad de ejecución incluye la Unidad Aritmético Lógica o ALU, la cual realiza las operaciones aritméticas y lógicas en los registros.

4.2.4.4 Controladores de Comunicación Serie

El MC68302 tiene tres Controladores de Comunicación Serie (SCC) Independientes. Cada SCC puede configurarse para implementar protocolos diferentes; por ejemplo, para funcionar como un gateway o como un interfaz para el acceso básico RDSI. Para simplificar la programación, cada implementación utiliza las mismas estructuras de datos.

Soportan los cinco protocolos indicados: HDLC, BISYNC, DDCMP, V.110, UART y un modo totalmente transparente.

Los pines de señal de reloj (RCLK, TCLK) para cada SCC pueden ser programados para tomar la señal de una fuente externa o interna; igualmente existe varias velocidades disponibles mediante programación para cada canal SCC.

Cada canal SCC también soporta las señales de control del standard para modem: request to send ($\overline{RTS}$), clear to send ($\overline{CTS}$) y detección de portadora ($\overline{CD}$). Se puede suministrar otras señales a través de las patillas de entrada / salida paralelas.

4.2.4.4.1 Características de los SCC

- Generador de velocidad programable (ratio en baudios) conducido por un reloj interno o externo
- Suministra las señales de modem $\overline{RTS}$, $\overline{CTS}$ y $\overline{CD}$
- Operación full-duplex
- Modo eco automático
- Modo bucle local
- Velocidad (ratio en baudios) del generador disponible externamente

4.2.4.4.2 Características del modo HDLC de los SCC

- Buffers de datos flexibles, permitiéndose que una trama ocupe varios buffers (en nuestro caso dos buffers: uno para el cabecero y otro para el campo de información si lo hay).
- Interrupciones separadas para las tramas y para los buffers (transmisión y recepción)
- Cuatro registros de comparación de direcciones con máscara
- Mantenimiento de cinco contadores de errores de 16 bits
- Generación / detección de flag / trama abortada / estado Idle
- Inserción / borrado de ceros

- Codificación de datos NRZ / NRZI
- Generación / chequeo de CRC-CCITT de 16 ó 32 bits
- Detección de tramas que no contienen un número múltiplo de octetos
- Flags 0-15 programables entre sucesivas tramas
- Retransmisión automática en caso de colisión

4.2.4.5 Controlador HDLC

4.2.4.5.1 Procesado en transmisión de tramas del canal HDLC

El transmisor HDLC está diseñado para trabajar sin casi ninguna intervención del núcleo M68000. Cuando el núcleo M68000 activa uno de los transmisores, éste empezará a transmitir banderas (o flags) tal y como se haya programado en el registro de modo HDLC.

El controlador HDLC sondeará el primer descriptor de buffer (BD) de la tabla de descriptores de buffers del canal de transmisión. Cuando haya alguna trama que transmitir, el controlador la recuperará de la memoria y la empezará a transmitir (después de haber transmitido el número mínimo de banderas entre tramas especificados por el usuario). Cuando se haya alcanzado el último descriptor de buffer y se haya enviado el último buffer que forma la trama, se añadirá el CRC (Código Redundante Cíclico) y la bandera de cierre de la trama, si fueron seleccionados (para ser enviados).

A continuación de la transmisión de la bandera de cierre, el controlador HDLC actualizará los bits de status en el descriptor de buffer y borrará el bit "ready". Cuando se haya llegado al final del actual descriptor de buffer y el último bit no esté activo (trabajando en el modo multibuffer), se borrará el bit "ready" sólamente.

En cualquier modo se producirá una interrupción si el bit de interrupción en el descriptor de buffer está activo. El controlador HDLC tratará entonces el siguiente descriptor de buffer de la tabla. De esta forma, el usuario puede ser interrumpido después de cada buffer, después de que un buffer específico sea transmitido o

después de cada trama. Nosotros emplearemos esta última opción. El software de nivel físico indicará al software de nivel 2 (al proceso correspondiente) una señal después de haber transmitido cada trama (signal FI_DATOS_ENVIADOS). De esta manera el nivel 2 sabrá cuando puede liberar la memoria dinámica solicitada para la formación del cabecero de esa trama ya enviada, ver función TratarFIdatosEnviados).

Para reordenar la cola de transmisión antes de que el IMP haya completado la transmisión de todos los buffers, se utilizará el comando STOP TRANSMIT. Esta técnica puede ser útil para enviar datos expéditos antes que los buffers previamente lincados o en caso de una situación de error.

Cuando el controlador HDLC recibe el comando STOP TRANSMIT, abortará la transmisión de la actual trama y comenzará a transmitir banderas. Cuando el controlador recibe un comando RESTART TRANSMIT, reanudará la transmisión.

4.2.4.5.2 Procesado en recepción de tramas del canal HDLC

El receptor HDLC está igualmente diseñado para trabajar sin casi ninguna intervención del núcleo M68000. El receptor HDLC puede realizar reconocimiento de direcciones, cheqeo del CRC y chequeo de máxima longitud de trama.

El usuario dispondrá de la trama recibida para aplicar cualquier protocolo basado en el HDLC (en nuestro caso el LAP-D para el Acceso Básico de RDSI).

Cuando el núcleo M68000 activa uno de los receptores, ese receptor espera hasta que le llegue una bandera de inicio. Cuando el receptor detecta el primer byte de la trama, el controlador HDLC comparará el campo de dirección de la trama con las direcciones programadas por el usuario. El usuario dispone de cuatro registros de 16 bits para direcciones y una máscara para la comparación de las direcciones. En ese momento el controlador HDLC comparará el campo de dirección recibido con los valores definidos por el usuario utilizando la máscara de direcciones. El controlador puede también detectar tramas de dirección de difusión (todos unos) si alguno de los registros está seleccionado todo a unos.

Si en la comparación se produce coincidencia, el controlador HDLC tomará el nuevo descriptor de buffer y, si éste está vacío, empezará a transferir (o escribir) la trama entrante al buffer de datos asociado al descriptor. Cuando el buffer de datos se haya llenado, el controlador HDLC borrará el bit "empty" del descriptor de buffer y generará una interrupción (si el bit de interrupción del descriptor de buffer está activado).

Si la trama entrante excede la longitud del buffer de datos, el controlador HDLC tomará un nuevo descriptor de buffer de la tabla y, si está vacío, continuará transfiriendo el resto de la trama al buffer de datos asociado a ese descriptor.

Durante este proceso el controlador HDLC chequeará si alguna trama excede la máxima longitud determinada por el usuario. Cuando se reciba toda la trama, se comparará el campo CRC con el CRC recalculado y se escribirá en el buffer de datos.

La longitud de la trama escrita hasta el último descriptor de buffer en la tramas HDLC es la longitud de la trama completa. Esto permite a los protocolos HDLC que pierden tramas reconocer la situación de una trama de longitud sobrepasada.

Entonces el controlador activa el bit de último buffer, escribe los bits de estado de trama en el descriptor de buffer e inactiva el bit "empty". El controlador HDLC generá a continuación una interrupción enmascarable indicando que se acaba de recibir una trama y está disponible en la memoria. Ahora el controlador esperará una nueva trama. Por último indicamos que es posible que dos tramas compartan una misma bandera o flag.

4.3 MODULO KERNEL

4.3.1 Descripción

El módulo Kernel (núcleo) constituye un conjunto de funciones que representan un grupo de servicios necesarios para el funcionamiento del software que representa los niveles o capas: Física, Capa de Enlace de Datos, Capa de Red y Aplicación.

4.3.2 Objetivos del Kernel

Básicamente sus objetivos son proporcionar el control de procesos, de pipes, de temporizadores (o contadores de tiempo), de memoria dinámica y del reloj en tiempo real. También controla la inicialización del hardware.

A continuación se describe brevemente cada uno de los objetivos del Kernel.

4.3.2.1 Control de procesos

Desarrollamos el control de procesos basándonos en un sistema operativo multiproceso que nos permitirá mantener varios procesos simultáneamente. Los procesos pueden estar activos o dormidos (inactivos) esperando por algún suceso o señal (signal) que los despierte o active.

Cada proceso se caracteriza por su contador de programa, sus variables propias, sus variables compartidas con otro(-s) proceso(-s) y por su pila que ejercerá las funciones generales de una pila de un programa en un sistema monoproceso, es decir, guardar parámetros entre funciones, variables locales, direcciones de retorno, etc.

Los procesos son creados por el módulo Kernel a partir de los ficheros implementados que, serán linkados conjuntamente para formar un sólo programa global. El kernel comenzará la ejecución de un proceso a partir de la función principal de ese proceso.

4.3.2.2 Control de pipes

Los pipes o tuberías representan caminos o conductos de comunicación entre procesos. Utilizaremos un pipe como un camino de un sólo sentido basado en el funcionamiento de una estructura de tipo cola en la que los mensajes (o información en general) enviados desde un proceso genérico, A, a otro B, se guardan (encolan) de forma secuencial hasta que el proceso destino (B) pueda atenderlos. Análogamente, cada proceso tendrá un pipe de donde se leerán las señales (signals) enviadas por los otros procesos.

Entre dos procesos, en general, existirán dos pipes: uno para cada sentido de la comunicación. Un proceso puede comunicarse con uno o más procesos. Para cada pareja de procesos que requieran comunicación, existirá una pareja de pipes para el envío / recepción de mensajes.

4.3.2.3 Control de temporizadores o contadores de tiempo

A partir de la fecha y hora en tiempo real del sistema, podemos desarrollar un servicio de temporizadores que nos permite:

- Solicitar un temporizador
- Activarlo para empezar una nueva cuenta cada vez que sea necesario
- Consultarlo o bien esperar a que termine la cuenta o aparezca alguna señal (signal) interrumpiendo.
- Por último, podemos liberar el contador de tiempo cuando no lo necesitemos utilizar más

4.3.2.4 Control de memoria dinámica

Durante la ejecución de un proceso puede ser necesario solicitar cierta cantidad de memoria dinámica según los requerimientos del algoritmo. Por ejemplo, es necesario solicitar memoria dinámica cada vez que se quiera formar el cabecero de una trama del protocolo LAP-D que se va a enviar a través del nivel físico al extremo par. Cuando el nivel físico nos indique que ya se ha enviado dicha trama (con el signal FI_DATOS_ENVIADOS), liberaremos la memoria correspondiente si no se va a retransmitir dicha trama.

Los bloques de memoria siempre empizan en posiciones pares. Control del reloj en tiempo real

4.3.2.5 Control del reloj en tiempo real

Como acabamos de mencionar, el sistema posee un reloj en tiempo real que nos permite obtener la fecha y la hora. La precisión de este reloj para el proceso de los temporizadores es de un segundo, es decir, la unidad mínima de cuenta es el segundo.

Capítulo 5: IMPLEMENTACION

5.1 GENERAL

La característica principal de la implementación del protocolo LAP-D realizada en los cinco ficheros mencionados es que sigue un flujo secuencial.

La implementación de los distintos estados en los que se puede encontrar una entidad (o entidad-proceso) de Enlace de Datos se realiza, en general, mediante funciones. Cada función (o función-estado) representa un estado concreto. El código devuelto por una función-estado representa el estado concreto al que evoluciona el algoritmo, es decir, indicará cual es la siguiente función-estado que debe ejecutarse.

Como ya hemos comentado, a partir del código de los ficheros implementados, se crearon y ejecutaron una serie de procesos bajo un sistema operativo multiproceso. Estos ficheros o procesos creados implementan las entidades de Capa o Nivel 2 (Capa de Enlace de Datos) de los lados del Equipo Terminal y del Terminal de Red, las entidades de Gestión de Capa de los lados del Equipo Terminal y del Terminal de Red, y además, se creó un quinto fichero auxiliar que da origen a un proceso multiplexor o concentrador necesario entre la Capa Física y las entidades-procesos de Capa de Enlace de Datos en el lado del Terminal de Red.

A los procesos que representan la implementación de una entidad de capa de la Capa de Enlace de Datos o de la Capa 3 los llamaremos entidades-procesos para enfatizar que se trata de procesos que implementan una entidad de capa.

Los procesos creados se ayudan de un módulo software que llamamos kernel (núcleo). El módulo kernel constituye un conjunto de funciones que proporciona el

control de procesos, de pipes (caminos o estructuras de comunicación tipo cola entre procesos), de temporizadores (o contadores de tiempo), de memoria dinámica y del reloj en tiempo real.

Diremos que la capa física es un conjunto de funciones que la implementan y no especificamente un proceso distinto. Así la Capa de Enlace de Datos utilizará estas funciones para, entre otras cosas, activar los puertos físicos, enviar y recibir datos a través de dichos puertos físicos, etc. Un puerto físico representará el canal D del bucle de abonado del Acceso Básico RDSI.

Con estos comentarios queremos insistir en la vinculación entre la implementación de los ficheros y la posterior ejecución de los procesos. En los ficheros se emplearán las funciones o servicios ofrecidos por el módulo kernel y por la Capa Física para, a partir de ellos (utilizando sus servicios), crear y ejecutar los procesos asociados en el entorno del sistema operativo multiproceso.

Así, por ejemplo, durante la ejecución del algoritmo puede ser necesario solicitar cierta cantidad de memoria dinámica según los requerimientos del flujo (para la formación de los cabeceros de las tramas que es necesario enviar).

5.2 ORDEN DE CREACION DE LOS PROCESOS

5.2.1 General

Los procesos se crearán con la ayuda del servicio del módulo kernel "KCreaProceso" al que tenemos que indicarle, entre otras cosas, la función con la que comienza la ejecución del proceso y el número de parámetros que recibe dicha función.

El orden de creación de los procesos será el siguiente: en primer lugar se crearán los procesos del lado del Equipo Terminal y a continuación los procesos del lado del Terminal de Red.

A cada proceso creado, el kernel le asociará un identificador. De esta forma será posible que un determinado proceso pueda enviar señales a otro (si sabe cual es el

identificador que le corresponde).

A continuación se describe el orden de creación de los procesos del lado del Equipo Terminal y del lado del Terminal de Red y los parámetros de entrada a las funciones que se ejecutan al crear cada proceso (funciones de partida de los procesos).

5.2.2 Orden de creación de los procesos del lado del Equipo Terminal

En primer lugar se crea el proceso que implementa la entidad de Capa de Enlace de Datos.

En segundo lugar se crea el proceso que implementa la entidad de gestión de capa de Capa de Enlace de Datos.

Y en tercer lugar se crearía el proceso que implementa la entidad de Capa 3.

A continuación se indican las definiciones de las funciones a partir de las cuales se crean los procesos del lado del Equipo Terminal así como los parámetros de entrada a cada una de ellas:

n2_ETmain(pipeLecturaN2N3, pipeLecturaN2GED, idPuerto)

A esta función, que representa la entidad de Capa de Enlace de Datos, se le pasa como parámetros de entrada el pipe de lectura donde recibirá los mensajes enviados por la entidad de Capa 3 y el pipe de lectura donde recibirá los mensajes enviados por la entidad de gestión de Capa de Enlace de Datos. Además se le indica cual es el puerto físico (que representa el canal D) con el que se comunicará.

n2_GED_ETmain(pipeEscrituraGED, pipeLecturaGED, identificadorN2)

A esta función, que representa la entidad de gestión de capa de la Capa de Enlace de Datos, se le pasará como parámetros los pipes de comunicación con la entidad de Capa de Enlace de Datos, es decir, el pipe de donde leerá los mensajes provenientes de la entidad-proceso de Capa de Enlace de Datos y el pipe donde escribirá los mensajes destinados a esta entidad-proceso. Además se le indica el

identificador de la entidad-proceso de Capa de Enlace de Datos para que le pueda enviar las señales necesarias.

SimulacionN3main(identificadorN2, pipeEscrituraN3N2, pipeLecturaN3N2)

A esta función, que representaría a la entidad de Capa 3 del lado del Equipo Terminal, se le pasa como parámetros de entrada los pipes de comunicación con la entidad de Capa de Enlace de Datos, es decir, el pipe de donde leerá los mensajes enviados por la entidad-proceso de Capa de Enlace de Datos y el pipe donde escribirá los mensajes dirigidos a esta entidad-proceso. Además se le pasa el identificador de dicha entidad-proceso de Capa de Enlace de Datos.

Hemos mencionado aquí la función a partir de la cual se crearía el proceso de simulación de la entidad de Capa 3; creado para la comprobación del correcto funcionamiento de la implementación del protocolo LAP-D en el sistema empotrado. Esta habría que sustituirla, en su caso, por la función que implemente la entidad completa de Capa 3.

Podemos observar en la gráfica 5.1 las ideas aquí expuestas sobre el orden de creación de los procesos y los parámetros de entrada a las funciones a partir de las cuales se crean. También se indica el contenido del primer mensaje enviado desde la entidad de gestión y desde la entidad de Capa 3 a la entidad de Capa de Enlace de Datos del lado del Equipo Terminal.

5.2.3 Paso de identificador de proceso a un proceso creado anteriormente

La creación de los procesos es secuencial: primero uno, luego otro, luego el siguiente, etc. De tal forma que si el primer proceso creado necesita comunicarse (enviándole alguna señal) a un proceso creado con posterioridad, necesitará que dicho proceso le indique de alguna forma cual es el identificador que le corresponde.

Así el proceso-entidad de capa de Enlace de Datos necesitará saber cuales son los identificadores de los procesos-entidades de Capa 3 y de gestión de capa de Enlace de Datos (que serán creados con posterioridad).

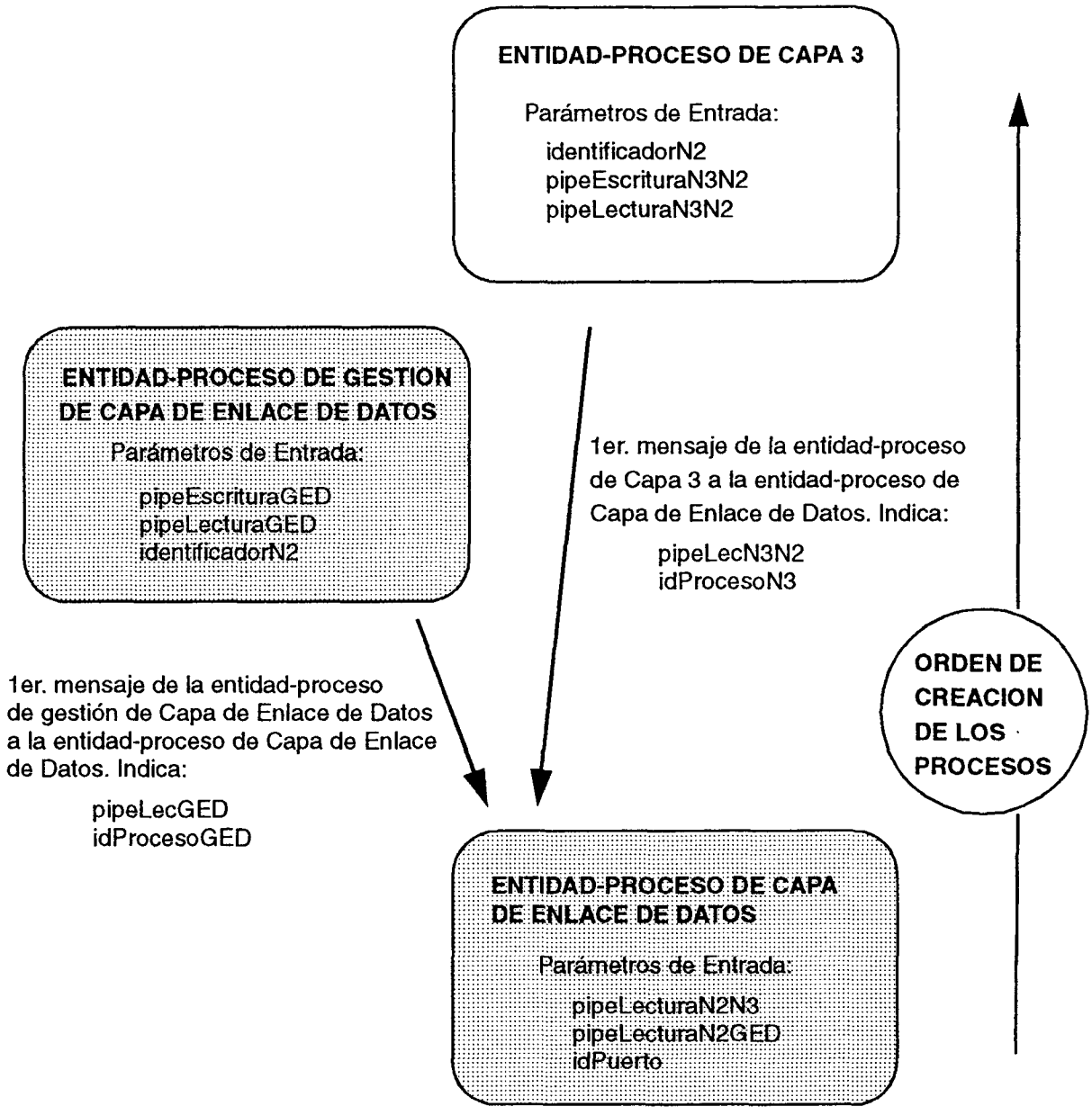


Figura 5.1 Orden de creación de los procesos y parámetros de entrada a las funciones de partida (lado del Equipo Terminal)

Este problema se resuelve de la siguiente manera: inmediatamente después de crear el proceso que representa la entidad de gestión de capa de Enlace de Datos, éste enviará un primer mensaje a la entidad-proceso de Capa de Enlace de Datos indicándole cual es su identificador asociado y el pipe de donde la entidad-proceso de gestión leerá los mensajes enviados por la entidad-proceso de Capa de Enlace de

Datos.

De forma análoga, al crearse la entidad-proceso de Capa 3, ésta enviará un primer mensaje a la entidad-proceso de Capa de Enlace de Datos indicándole cual es su identificador asociado y el pipe de donde la entidad-proceso de Capa 3 leerá los mensajes enviados por la entidad-proceso de Capa de Enlace de Datos.

5.2.4 Orden de creación de los procesos del lado del Terminal de Red

En primer lugar se crea el proceso multiplexor o concentrador necesario entre la capa física y las entidades de Capa de Enlace de Datos.

En segundo lugar se crea los "NUM_PROCESOS_N2_TR" (12) procesos que representan las entidades de Capa de Enlace de Datos del lado del Terminal de Red.

En tercer lugar se crearía el proceso multiplexor o concentrador previo a la entidad de Capa 3.

Y en cuarto lugar el proceso que representa la entidad de Capa 3.

A continuación se indican las definiciones de las funciones a partir de las cuales se crean los procesos del lado del Terminal de Red así como los parámetros de entrada a cada una de ellas:

n2_MuxTR(pipeLecturaMux, idPuerto)

Esta función representa el proceso multiplexor o concentrador entre la Capa Física y las entidades-procesos de Capa de Enlace de Datos del lado del Terminal de Red. A esta función se le pasa como parámetros de entrada el pipe de donde leerá los mensajes enviados por las entidades de Capa de Enlace de Datos y el identificador del puerto físico con el que se comunicará.

N2_TRmain(pipeLecturaN2N3, pipeLecturaN2GED, pipeLecturaN2mux, pipeEscrituraN2mux, identificadorN2master, identificadorMux)

A esta función, que representa la entidad de Capa de Enlace de Datos del lado del Terminal de Red, se le pasa como parámetros de entrada el pipe de lectura donde

recibirá los mensajes enviados por el proceso multiplexor de Capa 3 y el pipe de lectura donde recibirá los mensajes enviados por la entidad de gestión de Capa de Enlace de Datos, los pipes de comunicación con el proceso multiplexor: el pipe de lectura de donde leerá los mensajes enviados desde el proceso multiplexor y el pipe donde escribirá los mensajes dirigidos al proceso multiplexor. Además se indicará con el parámetro "identificadorN2master" cual de las entidades-procesos de Capa de Enlace de Datos funcionará como "master" (única entidad-proceso de Capa de Enlace de Datos del lado del Terminal de Red que se comunicará con la entidad de gestión de Capa de Enlace de Datos) y también se le indicará cual es el identificador del proceso multiplexor de nivel 2.

Por último haremos referencia a la función que representa el proceso multiplexor o concentrador de Capa 3 del lado del Terminal de Red (necesario entre las entidades-procesos de Capa de Enlace de Datos y la entidad de Capa 3) aunque su implementación no corresponda a este Trabajo Fin de Carrera. A esta función se le pasaría, entre otros parámetros, el pipe de lectura donde recibiría los mensajes enviados por las entidades-procesos de Capa de Enlace de Datos. Por otra parte este proceso multiplexor también debe recibir como parámetros de entrada los identificadores de las entidades-procesos de Capa de Enlace de Datos, así como los pipes de escritura y lectura del multiplexor con cada una de estas entidades-procesos.

De forma análoga a la figura 5.1 en la figura 5.2 podemos observar las ideas aquí expuestas sobre el orden de creación de los procesos y los parámetros de entrada a las funciones a partir de las cuales se comunican.

También se indica el contenido del primer mensaje enviado desde la entidad de gestión a la entidad de Capa de Enlace de Datos master y el contenido del primer mensaje enviado desde el proceso multiplexor de Capa 3 a cada una de las entidades de Capa de Enlace de Datos del lado del Terminal de Red.

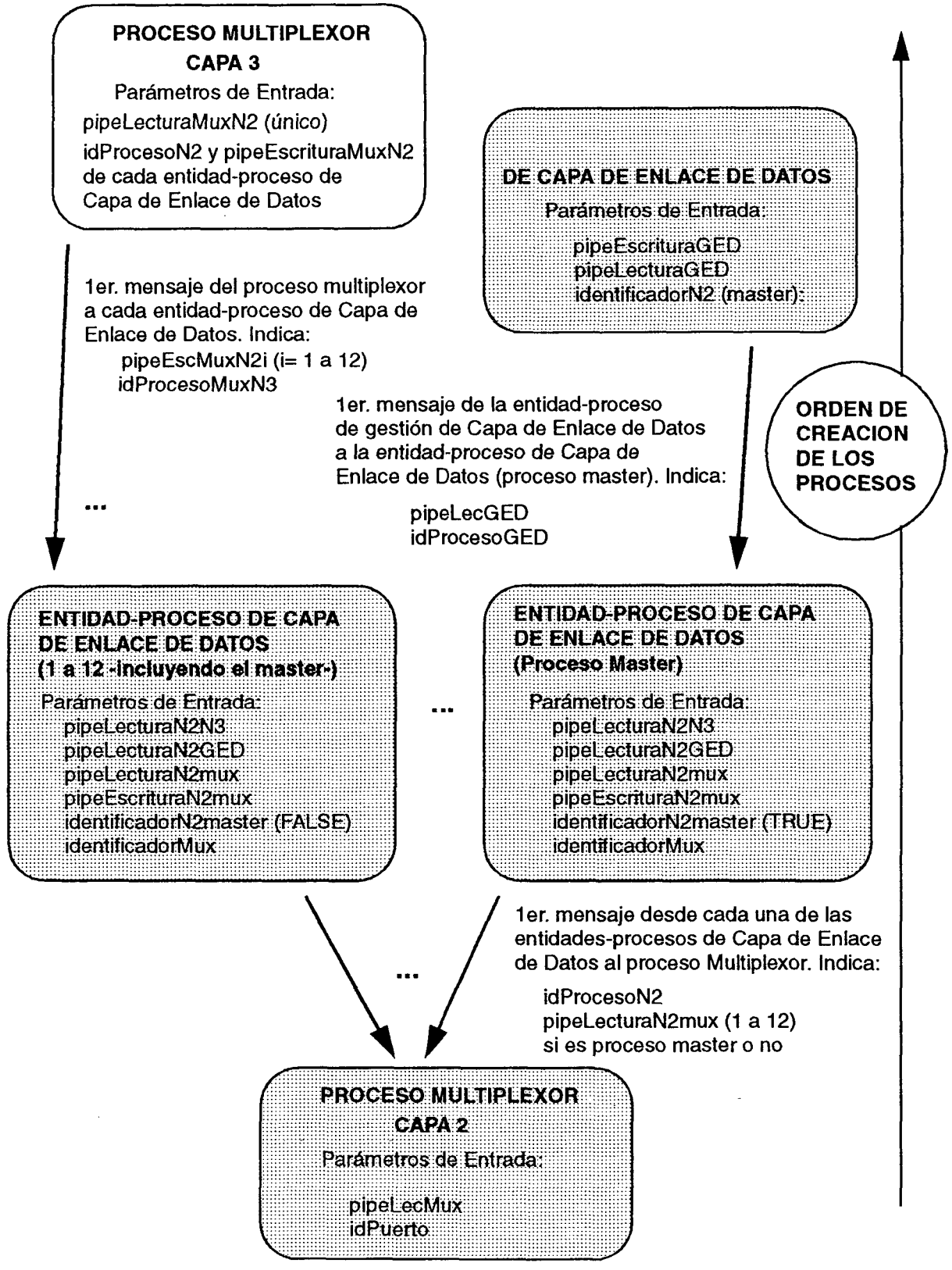


Figura 5.2 Orden de creación de los procesos y parámetros de entrada a las funciones de partida (lado del Terminal de Red)

5.3 COMUNICACION ENTRE PROCESOS

La comunicación entre procesos se realiza mediante señales y mediante pipes o caminos para el envío de información.

5.3.1 Señales

Las señales (signals) son indicaciones o avisos que se envían entre procesos.

Un proceso fuente enviará una señal a un proceso destino para indicarle algún suceso (por ejemplo, petición de liberación de memoria) o para que el proceso destino realice alguna acción determinada (por ejemplo leer del pipe la información o mensaje asociado a una determinada señal).

Es esta implementación el intercambio de primitivas entre entidades de capa (entre la entidad de Capa 3 y la entidad de Capa de Enlace de Datos y, entre la entidad de Capa de Enlace de Datos y la entidad de gestión de capa de Capa de Enlace de Datos) se implementa mediante el intercambio de señales.

Además el intercambio de señales es el medio de mantener la comunicación entre procesos. Por ejemplo, la forma de comunicarse el multiplexor del lado ("de capa 2") del Terminal de Red con las entidades-procesos de Capa de Enlace de Datos es mediante el intercambio de señales. En este caso no coincidirían con primitivas, puesto que los procesos multiplexores son procesos auxiliares al diseño implementado, pero los nombres de las señales en este caso tomarán la forma de unas supuestas primitivas.

Algunas de las señales implican realizar una lectura del pipe de comunicación entre esos dos procesos, de manera que, el proceso destino sepa que debe leer cierta información (en forma de mensaje) asociada a esas señales, del pipe donde el proceso fuente escribe.

5.3.2 Pipes de comunicación entre procesos

Los pipes o tuberías, como ya se ha comentado, representan caminos o conductos de comuncicaión entre procesos. Un pipe estará implementado mediante

una estructura de tipo cola en la que un proceso fuente escribirá información para que un proceso destino la lea cuando a éste le sea posible. Como es necesario que el proceso destino sepa que tiene información en espera en ese pipe, el proceso origen le enviará una señal para indicárselo: el proceso fuente escribirá primero la información en el pipe y luego enviará la señal que corresponda.

Los signals o señales, a su vez, se envían entre procesos utilizando también estructuras de tipo cola (pipes de señales). Las señales enviadas desde uno o más procesos a otro proceso destino, se guardarán en un pipe de señales (o estructura tipo cola de señales). El proceso destino las leerá, según su orden de llegada, en cuanto le sea posible según el flujo de ejecución del algoritmo.

5.3.2.1 Pipes utilizados

En el lado del Equipo Terminal utilizaremos dos pipes para la comunicación entre la entidad de Capa de Enlace de Datos y la entidad de gestión (uno para cada sentido de la comunicación), también utilizaremos dos pipes para la comunicación entre la entidad de Capa 3 y la entidad de Capa de Enlace de Datos (también uno para cada sentido de la comunicación).

De forma genérica, el pipe de escritura para un proceso es de lectura para otro y viceversa.

En el lado del Terminal de Red utilizaremos un pipe común en el que todas las entidades de Capa de Enlace de Datos escribirán los mensajes destinados al proceso multiplexor (proceso multiplexor intermedio entre las entidades de Capa de Enlace de Datos y la Capa Física). Este proceso multiplexor escribirá los mensajes destinados a cada una de las entidades de Capa de Enlace de Datos en un pipe individual (uno para cada entidad-proceso de Capa de Enlace de Datos).

El proceso-entidad de Capa de Enlace de Datos se comunicará con el proceso-entidad de gestión de Capa de Enlace de Datos del lado del Terminal de Red mediante dos pipes (uno para cada sentido de la comunicación).

Por último las entidades de Capa de Enlace de Datos escribirán los mensajes destinados al proceso multiplexor "de Capa 3" (proceso multiplexor intermedio entre las entidades de Capa de Enlace de Datos y la entidad de Capa 3) en un mismo pipe (pipe compartido); mientras este proceso multiplexor escribirá los mensajes destinados a cada una de estas entidades-procesos en un pipe individual.

Podemos observar estas ideas en las dos siguientes figuras (figuras 5.3 y 5.4) en las que se representa mediante flechas los pipes de comunicación entre los distintos procesos del lado del Equipo Terminal y del lado del Terminal de Red..

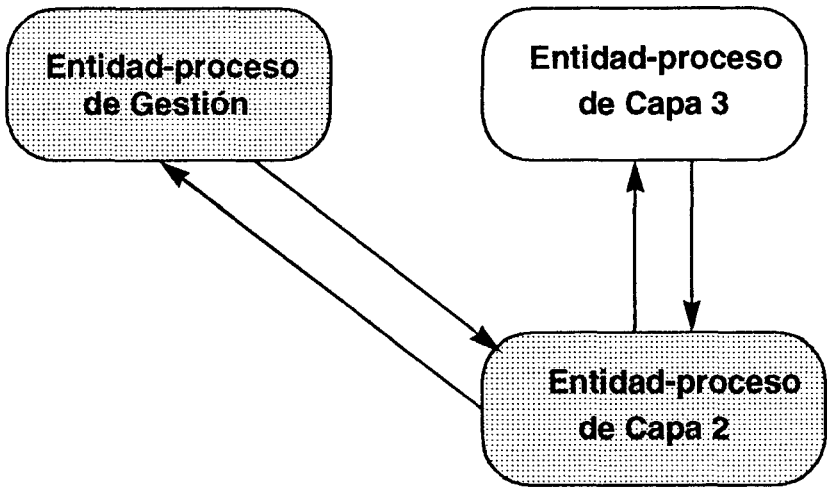


Figura 5.3 Pipes ente las entidades-procesos del lado del Equipo Terminal

5.4 CONSIDERACIONES COMUNES A TODOS LOS PROCESOS

Cada proceso se creará ejecutándose a partir de una función que constituye el cuerpo principal de ese proceso.

Todas estas funciones de partida comparten una serie de características comunes:

Después de ejecutar una serie de acciones para inicializar variables globales, manejo de primeros mensajes (si es el caso), pedir al kernel que indique cual es el identificador de proceso correspondiente al proceso en curso, y solicitar al kernel un termporizador (si es el caso), se entrará en un bucle infinito implementado con la

estructura "while(TRUE)". De esta forma cada uno de los procesos vivirá para siempre, es decir, que el cuerpo de este bucle se ejecutará de forma indefinida en el tiempo.

En la figura 5.5 podemos observar un organigrama que ilustra estas ideas.

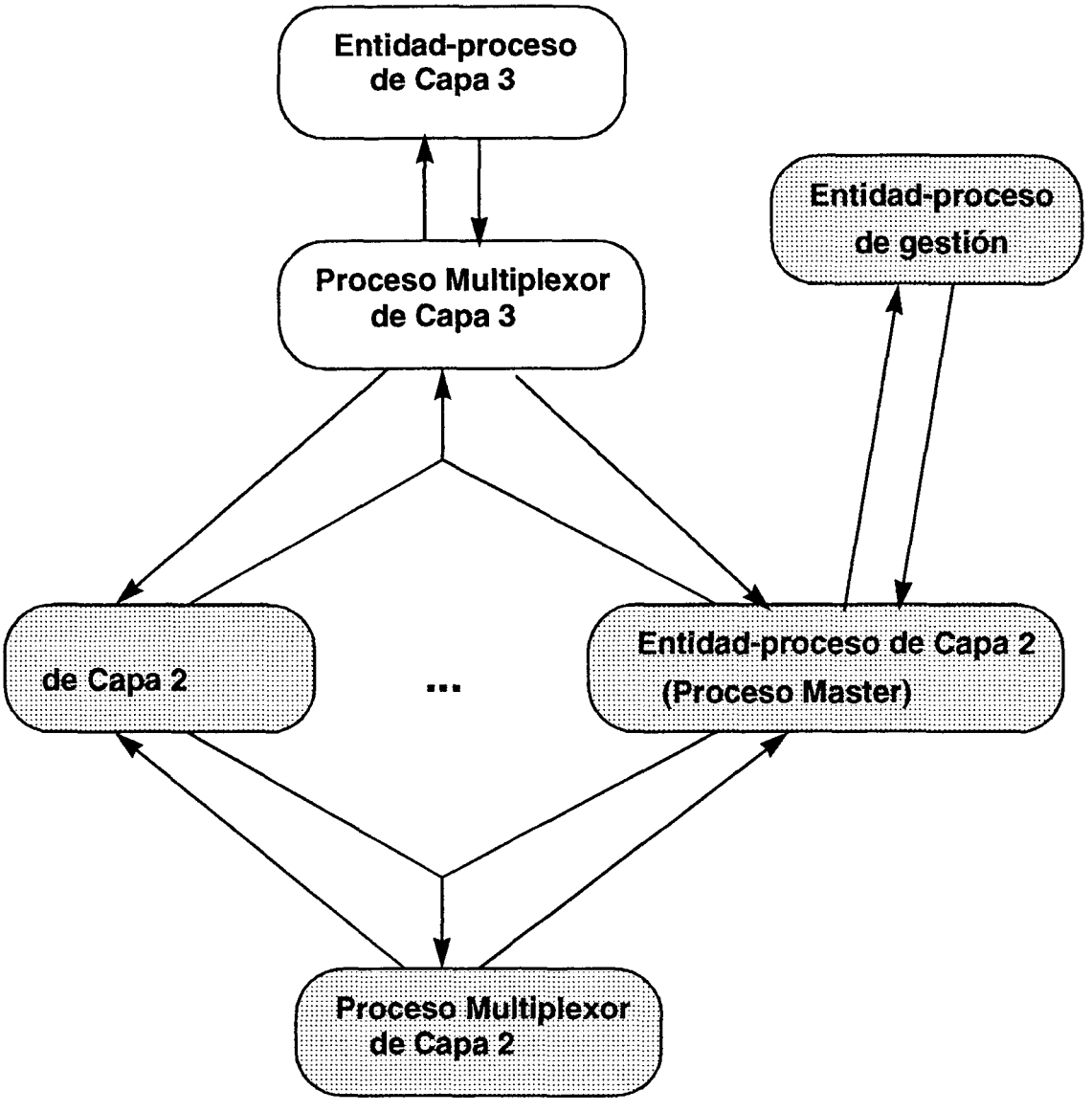


Figura 5.4 Pipes entre las entidades-procesos del lado del Terminal de Red

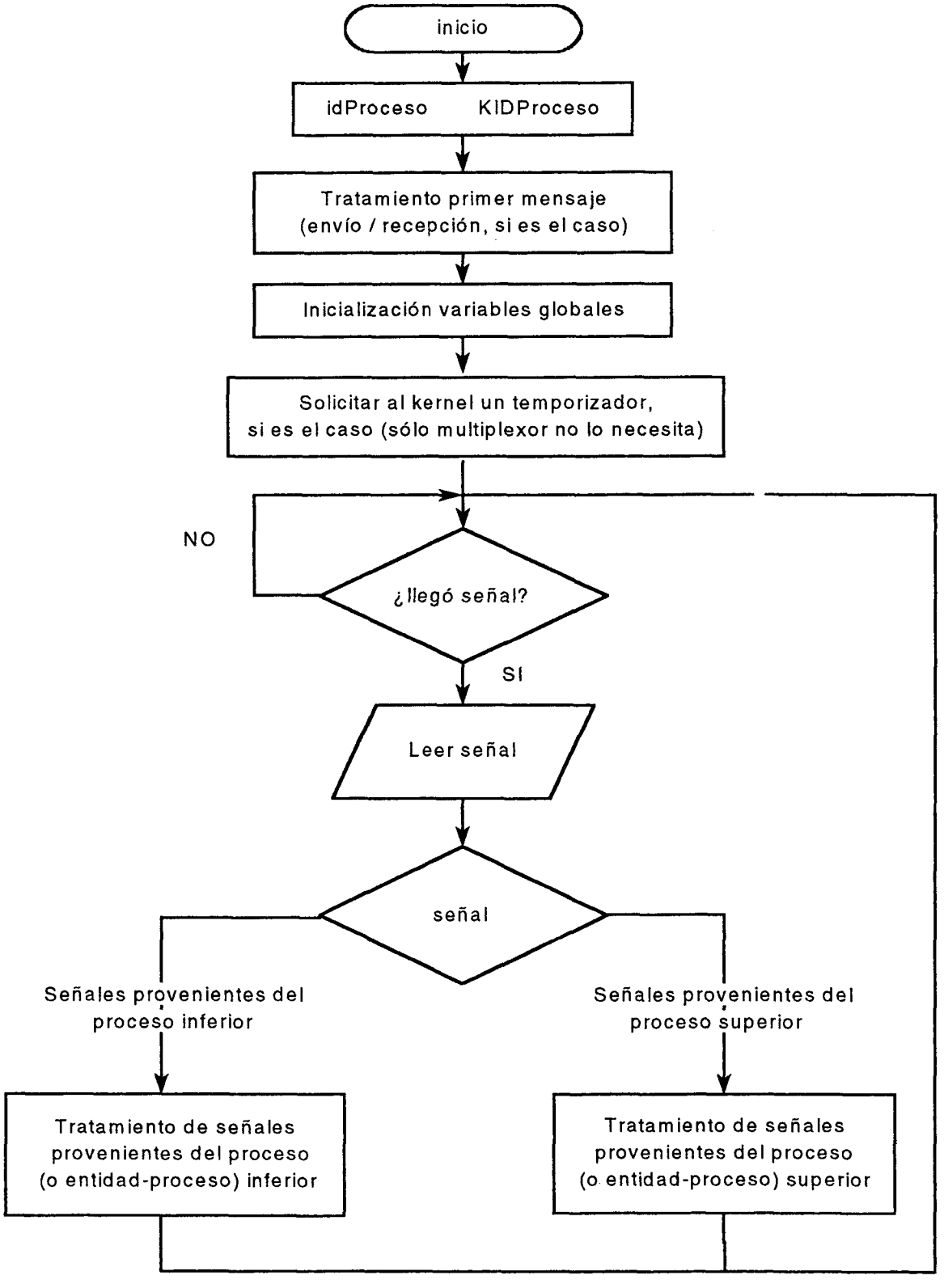


Figura 5.5 Organigrama general de un proceso genérico

5.5 CONSIDERACIONES COMUNES A LAS ENTIDADES DE CAPA DE ENLACE DE DATOS DE LOS LADOS DEL ET Y DEL TR

De forma general, para las dos entidades de capa de Enlace de Datos de los lados del Equipo Terminal y del Terminal de Red se implementa de igual forma los procedimientos de establecimiento y liberación del modo multitrama y los procedimientos para la transferencia de información en el modo multitrama. Se sigue las indicaciones de la normativa al respecto (ver apartados 3.13 y 3.14 del capítulo 3).

En la figura 5.7 podemos observar un organigrama general de la ejecución de una entidad-proceso de Capa de Enlace de Datos. La diferencia entre la entidad de Capa de Enlace de Datos del lado del Equipo Terminal y la del lado del Terminal de Red es que la primera precisa de un estado más que llamamos "IETnoAsignado" necesario en el establecimiento de la conexión de Enlace de Datos.

El estado EnlaceEstablecido, que constituye la implementación del estado de funcionamiento multitrama, estará compuesto a su vez por cuatro subestados que vendrán definidos en función de dos parámetros:

1.- El estado de espera de confirmación de la trama I (ventana de tamaño uno)

2.- El estado de la entidad remota: Receptor Preparado o No Preparado para recibir más tramas I

Así tenemos los siguientes cuatro estados:

NoWaitingAck_RR, NoWaitingAck_RNR, WaitingAck_RR y WaitingAck_RNR

En la figura 5.6 se muestra un diagrama de estados en el que podemos observar las transiciones entre los cuatro estados internos al estado EnlaceEstablecido. Esta será, de forma general, la implementación de los procedimientos para la transferencia de información en el modo multitrama.

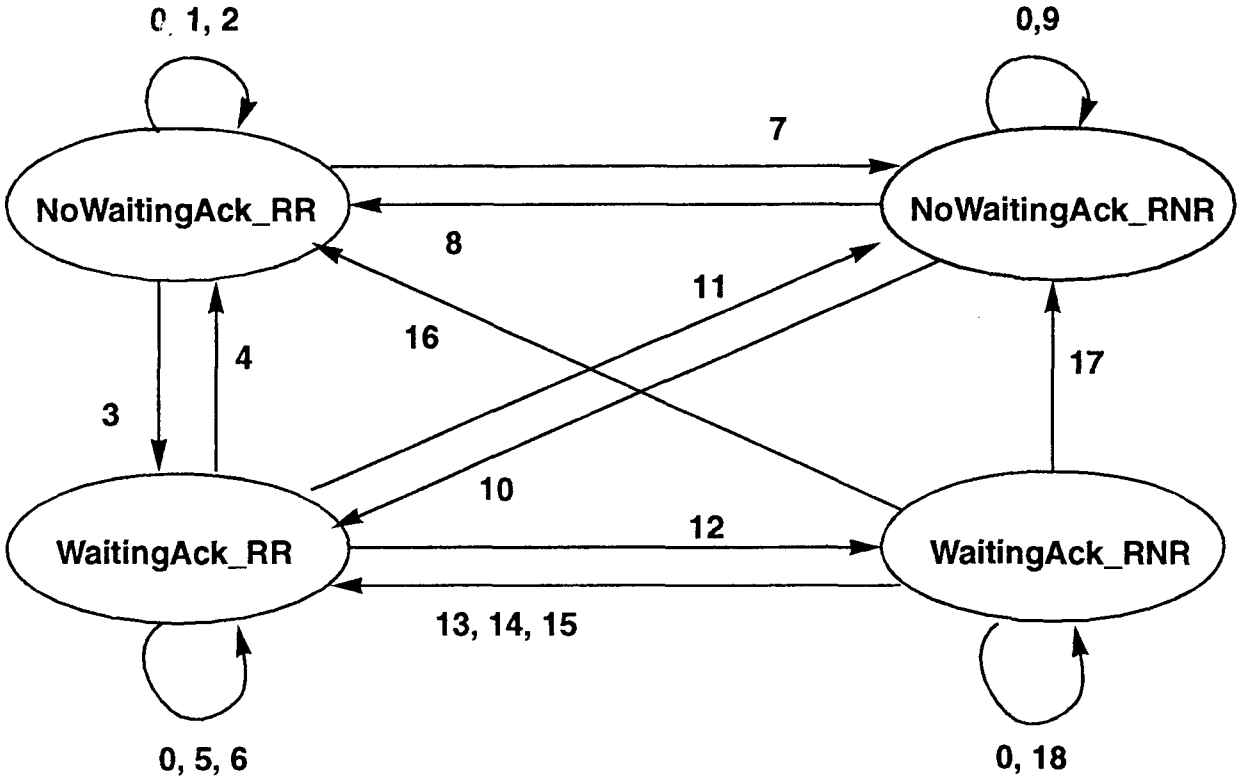


Figura 5.6 Transiciones entre estados internos al estado EnlaccEstablecido

A continuación se describe brevemente las transiciones entre estados indicando las entradas (en general, recepción de tramas o señales) y las salidas (envío de tramas, señales y cambios de estado).

0: En cualquier estado, sí llega un signal ED_UNIDAD_DATOS_REQ / Se tx. una trama UI

1: Sí llega una trama I / Envío una trama RR

2: Sí expira el temporizador T203 sg. / Envío trama RR

3: Sí llega un signal ED_DATOS_REQ / Envío trama I

4: Llega un RR confirmando y no hay datos en espera

5: REJ / I y arranco el temporizador T200 sg

6: RR confirmando y hay datos en espera / Envío trama I

7: Se recibe una trama RNR

8: RR y no hay datos en espera

9: RNR

10: RR y hay datos en espera / Envío trama I

11: RNR confirmando

12: RNR no confirmando

13: RR confirmando y hay datos en espera / Envío trama I

14: RR sin confirmar

15: REJ / Envío trama I (ns= nr de la trama REJ)

16: RR confirmando y no hay datos en espera

17: RNR confirmando

18: RNR no confirmando

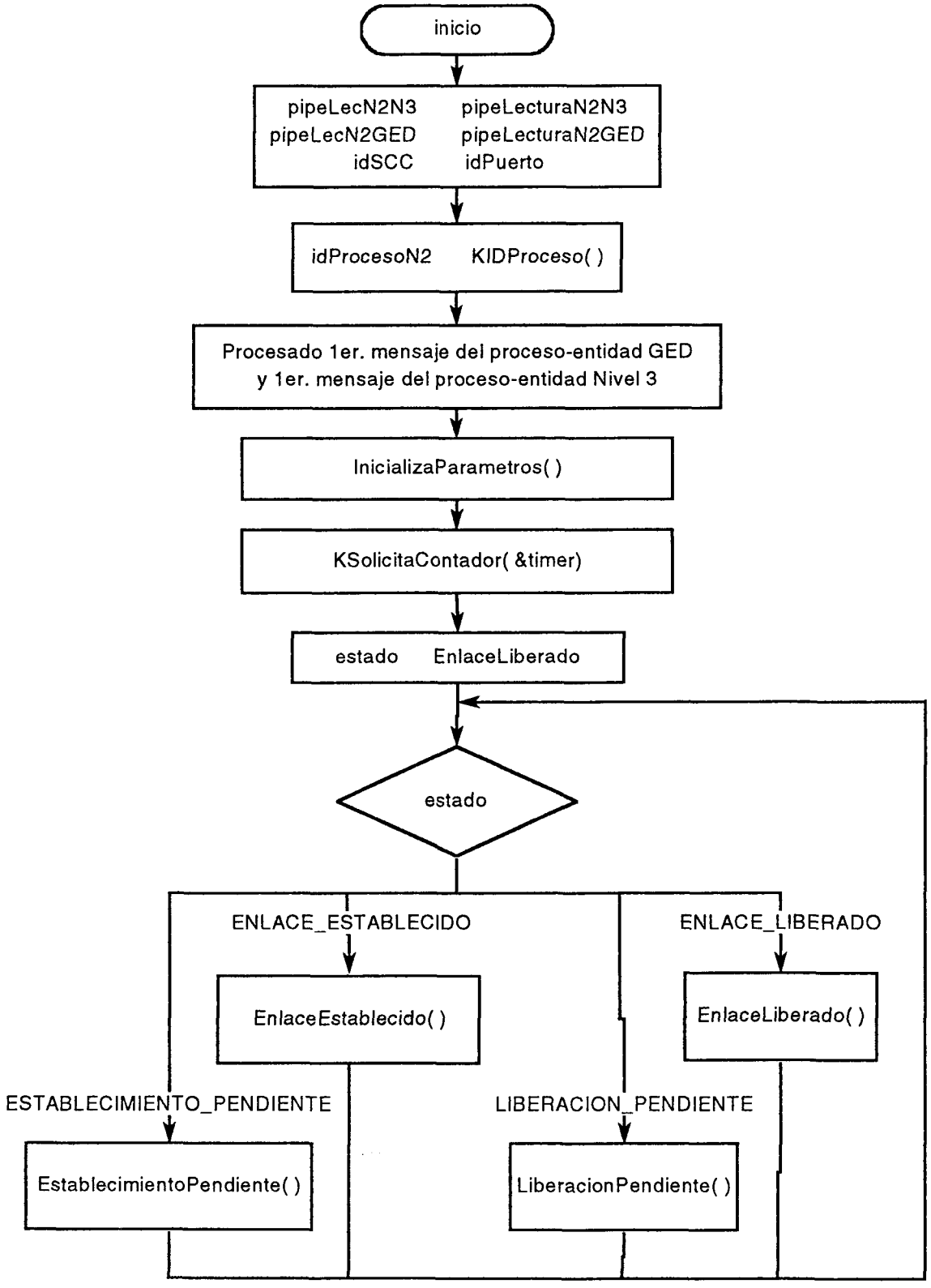


Figura 5.7 Organigrama general entidad-proceso de Capa de Enlace de Datos

5.6 CONSIDERACIONES COMUNES A LAS ENTIDADES DE GESTION DE LOS LADOS DEL ET Y DEL TR

La gestión de IET (Identificador de Equipo Terminal) está basada en los siguientes procedimientos:

- **Procedimientos de asignación de IET**
- **Procedimientos de prueba de IET**
- **Procedimientos de supresión de IET**
- **Procedimientos de verificación de identidad de IET iniciados por el equipo del usuario**

Un equipo de usuario que se encuentre en el estado "IETnoAsignado" deberá emplear los procedimientos de asignación de IET para pasar al estado en el que tiene un IET asignado (que en nuestra implementación llamamos "EnlaceLiberado").

Los procesos-entidades de capa de Enlace de Datos del lado del Equipo Terminal y del lado del Terminal de Red implementan estos procedimientos.

Los mensajes utilizados en los procedimientos de asignación de IET se tratarán de la siguiente forma:

- **Las entidades de gestión de Capa de Enlace de Datos del lado del Terminal de Red transmiten o reciben mensajes de entidad de gestión utilizando las señales GED_UNIDAD_DATOS_REQ y GED_UNIDAD_DATOS_IND respectivamente.**
- **La entidad de Capa de Enlace de Datos receptora de uno de estos mensajes de entidad de gestión los transmitirá en tramas de instrucción UI (añadiéndole el correspondiente cabecero formado por el campo de dirección en el que el valor del IPAS será el IPAS_GED = 63 y el valor del IET será el valor IET_DE_GRUPO = 127).**

5.7 LADO DEL EQUIPO TERMINAL

5.7.1 Entidad de Capa de Enlace de Datos del lado del ET

A continuación podemos observar el organigrama general de la ejecución del proceso de Entidad de Capa de Enlace de Datos del lado del Equipo Terminal donde

podemos observar el nuvo estado "IETnoAsignado" necesario para el establecimiento de la conexión de Enlace de Datos.

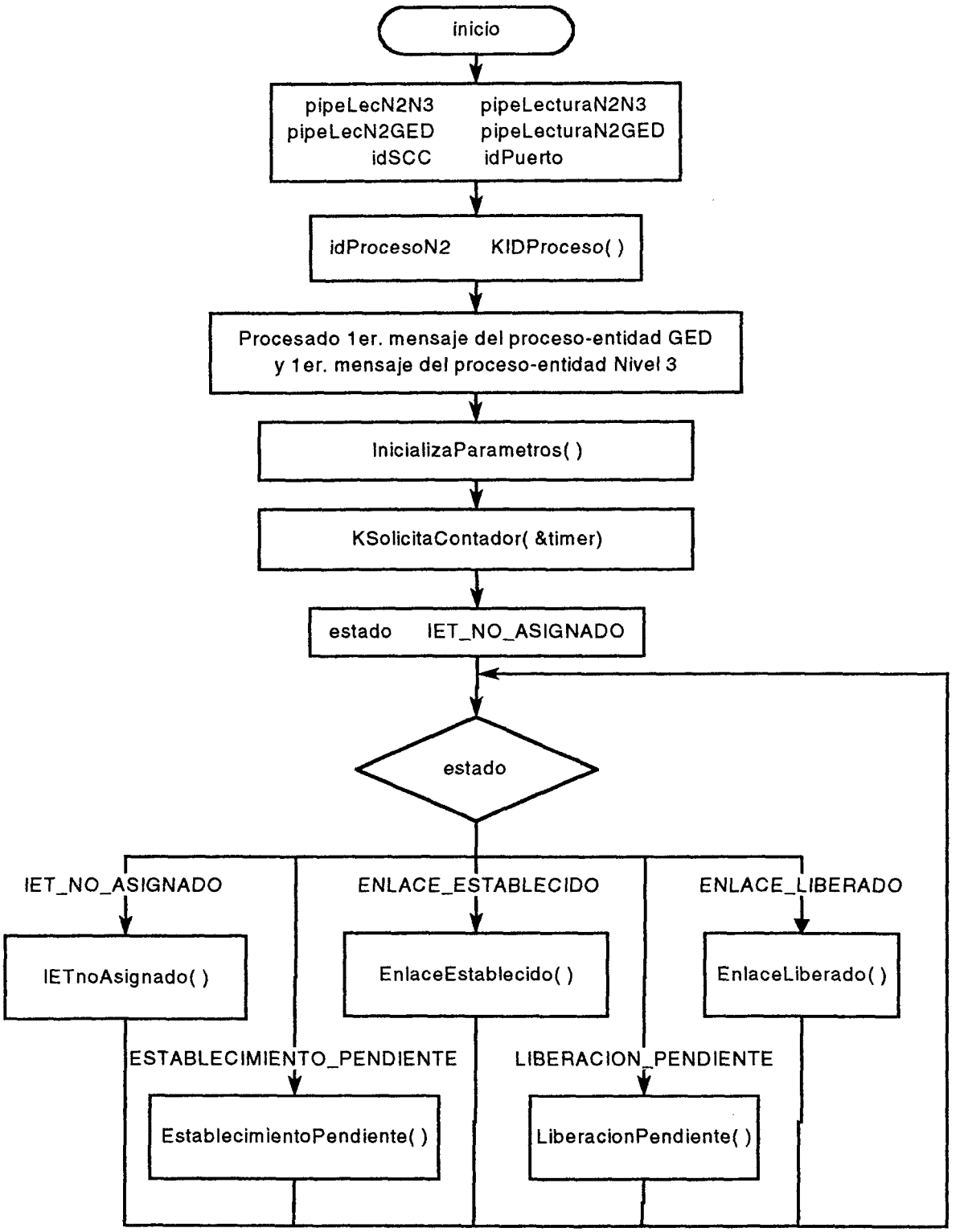


Figura 5.8 Organigrama general entidad-proceso de Capa de Enlace de Datos del lado del ET

5.7.1.1 Descripción de funciones

En este y los siguientes apartados de descripción o definición de funciones realizaremos una descripción de cada una de las funciones de los ficheros implementados.

Para cada una de las funciones se indicará los siguientes campos: su nombre, su definición, los parámetros de entrada y de salida (o entrada / salida), el código devuelto indicando que significa en cada caso, una breve descripción de las tareas de la función y por último, el campo observaciones en el que se comentarán, entre otras cosas, aquellas variables globales que son modificadas en el flujo de la función y en qué casos son modificadas.

Cuando una función se repita en distintos ficheros, por ejemplo, los ficheros que implementan las entidades de capa de Enlace de Datos de los lados del Equipo Terminal y del Terminal de Red utilizan casi las mismas funciones, se indicará haciendo referencia a la primera vez que se haya descrito dicha función.

NOMBRE: N2_ETmain

definición:

void N2_ETmain(pipeLecturaN2N3, pipeLecturaN2GED, idPuerto)

parámetros de entrada:

UWORD pipeLecturaN2N3 Pipe de comunicación del nivel 3 al nivel 2. El proceso que representa a la entidad de nivel 2 lee de este pipe lo que el proceso de nivel 3 le escribe

UWORD pipeLecturaN2GED Pipe de comunicación del GED al nivel 2. El proceso que representa a la entidad de nivel 2 lee de este pipe lo que el proceso GED le escribe

UWORD idPuerto Identificador puerto físico que controla (con el que se comunica)

parámetros de salida:

código devuelto:

descripción:

Esta función es la llamada al código que representa la entidad de capa de enlace de datos del lado del Equipo Terminal. Después de procesar el primer mensaje procedente de las entidades de nivel 3 y de la entidad de gestión de capa del lado del Equipo Terminal. Las tareas que realiza son:

- Solicitar al Kernel que indique cúal es el identificador del proceso que se ha creado a partir de la ejecución de esta función
- Procesado del primer mensaje proveniente de la entidad-proceso de nivel 3 y del primer mensaje proveniente de la entidad de gestión de capa del lado del Equipo Teminal
- Ejecutar función que inicializa parámetros o variables globales
- Solicitar al Kernel el temporizador o contador de tiempo que utilizaremos durante el resto del proceso (durante el resto de código del proceso).
- Inicializar estado del punto extremo de conexión a IET_NO_ASIGNADO
- Entrar en el bucle infinito "while(TRUE)" de los estados a los que irá evolucionando el proceso durante su ejecución (estados del punto extremo de conexión)

observaciones:

NOMBRE: InicializaParametros

definición:

void InicializaParametros( )

parámetros de entrada:

parámetros de salida:

código devuelto:

descripción:

Su función es inicializar determinados parámetros (variables globales) que se utilizarán en la comunicación con el puerto físico, concretamente con el Controlador

de Comunicaciones Serie (SCC). También inicializa los índices de los buffers utilizados, la variable "numIPASasignado" a cero y la variable "esperandoRespuesta" a FALSE

observaciones:

Las variables globales modificadas por esta función son las arriba mencionadas

NOMBRE: GuardaBtx

definición:

UWORD GuardaBtx(tipo, header, info)

parámetros de entrada:

UWORD	tipo	Tipo de trama
UBYTE	*header	Cabecero de la trama
UBYTE	*info	Campo de información si existe

parámetros de salida:

código devuelto:

TRUE Si se ha podido insertar un elemento (tipo, header e info) en la cola bufferTx

FALSE Si la cola está llena y no se insertó el elemento

descripción:

Inserta un elemento en la cola de tramas transmitidas "bufferTx", si es posible

observaciones:

La cola "bufferTx" y las funciones auxiliares "RecuperaBtx" y "LiberaBtx" son necesarias para tener un control secuencial de las tramas enviadas al nivel físico que esperan recibir un signal FI_DATOS_ENVIADOS para liberar o no (según el tipo de

trama) la memoria dinámica utilizada para formar el cabecero. Incrementará adecuadamente el índice "finalBtx" si se logra introducir un elemento en la cola

NOMBRE: RecuperaBtx

definición:

UWORD RecuperaBtx( )

parámetros de entrada:

parámetros de salida:

código devuelto:

TRUE	Ha sido posible tratar el primer elemento de la cola bufferTx
FALSE	La cola está vacía

descripción:

Esta función se ejecutará cada vez que se reciba un signal FI_DATOS_ENVIADOS. Se liberará o no la memoria asociada a la trama correspondiente (primera trama en espera) que se encuentra en la estructura bufferTx según el tipo de trama. Para una descripción más concisa, ver comentario en el correspondiente listado del anexo

observaciones:

Incrementa apropiadamente la variable global índice "frenteBtx" si la cola no está vacía

NOMBRE: LiberaBtx

definición:

void LiberaBtx( )

parámetros de entrada:

parámetros de salida:

código devuelto:

descripción:

Realiza los mismos pasos que la función "RecuperaBtx" pero para cada trama que se encuentre en la cola "bufferTx" (para liberar la memoria dinámica correspondiente a los cabeceros de todas las tramas)

observaciones:

Incrementará adecuadamente el índice "frenteBtx" hasta que la cola "bufferTx" esté vacía

NOMBRE: GuardaBtramasI

definición:

UWORD GuardaBtramasI(info, longInfo)

parámetros de entrada:

UWORD *info Información de un mensaje proveniente del nivel 3 que deberá transmitirse en el campo de información de una trama I

UWORD longInfo Longitud de esa información recibida

parámetros de salida:

código devuelto:

FALSE Si la estructura "bufferTramasI" está llena

TRUE Si se pudo añadir el mensaje de información y su longitud

descripción:

Se ejecutará esta función por cada signal ED_DATOS_REQ recibido añadiendo a la cola "bufferTramasI" la información del mensaje proveniente del nivel 3

observaciones:

Incrementará adecuadamente el índice "finalBtramasI" si se logra insertar la información del mensaje y su longitud

NOMBRE: LiberaBtramasI

definición:

void LiberaBtramasI( )

parámetros de entrada:

parámetros de salida:

código devuelto:

descripción:

Se ejecutará esta función cuando sea necesario liberar el buffer de tramas I (es decir cuando se desactive el nivel físico o cuando se libere el enlace de datos). Se libera la información enviada en los mensajes asociados con los signals ED_DATOS_REQ pendientes de envío

observaciones:

Incrementa la variable global o índice "frenteBtramasI" liberando la información hasta que el buffer está vacío

NOMBRE: GuardaBtramasUI

definición:

UWORD GuardaBtramasUI(info, longInfo, numIPAS)

parámetros de entrada:

UBYTE *info Información del mensaje asociado con el signal ED_UNIDAD_DATOS_REQ recibido del nivel 3 o con el signal GED_UNIDAD_DATOS_REQ recibido del Gestor de Enlace de Datos(del lado del ET)

UWORD longInfo Longitud de esa información

UBYTE numIPAS Identifica de donde proviene la trama: 0 ó 63 (IPAS_CERO ó IPAS_GED) si proviene del nivel 3 o del Gestor de Enlace de Datos

parámetros de salida:

código devuelto:

FALSE Si la cola está llena

TRUE Si se pudo insertar la información, su longitud y el correspondiente "numIPAS"

descripción:

Se añade a la cola "bufferTramasUI" la información del mensaje correspondiente (información, su longitud y el numIPAS que me identifica de donde proviene la trama UI). Se ejecutará esta función por cada signal ED_UNIDAD_DATOS_REQ o GED_UNIDAD_DATOS_REQ recibido.

observaciones:

Se incrementa adecuadamente la variable global índice "finalBtramasUI"

NOMBRE: LiberaBtramasUI

definición:

void LiberaBtramasUI( )

parámetros de entrada:

parámetros de salida:

código devuelto:

descripción:

Libera la información enviada en los mensajes asociados con los signals ED_UNIDAD_DATOS_REQ o GED_UNIDAD_DATOS_REQ pendientes de envío (enviando el correspondiente mensaje y signal de indicación de liberación) a la entidad de nivel 3 o entidad GED según el caso (según el "numIPAS")

observaciones:

Será necesario liberar el buffer de tramas UI sólo si se desactiva el nivel físico

NOMBRE: GuardaBtramasCtrl

definición:

UWORD GuardaBtramasCtrl(header, lengHeader)

parámetros de entrada:

UBYTE *header Cabecero de la trama de control

UWORD lengHeader Longitud del cabecero de la trama de control

parámetros de salida:

código devuelto:

FALSE Si la cola está llena

TRUE Si se logró insertar el cabecero y su longitud de la correspondiente trama de control

descripción:

Guarda el cabecero y su longitud de una trama de control cuando no se pueda

enviar en un determinado momento (para poderla enviar posteriormente)

observaciones:

Incrementa adecuadamente la variable global índice finalBtramasCtrl

NOMBRE: LiberaBtramasCtrl

definición:

void LiberaBtramasCtrl( )

parámetros de entrada:

parámetros de salida:

código devuelto:

descripción:

Se libera la memoria asociada a las tramas del buffer de tramas de control sólo cuando se desactiva el nivel físico

observaciones:

Incrementa adecuadamente la variable global índice frenteBtramasCtrl hasta que se haya liberado todo el bufferTramasCtrl

NOMBRE: LiberaVentana

definición:

void LiberaVentana( )

parámetros de entrada:

parámetros de salida:

código devuelto:

descripción:

Libera la memoria dinámica correspondiente al cabecero y a la información de la trama I enviada en espera de confirmación que forma la ventana (que se encuentra en la estructura ventana: cabecero y campo de información)

observaciones:

Se ejecutará esta función cuando se confirma la trama I que forma la ventana, o cuando se reinicializa o se pierde el enlace de nivel 2 o la conexión física y existe una trama I por confirmar en la ventana (en la estructura ventana)

NOMBRE: FormarMensaje23

definición:

void FormarMensaje23(mensaje23, tipoMens, longDatos, datos)

parámetros de entrada:

UWORD tipoMens Tipo de mensaje (generalmente será una copia del valor del signal asociado que es necesario enviar para que la entidad receptora de este mensaje se de cuenta de que tiene un nuevo mensaje en espera de ser atendido -con ese fin se definió- ; en el caso que no sea una copia del signal, indicará el propio tipo de mensaje)

UBYTE *datos Información asociada con ese mensaje

UWORD longDatos Longitud de la información asociada con ese mensaje

parámetros de salida:

MENSAJE23 *mensaje23 Estructura (variable global) mensaje23 (mensaje a enviar o recibido; para esta función mensaje a enviar)

código devuelto:

descripción:

Formar un mensaje,es decir, actualiza la estructura mensaje con los campos apropiados que quieren transmitir; para posteriormente enviarla a la entidad de nivel 3 o al Gestor de Enlace de Datos

observaciones:

Se modifica la estructura (variable global) mensaje23

NOMBRE: FormarCabeceroTrama

definición:

UWORD FormarCabeceroTrama(header, headerLength, tipoTrama, bitIR, bitPF)

parámetros de entrada:

UBYTE bitIIR Valor del bit Instrucción / Respuesta para formar el campo de dirección de la trama (1 ó 0, ACTIVO o INACTIVO)

UBYTE bitPF Valor del bit Polling / Final para formar el campo de control de la trama

UWORD tipoTrama Tipo de trama que se quiere formar

parámetros de salida:

UBYTE **header Cabecero de la trama que se forma

UWORD *headerLength Longitud del cabecero de la trama que se forma

código devuelto:

TRUE Si se pudo formar la trama correctamente y obtuvimos la memoria dinámica solicitada

FALSE Si no se pudo pedir memoria dinámica para formar la trama

descripción:

Esta función forma el cabecero de la trama que se solicite.

observaciones:

Se utilizan las variables globales "numIPASasignado" y "numIETasignado" (como si fueran parámetros de entrada) para formar el campo de dirección de la trama solicitada.

En el caso de las tramas de transferencia de información y las tramas no numeradas no se tomará el valor del parámetro de entrada, sino el valor especificado en el código, pues éste es fijo. Sólo las tramas de supervisión utilizarán el valor del parámetro de entrada bitPF que es variable en su caso.

Siempre supondremos que existe suficiente memoria dinámica para formar una determinada trama; si ho la hubiese habría que modificar el hardware para aumentar la memoria RAM de la que los procesos solicitan la memoria dinámica

NOMBRE: IdentificaTrama

definición:

UWORD IdentificaTrama(frame, frameLength, bitioIR, bitioPF, numIETrecibido, numIPASrecibido)

parámetros de entrada:

UBYTE *frame	Trama recibida (cabecero y campo de información si lo hubiese)
UWORD frameLength	Longitud de la trama recibida

parámetros de salida:

UBYTE *bitioIR	Valor del bit I/R de la trama recibida
UBYTE *bitioPF	Valor del bit P/F de la trama recibida
UBYTE *numIETrecibido	Valor del Identificador de Equipo Terminal

recibido (IET) de la trama recibida

UBYTE *numIPASrecibido Valor del Identificador de Punto de Acceso al Servicio (IPAS) de la trama recibida

código devuelto:

Tipo de trama recibida (I, RR, RNR, REJ, SABME, DM, UI, DISC, UA) o DESCONOCIDO en el caso de que la trama recibida no concuerde con ninguna de las soportadas en esta implementación del protocolo LAP-D

descripción:

Identifica el tipo de trama recibida a partir de la trama recibida por el nivel físico y de su longitud, teniendo encuenta los distintos campos que caracterizan a las tramas (campo de dirección y campo de control) y los distintos valores que pueden tomar

observaciones:

En esta implementación del protocolo LAP-D no se reconocen las tramas XID y FRMR (opcional) que se reciban.

NOMBRE: EnviarTramaDelBufferUI

definición:

UWORD EnviarTramaDelBufferUI( )

parámetros de entrada:

parámetros de salida:

código devuelto:

TRUE Si se pudo enviar la primera trama del buffer de tramas UI

FALSE Si no se pudo enviar la trama UI en espera

descripción:

Se ejecutará esta función cada vez que se quiere enviar la primera trama en espera del buffer de tramas UI. Si se logra enviar, se guarda en el buffer de transmisión (bufferTx) y se incrementa la variable global frenteBtramasUI apropiadamente. Si no se puede enviar la función devuelve FALSE

observaciones:

Si se puede enviar la trama, se incrementa la variable global frenteBtramasUI

NOMBRE: EnviarTramaDelBufferCtrl

definición:

UWORD EnviarTramaDelBufferCtrl( )

parámetros de entrada:

parámetros de salida:

código devuelto:

TRUE Si se logra enviar la trama de control del buffer de tramas de control en espera

FALSE Si no se logra enviar la trama

descripción:

Se ejecuta cada vez que se quiere enviar la primera trama en espera del buffer de tramas de control (bufferTramasCtrl). Esta función es análoga a la función "EnviarTramaDelBufferUI"

observaciones:

Si se logra enviar la trama, se incrementará apropiadamente la variable global frenteBtramasCtrl

NOMBRE: EnviarTramaVentana

definición:

UWORD EnviarTramaVentana( )

parámetros de entrada:

parámetros de salida:

código devuelto:

TRUE	Si se logra enviar la trama
FALSE	Si no se logra enviar la trama

descripción:

Se ejecuta cada vez que se quiere enviar la trama I contenida en la ventana de transmisión. Si se logra enviar, se guarda en el buffer de transmisión

observaciones:

NOMBRE: TratarTramaUIrecibida

definición:

void TratarTramaUIrecibida(numIETrecibido, numIPASrecibido)

parámetros de entrada:

UBYTE numIETrecibido	Valor del IET de la trama UI recibida
UBYTE numIPASrecibido	Valor del IPAS de la trama UI recibida

parámetros de salida:

código devuelto:

descripción:

Esta función se ejecuta cada vez que se recibe una trama UI. Si se trata de una trama UI, se enviará a la entidad de capa 3. Si se trata de una trama UI del gestor (trama TIPO_GED_UI), se enviará a la entidad de gestión

observaciones:

NOMBRE: TratarEDdatosReq

definición:

UWORD TratarEDdatosReq( )

parámetros de entrada:

parámetros de salida:

código devuelto:

TRUE Si se pudo leer del pipe de comunicación con la capa 3 (pipe de escritura para la capa 3, pipe de lectura para la entidad de Capa de Enlace de Datos

FALSE Si no se pudo leer del pipe

descripción:

Guarda el mensaje de nivel 3 recibido para su posterior envío formando una trama I. Se ejecuta cuando estemos en uno de los estados que no permiten enviar una trama I: NoWaitingAck_RNR, WaitingAck_RR y WaitingAck_RNR

observaciones:

Siempre supondremos que se puede leer del pipe

NOMBRE: TratarEDunidadDatosReq

definición:

UWORD TratarEDunidadDatosReq( )

parámetros de entrada:

parámetros de salida:

código devuelto:

TRUE Si se pudo leer del pipe de comunicación con la capa 3 (pipe de escritura para la capa 3, pipe de lectura para la entidad de Capa de Enlace de Datos

FALSE Si no se pudo leer del pipe

descripción:

Esta función se ejecutará cada vez que se reciba una señal ED_UNIDAD_DATOS_REQ. Si el buffer de tramas UI está vacío y se puede transmitir la información recién llegada formando una trama UI, ésta se añadirá al buffer de transmisión.

Si no se pudo transmitir la trama o si el buffer de tramas UI no está vacío, se añadirá la información recién llegada al propio buffer de tramas UI con numIPAS = IPAS_CERO = 0

observaciones:

Es necesario contemplar el caso que no se haya transmitido ninguna trama y llegue una señal ED_UNIDAD_DATOS_REQ, por eso si el buffer de tramas UI está vacío, se intenta transmitir la información recién llegada. Si no se contemplase este caso, no llegaría una señal FI_DATOS_ENVIADOS que permitiese transmitir esa primera trama UI al comenzar el programa.

NOMBRE: TratarGEDunidadDatosReq

definición:

UWORD TratarGEDunidadDatosReq( )

parámetros de entrada:

parámetros de salida:

código devuelto:

TRUE Si se pudo leer del pipe de comunicación con la capa 3 (pipe de escritura para la capa 3, pipe de lectura para la entidad de Capa de Enlace de Datos

FALSE Si no se pudo leer del pipe

descripción:

Esta función es análoga a la función "TratarEDunidadDatosReq".

Se ejecuta cada vez que se recibe una señal GED_UNIDAD_DATOS_REQ. Si el buffer de tramas UI está vacío y se puede transmitir la información recién llegada formando una trama TIPO_GED_UI, se añade al buffer de transmisión.

Si no se puede transmitir o si el buffer de tramas UI no está vacío, se añade la información recién llegada al propio buffer de tramas UI teniendo encuenta que el número de IPAS destino será el IPAS_GED, es decir, el destino será el gestor de Enlace

observaciones:

NOMBRE: TratarFIdatosEnviados

definición:

UWORD TratarFIdatosEnviados( )

parámetros de entrada:

parámetros de salida:

código devuelto:

TRUE Siempre devuelve este valor indicando que todo fue correcto

descripción:

Se ejecutará esta función por cada señal FI_DATOS_ENVIADOS que se reciba.

Se ejecuta la función RecuperaBtx y después:

Si el buffer de tramas de control no está vacío, se enviará la primera trama de control en espera.

Si el buffer de tramas de control está vacío:

Si la variable "ventanaEsperandoTx" tiene valor "TRUE", se enviará la trama I de la ventana y se asignará a la variable global "ventanaEsperandoTx" en valor "FALSE"

Si la variable "ventanaEsperandoTx" tiene valor "FALSE", se enviará la primera trama del buffer de tramas UI en espera

observaciones:

NOMBRE: EnviarTramaCtrl

definición:

UWORD EnviarTramaCtrl(frameType, bitioIR, bitioPF)

parámetros de entrada:

UWORD frameType Tipo de trama de control que se quiere enviar

UBYTE bitioIR Valor del bit I / R para formar el cabecero de la trama que se quiere enviar

UBYTE bibioPF Valor del bit P / F para formar el cabecero de la trama que se quiere enviar

parámetros de salida:

código devuelto:

TRUE Siempre devuelve este valor indicando que todo fue correcto

descripción:

Si el buffer de tramas de control está vacío y se pudo transmitir, se guarda en el buffer de transmisión

Si no se pudo transmitir o si el buffer de tramas de control no está vacío, se guarda en el buffer de tramas de control para su posterior transmisión

observaciones:

NOMBRE: IETnoAsignado

definición:

UWORD IETnoAsignado( )

parámetros de entrada:

parámetros de salida:

código devuelto:

ESTABLECIMIENTO_PENDIENTE Si se evoluciona a ese estado

descripción:

Esta función es la implementación del estado de la entidad-proceso de Capa de Enlace de Datos en la que no tiene un IET asignado para establecer una conexión de Enlace de Datos

observaciones:

Se modifican las variables globales "vs", "va" y "vr" según el flujo de instrucciones. Otras variables globales que se pueden modificar dependiendo del flujo de la ejecución son las siguientes: "establecimientoPendiente", "numIETasignado"

NOMBRE: EnlaceLiberado

definición:

UWORD EnlaceLiberado( )

parámetros de entrada:

parámetros de salida:

código devuelto:

ESTABLECIMIENTO_PENDIENTE Cuando se evoluciona a este estado según el flujo del autómata

ENLACE_LIBERADO Si se vuelve al estado actual

descripción:

Esta función es la implementación del estado "IET asignado" de una conexión de Enlace de Datos (que nosotros hemos llamado "EnlaceLiberado")

observaciones:

Se modifican las variables globales "vs", "va" y "vr" según el flujo de instrucciones.

NOMBRE: EstablecimientoPendiente

definición:

UWORD EstablecimientoPendiente( )

parámetros de entrada:

parámetros de salida:

código devuelto:

ENLACE_LIBERADO Si se evoluciona al estado EnlaceLiberado

ENLACE_ESTABLECIDO Si se evoluciona al estado EnlaceEstablecido

descripción:

Esta función implementa el estado de la conexión de enlace de datos, en la que

está pendiente por establecerse.

observaciones:

Se modifican las variables globales "vs", "va", "vr" y n200 según el flujo de ejecución del proceso

NOMBRE: EnlaceEstablecido

definición:

UWORD EnlaceEstablecido( )

parámetros de entrada:

parámetros de salida:

código devuelto:

LIBERACION_PENDIENTE

ESTABLECIMIENTO_PENDIENTE

ENLACE_LIBERADO Cuando se devulve uno de estos valores, se pasa a la estructura de selección múltiple "switch(estado)" de la función n2_ETmain

descripción:

Existen cuatro subestados dentro del estado "EnlaceEstablecido" que vienen definidos según dos parámetros:

1.- El estado de espera de confirmación de la trama I (recordemos tamaño de la ventana es uno): esperando acuse de recibo o no

2.- El estado de la entidad remotra: Receptor Preparado (RR) o No Preparado para recibir más tramas I

observaciones:

Se inicializan las siguientes variables globales empleadas en los subestados inernos al estado EnlaceEstablecido: n200 y numTxTramaI. También se modifica la variable global estadoEnTx tomando el valor del subestado (o estado en su caso) al que se evolucionará

NOMBRE: NoWaitingAck_RR

definición:

UWORD NoWaitingAck_RR( )

parámetros de entrada:

parámetros de salida:

código devuelto:

NO_WAITING_ACK_RR	Se evolucionará denuevo al estado NoWaitingAck_RR
NO_WAITING_ACK_RNR	Se evolucionará al estado NoWaitingAck_RNR
WAITING_ACK_RR	Se evolucionará al estado WaitingAck_RR

descripción:

Esta función implementa el subestado del estado EnlaceEstablecido que corresponde con la siguiente situación: no se espera confirmación de trama I transmitida y receptor está en condición "Ready" o dispuesto a aceptar una trama I entrante

observaciones:

Se modifican las variables globales "ns", "nr", "vr", "n200", "ns", "vs", "va", num "TxTramasI", "esperandoRespuesta", "ventanaEsperandoTx" y la estructura "ventana" según el flujo de ejecución del proceso

NOMBRE: NoWaitingAck_RNR

definición:

UWORD NoWaitingAck_RNR( )

parámetros de entrada:

parámetros de salida:

código devuelto:

NO_WAITING_ACK_RNR Se evolucionará denuevo al estado NoWaitingAck_RNR

WAITING_ACK_RR Se evolucionará al estado WaitingAck_RR

WAITING_ACK_RNR Se evolucionará al estado WaitingAck_RNR

descripción:

Esta función implementa el subestado del estado EnlaceEstablecido que corresponde con la siguiente situación: no se espera confirmación de trama I transmitida y el receptor está en condición "Not Ready" o no dispuesto a aceptar una trama I entrante

observaciones:

Se modifican las variables globales "ns", "nr", "vr", "n200", "ns", "vs", "va", num "TxTramasI", "esperandoRespuesta", "ventanaEsperandoTx" y la estructura "ventana" según el flujo de ejecución del proceso

5.7.2 Entidad de Gestión de Capa de Enlace de Datos del lado del ET

A partir del fichero ged_et.c se ejecutará el proceso que implementa la entidad

de gestión de Capa de Enlace de Datos del lado del Equipo Terminal.

Las funciones del proceso de entidad de gestión del lado del Equipo Terminal son básicamente las siguientes:

• Realizar una petición de identidad (IET) al ser requerida por la entidad de Capa de Enlace de Datos (al recibir la señal GED_ASIGNACION_IET_IND)

• Indicar a la entidad de Capa de Enlace de Datos la recepción de un mensaje de identidad asignada o rechazada enviándole la señal GED_ASIGNACION_IET_REQ o GED_NO_DISPONIBLE_IET_REQ respectivamente

• Enviar un mensaje de respuesta de prueba de identidad al ser requerido por un mensaje de petición de prueba de identidad enviado por la entidad de gestión par del lado del Terminal de Red

• Indicar a la entidad de Capa de Enlace de Datos la llegada de un mensaje de supresión de identidad proveniente de la entidad de gestión del lado del Terminal de Red mediante la señal GED_SUPRESION_IET_REQ

El proceso de gestión de capa de Enlace de Datos del lado del Equipo Terminal se creará a partir de la función n2_GED_ETmain a la que se pasa como parámetros de entrada los pipes de escritura y lectura para la comunicación con el proceso que representa la entidad de Capa de Enlace de Datos del lado del Equipo Terminal; además también se le pasa el identificador de ese proceso master.

La función n2_GED_ETmain primeramente realiza ciertas tareas de inicialización del proceso:

• Solicitar al Kernel que le indique el identificador de proceso que le corresponde

• Enviar un primer mensaje a la entidad de Capa de Enlace de Datos del lado del Equipo Terminal donde se le indica el pipe donde debe escribir la información (mensajes) destinados a la entidad de gestión y el identificador del proceso de gestión

• Se solicita el temporizador necesario para el proceso de comunicación con la entidad de Capa de Enlace de Datos.

Siguiendo con la ejecución de la función, se entra en un bucle infinito implementado con la estructura "while(TRUE)" lo que implica que el proceso vivirá

para siempre.

Dentro de este bucle existe una estructura de selección múltiple de señales: "switch(signal)" que se ejecutará cuando llegue alguna señal al proceso.

Las señales que pueden llegar a este proceso son las siguientes:

GED_ASIGNACION_IET_IND: La entidad de Capa de Enlace de Datos envía esta señal cuando desea establecer una conexión de Enlace de Datos y necesita un IET para identificarla. En ese momento la entidad de gestión del lado del Equipo Terminal enviará un mensaje de petición de identidad a la entidad par de gestión del lado del Terminal de Red y esperará un mensaje respuesta indicando la identidad asignada. Si se recibe un mensaje de identidad rechazada o transcurre los N202 (3) periodos correspondientes sin recibir ningún mensaje, se indicará a la entidad de Capa de Enlace de Datos que no hay IET disponible para poder establecer una conexión de Enlace de Datos enviándole la señal GED_NO_DISPONIBLE_IET_REQ.

GED_LIBERACION_MEMO_IND: Como ya hemos comentado esta señal se envía para liberar la memoria correspondiente al mensaje asociado que se recibe en el pipe de mensajes.

GED_UNIDAD_DATOS_IND: Indica la llegada de alguno de los posibles mensajes provenientes de la entidad de gestión par del lado del Terminal de Red. En este caso nos estamos refiriendo a los dos siguientes tipos de mensajes:

Mensaje de petición de prueba de identidad al que la entidad gestora responderá con un mensaje respuesta de prueba de identidad.

Mensaje supresión de identidad: la entidad de gestión enviará la mencionada señal GED_SUPRESION_IET_REQ para indicar a la entidad de Capa de Enlace de Datos que se debe liberar la conexión de Enlace de Datos.

Para la entidad de gestión del lado del Equipo Terminal no se crearon distintos estados como en el caso de la entidad de gestión del lado del Terminal de Red, pues

su funcionamiento es más sencillo. Así la entidad de gestión del lado del Terminal de Red se describirá de una forma más genérica, definiendo los estados en los que se puede encontrar según los tipos de mensajes que puede recibir en cada caso.

5.7.2.1 Descripción de funciones

Indicar que la función "FormarMensaje23" que se utiliza en este proceso es la misma que se utiliza en las entidades-procesos de Capa de Enlace de Datos y nos permite actualizar los valores de los campos de la estructura mensaje23 para un posterior envío.

NOMBRE: n2_GED_ETmain

definición:

void n2_GED_ETmain(pipeEscrituraGED, pipeLecturaGED, identificadorN2)

parámetros de entrada:

UWORD pipeEscrituraGED Sobre este pipe escribe los mensajes la entidad de gestión de Capa de Enlace de Datos que la entidad de Capa de Enlace de Datos leerá

UWORD pipeLecturaGED De este pipe la entidad de gestión lee los mensaje que le envía la entidad de Capa de Enlace de Datos

UBYTE identificadorN2 Identificador del proceso de Capa de Enlace de Datos

parámetros de salida:

código devuelto:

descripción:

Esta función es la implementación de la entidad de gestión de capa de Enlace

de Datos del lado del Equipo Terminal. A partir de esta función se creará el proceso correspondiente a esta entidad

observaciones:

Se modifica la variable global numIETasignado

NOMBRE: IdentificaTramaGED_UI

definición:

UWORD IdentificaTramaGED_UI(*frame, frameLength, *numRef, *tipoMensaje, *indicadorAccion)

parámetros de entrada:

UBYTE *frame Trama recibida del nivel físico

UWORD frameLength Longitud de la trama recibida del nivel físico

parámetros de salida:

UWORD *numRef Número de referencia contenido en la trama

UBYTE *tipoMensaje Tipo de mensaje que corresponde a la trama recibida

UBYTE *indicadorAccion Valor del indicador de Acción recibido en la trama

código devuelto:

TRUE Si la trama es de alguno de los tipos de mensajes definidos

FALSE Si la trama no es de ninguno de los tipos de mensajes recibidos

descripción:

Identifica el tipo de mensaje al que corresponde la trama recibida y devuelve además el número de referencia y el indicador de acción asociados

observaciones:

NOMBRE: FormarTramaGED_UI

definición:

void FormarTramaGED_UI(**trama, *longTrama, *numRef, tipoMensaje, indicadorAccion)

parámetros de entrada:

UBYTE tipoMensaje Valor del campo que codifica el tipo de mensaje

UBYTE indicadorAccion Valor de este campo de la trama que se quiere formar

parámetros de salida:

UBYTE **trama Este descriptor apuntará a la zona de memoria donde se encuentre la trama GED_UI que se quiere formar

UWORD *longTrama Valor de la longitud de la trama que se quiere formar

UBYTE *numRef Valor del campo número de referencia-

código devuelto:

descripción:

Esta función nos permite formar una trama del tipo "TIPO_GED_UI" a partir del tipo de mensaje y del indicador de acción que irán codificados en la trama

observaciones:

NOMBRE: EnviarGEDliberacionMemorReq

definición:

void EnviarGEDliberacionMemoReq( )

parámetros de entrada:

parámetros de salida:

código devuelto:

descripción:

Actualiza la estructura "mensaje23" para escribir un mensaje en el pipe y enviar la señal para indicar la liberación de la información asociada al mensaje

observaciones:

NOMBRE: TratarGEDasignacionIETind

definición:

UWORD TratarGEDasignacionIETind( )

parámetros de entrada:

parámetros de salida:

código devuelto:

FALSE Si no hay IET disponible para que la entidad-proceso de Capa de Enlace de Datos establezca una conexión de Enlace de Datos

TRUE Si hay IET disponible

descripción:

Se ejecutará esta función cuando llegue la señal GED_ASIGNACION_IET_IND. Se enviará a la entidad de gestión par un mensaje de petición de identidad

observaciones:

NOMBRE: CheckNumRef

definición:

UWORD CheckNumRef(numRefRecibido)

parámetros de entrada:

UWORD numRefRecibido Numero referencia recibido

parámetros de salida:

código devuelto:

TRUE Si el número de referencia recibido coincide con alguno de los almacenados

FALSE Si el número de referencia recibido no coincide con ninguno de los almacenados

descripción:

Compara el número de referencia recibido con los almacenados en el vector "vecNumRef". Desde que encuentre uno igual, devuelve TRUE; si no encuentra ninguno igual devuelve FALSE

observaciones:

5.8 LADO DEL TERMINAL DE RED

5.8.1 Entidad de Capa de Enlace de Datos del lado TR

La implementación de la entidad de Capa de Enlace de Datos del lado del Terminal de Red es totalmente análoga a la entidad de Capa de Enlace de Datos del

lado del Equipo Terminal. La diferencia más importante es que no existe un estado "IETnoAsignado", pués será la entidad-proceso de Capa de Enlace de Datos del lado del Equipo Terminal la que solicite establecer la conexión de Enlace de Datos y, por tanto, no será necesario este estado en el lado del Terminal de Red.

Otra diferencia importante es la codificación del bit I/R o bit Instrucción/ Respuesta, que es opuesta en el lado del Terminal de Red de la del lado del Equipo Terminal como ya se explicó en el apartado 3.8.1.2 .

5.8.1.1 Descripción de funciones

Las funciones utilizadas por esta entidad-proceso son del todo análogas a las utilizadas por la entidad-proceso de Capa de Enlace de Datos del lado del Equipo Terminal, sólo variando el interfaz con la Capa Física, que ahora será con el proceso multiplexor. Esto implica que en vez de utilizar las funciones que definen la Capa Física para enviar y recibir las tramas, se leerá / escribirá de / en los pipes de comunicación entre la entidad de Capa de Enlace de Datos y el multiplexor.

5.8.2 Entidad de Gestión de Capa de Enlace de Datos del lado TR

A partir del fichero ged_tr.c se ejecutará el proceso que implementa la entidad de gestión de Capa de Enlace de Datos del lado del Terminal de Red.

Las funciones del proceso de entidad de gestión del lado del Terminal de Red son básicamente las siguientes:

- **Responder a un mensaje recibido de petición de identidad asignando una identidad o rechazando la asignación.**
- **Enviar una petición de prueba de identidad para verificar la posible existencia de una múltiple asignación de IET.**
- **Enviar una supresión de identidad como resultado de la existencia de una múltiple asignación de IET.**

El proceso de gestión de capa de Enlace de Datos del lado del Terminal de Red se creará a partir de la función n2_GED_TRmain a la que se pasa como parámetros

de entrada los pipes de escritura y lectura para la comunicación con el proceso que representa la entidad de Capa de Enlace de Datos (entidad-proceso master); además también se le pasa el identificador de ese proceso master.

La función n2_GED_TRmain primeramente realiza ciertas tareas de inicialización del proceso:

- **Solicitar al Kernel que le indique el identificador de proceso que le corresponde**
- **Enviar un primer mensaje a la entidad de Capa de Enlace de Datos master donde se le indica el pipe donde debe escribir la información (mensajes) destinados a la entida de gestión y el identificador del proceso de gestión**
- **Se solicita el temporizador necesario para el proceso de comunicación con la entidad de Capa de Enlace de Datos.**
- **Se inicializa una serie de parámetros o variables globales**

Siguiendo con la ejecución de la función, se entra en un bucle infinito implementado con la estructura "while(TRUE)" lo que implica que el proceso vivirá para siempre.

Dentro de este bucle existe una estructura de selección múltiple de estado: "switch(estado)". Inicialmente el valor de la variable estado será el valor definido como "IDLE" y por tanto la función-estado que inicialmente se ejecutará será la función "Idle()".

En la figura 5.10 podemos observar gráficamente el organigrama que representa a la función "n2_GED_TRmain" y por tanto podemos visualizar de forma sencilla la forma de comportarse del proceso-entidad de gestión de Capa de Enlace de Datos.

La descripción de este proceso se hará desde el punto de vista de los mensajes recibidos y enviados a/de la entidad par de gestión, haciendo referencia sólo al final, a los tipo de señales que se puede recibir.

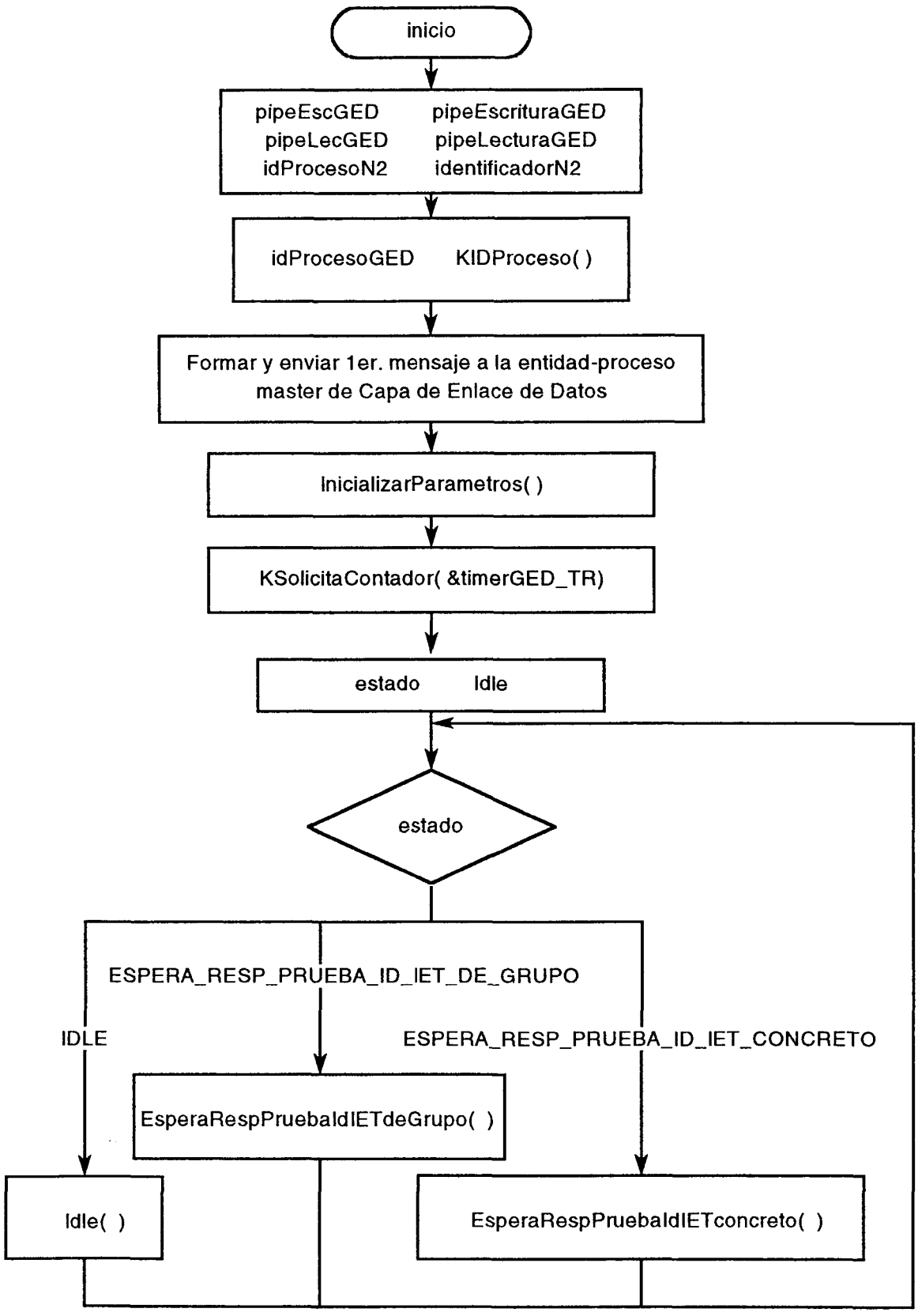


Figura 5.10 Organigrama general entidad-proceso de gestión de Capa de Enlace de Datos (lado TR)

El proceso siempre estará ejecutando una de las funciones incluidas dentro de la estructura de selección múltiple "switch(estado)". Es decir, el proceso siempre estará en alguno de los siguientes tres estados:

Idle: Representa un estado de reposo en el que se encontrará la entidad de gestión la mayoría del tiempo

EsperaRespPruebaIdIETconcreto: Se ejecutará esta función-estado cuando la entidad de gestión del lado del usuario haya enviado un mensaje solicitando verificación de identidad. Al evolucionar a este estado, se enviará un mensaje de petición de prueba de identidad al lado de usuario hasta dos veces separadas un segundo.

Si no se recibe respuesta de prueba de identidad, se supondrá que el valor de IET está libre y disponible para reasignación. Si al final de los dos envios de mensaje de verificación de identidad se recibe más de una respuesta, se considera que existe múltiple asignación. Se enviará entonces un mensaje de supresión de identidad dos veces seguidas. No se realizará ninguna acción si sólo se recibió una respuesta de prueba de identidad.

Se permanecerá en este estado enviando peticiones de prueba de identidad mientras haya alguna petición de verificación de identidad en el buffer de verificaciones de identidad (bufferVerificacionesId).

Si estando en este estado llegase alguna petición de identidad, se enviará un mensaje asignando identidad. Si no quedasen más IETs libres, se guardarán en el buffer de peticiones de identidad (bufferPeticionesId).

Si llega alguna solicitud de verificación de identidad se guarda en el buffer de verificaciones de identidad (bufferVerificacionesId).

Cuando hayamos atendido a todas las peticiones de verificación de identidad del buffer de peticiones de identidad (bufferPeticonesId), se evolucionará al estado "Idle()", si ese que no hay ninguna petición de identidad pendiente. Si existe alguna

petición de identidad pendiente se llevará acabo una petición de prueba de identidad para verificar todos los IETs y se pasará al estado "EsperaRespPruebaIdIETdeGrupo".

EsperaRespPruebaIdIETdeGrupo: Se ejecutará esta función-estado cuando sea necesario enviar un mensaje de petición de prueba de identidad para verificar todos los IET. En este caso el valor del indicador de acción del mensaje a enviar será 127.

Si llega alguna petición de identidad o verificación de identidad, se guardarán es sus respectivos buffers que se atenderán cuando finalize los dos periodos de espera de mensajes de respuesta a prueba de identidad.

Al finalizar dichos dos periodos (2 * T201 segundos), se atenderán las peticiones de identidad contenidas en el buffer de peticiones de identidad. Si en algún momento no hubiera IETs libres para asignar, se enviará un nuevo mensaje de petición de prueba de identidad para todos los IETs y se liberarán los buffers de peticiones y de verificaciones de identidad.

Si al finalizar los dos periodos mencionados, y una vez atendidas las posibles peticiones de identidad, se comprobará si el buffer de verificaciones se encuentra vacío o no. Si está vacío se evolucionará al estado Idle, sino está vació, se evolucionará al estado EsperaRespPruebaIdIETconcreto.

En la siguiente figura 5.11 podemos observar el diagrama de estados que representa a la entidad de gestión de Capa de Enlace de Datos del lado del Terminal de Red y en el cual podemos observar las transiciones entre estados.

Una transición de un estado a otro, como ya se ha comentado, se hará al devolver la función-estado de estado origen, el código que representa la función-estado destino. De esta forma salimos de la función origen a la estructura de selección múltiple "switch(estado)", ejecutando a continuación la función-estado

destino.

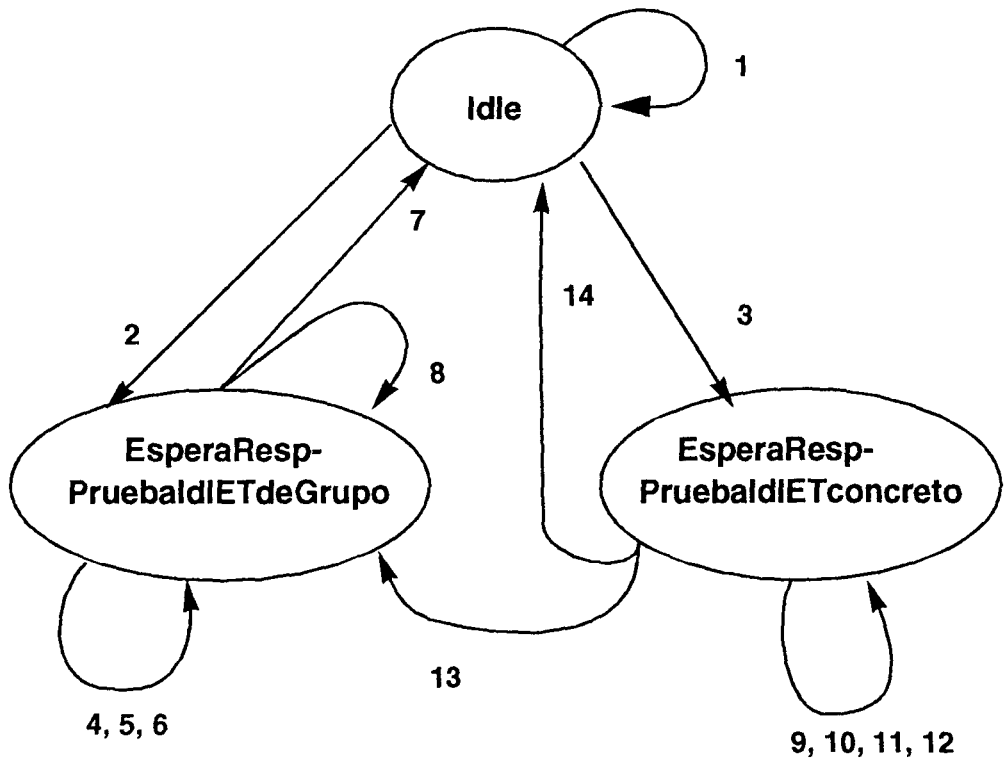


Figura 5.11 Diagrama de estados entidad-proceso de gestión de Capa de Enlace de Datos (lado TR)

A continuación se describen las transiciones entre estados de forma simplificada. Para cada una de las transiciones se indica el suceso o circunstancia de entrada y la respuesta correspondiente (La notación "Id." representa "Identidad", "IET" representa "Identificador de Equipo Terminal" y "Ai" representa "indicador de acción") :

1: Petición Identidad / Se asigna identidad sí hay algún IET disponible

2: Petición Id. / Petición Prueba Id. sí no hay ningún IET disponible (se codificará Ai = 127)

3. Solicitud Verificación Id. (IET concreto) / Se guarda en cola Petición Prueba Id.

4: Petición Id. / Se guarda en cola Peticiones Id.

5: Solicitud Verificación IET concreto / Se guarda en cola Peticiones Verificación Id.

6: Respuesta a Prueba Id. / actualizar asignación en el vector "IETasignados"

7: Fin espera RespPruebaIETdeGrupo y cola Peticiones Id. no vacía / responder asignando IETs hasta que cola esté vacía

8: Fin espera RespPruebaIETdeGrupo y no hay IETs disponibles / Enviar Petición Prueba Id. y liberar resto de la cola de Peticiones Id.

9: Mientras cola Solicitudes Verificación no vacía / Enviar Petición Prueba Id.

10: Petición Id. / Sí hay IET disponible, se responde asignándolo; sí no se guarda en la cola de Peticiones Id.

11: Solicitud Verificación Id. / Guardar en cola Solicitudes Id.

12: Sí terminó el temporizador 2 veces y respPruebaId > 1 / Enviar Supresión Id. y se libera ese IET

13: Sí cola Solicitudes Verificación vacía y cola Petiticiones Id. no vacía / Petición Prueba Id. (IET de Grupo)

14: Sí cola Solicitudes Verificación vacía y cola Peticiones Id. vacía / pasar al estado Idle

Respecto a las señales que pueden llegar a este proceso (a cualquiera de los tres estados mencionados) son las siguientes:

GED_LIBERACION_MEMO_IND: Como ya hemos comentado esta señal se envía para liberar la memoria correspondiente al mensaje asociado que se recibe en el pipe de mensajes.

GED_UNIDAD_DATOS_IND: Indica la llegada de alguno de los posibles mensajes provenientes de la entidad de gestión par del lado del Equipo Terminal.

5.8.2.1 Descripción de funciones

Las siguientes funciones de la entidad-proceso de gestión del lado del Terminal de Red son idénticas a las definidas para la entidad-proceso de gestión del lado del Equipo Terminal:

"FormarTramaGED_UI" : Se definió una sola función que puede ser utilizada por ambas entidades-procesos de gestión de capa, según sus necesidades de formar tramas distintas.

"IdentificaTramaGED_UI" : También se definió una sola función para identificar el tipo de trama recibida, para ambas entidades-procesos de gestión de capa. Los servicios que ofrecen ambas funciones serán aprovechados por cada una de las entidades de gestión de forma complementaria (por ejemplo la entidad de gestión del lado del ET necesitará identificar las tramas que le envía la entidad de gestión del lado TR pero no las tramas que genera la propia entidad de gestión del lado del ET y v.v.).

"EnviarGEDliberacionMemoReq": Será utilizada de la misma forma por las dos entidades de gestión de Capa de Enlace de Datos.

"FormarMensaje23" : Es la misma función que la utilizada por las entidades de Capa de Enlace de Datos de los lados del ET y del TR.

NOMBRE: n2_GED_TRmain

definición:

n2_GED_TRmain(pipeEscrituraGED, pipeLecturaGED, identificadorN2)

parámetros de entrada:

UWORD pipeEscrituraGED Pipe donde el proceso-entidad de gestión escribirá los mensajes dirigidos al proceso-entidad de Capa de Enlace de Datos

UWORD pipeLecturaGED Pipe de donde el proceso-entidad de gestión leerá los mensajes escritod por el proceso-entidad de Capa de Enlace de Datos

parámetros de salida:

código devuelto:

descripción:

A partir de esta función se ejecutará el proceso que representa a la entidad de gestión de Capa de Enlace de Datos del lado del Terminal de Red

observaciones:

NOMBRE: Idle

definición:

UWORD Idle( )

parámetros de entrada:

parámetros de salida:

código devuelto:

ESPERA_RESP_PRUEBA_ID_IET_DE_GRUPO Si se prentende evolucionar al estado-función EsperaRespPruebaIdIETdeGrupo.

ESPERA_RESP_PRUEBA_ID_IET_CONCRETO Si se prentende evolucionar al estado-función EsperaRespPruebaIdIETconcreto

IDLE Si se va a evolucionar al estado-función Idle denuevo

descripción:

Esta función es la implementación del estado "idle" o de reposo en el que se encontrará la entidad de gestión del lado del Terminal de Red la mayoría del tiempo

observaciones:

Se modifica la variable global vector de IETs asignados ("vecIETsAsignados")

NOMBRE: EsperandoRespPruebaIdIETdeGrupo

definición:

UWORD EsperandoRespPruebaIdIETdeGrupo( )

parámetros de entrada:

parámetros de salida:

código devuelto:

ESPERA_RESP_PRUEBA_ID_IET_DE_GRUPO Si se prentende evolucionar al estado-función EsperaRespPruebaIdIETdeGrupo, o lo que es lo mismo, permanecer en él

ESPERA_RESP_PRUEBA_ID_IET_CONCRETO Si se prentende evolucionar al estado-función EsperaRespPruebaIdIETconcreto

IDLE Si se pretende evolucionar al estado-función Idle

descripción:

Se ejecutará esta función-estado cuando la entidad de gestión del lado de usuario haya enviado un mensaje solicitnado verificación de identidad

observaciones:

Se modifican las varialbes globales "tramaIdentificada", "indicadorAccion", "frenteBpeticionesId", "finalBpeticionesId", "frenteBverificacionesId", "finalBverificacionesId" y el vector "vecIETsAsignados"

NOMBRE: EsperaRespPruebaIdIETdeGrupo

definición:

UWORD EsperaRespPruebaIdIETdeGrupo( )

parámetros de entrada:

parámetros de salida:

código devuelto:

ESPERA_RESP_PRUEBA_ID_IET_DE_GRUPO Si se prentende evolucionar al estado-función EsperaRespPruebaIdIETdeGrupo

ESPERA_RESP_PRUEBA_ID_IET_CONCRETO Si se prentende evolucionar al estado-función EsperaRespPruebaIdIETconcreto, o lo que es lo mismo, permanecer en él

IDLE Si se pretende evolucionar al estado-función Idle

descripción:

Se ejecutará esta función-estado cuando sea neceario enviar un mensaje de petición de prueba de identida para verificar todos los IETs. En este caso el valor del indicador de acción del mensaje a enviar será 127

observaciones:

Se modifican las siguientes variables globales en función del flujo de ejecución: "numIETenPrueba", "n204ByMe"

NOMBRE: GetNewIET

definición:

UBYTE GetNewIET( )

parámetros de entrada:

parámetros de salida:

código devuelto:

Devolverá un valor libre de IET del rango 64 a 126 o el IET_DE_GRUPO (127)

descripción:

Esta función devolverá un valor libre de IET del rango 64 a 126 o el IET_DE_GRUPO (127) indicando que no hay ningún IET disponible

observaciones:

NOMBRE: EnviarGEDunidadDatosReq

definición:

void EnviarGEDunidadDatosReq( )

parámetros de entrada:

parámetros de salida:

código devuelto:

descripción:

Esta función sirve para enviar la señal GED_UNIDAD_DATOS_REQ y formar y enviar el mensaje asociado

observaciones:

NOMBRE: GuardaBverificacionesId

definición:

UWORD GuardaBverificacionesId(numIET)

parámetros de entrada:

UBYTE numIET Número de IET que corresponde al proceso emisor

parámetros de salida:

código devuelto:

TRUE Si logra insertar con éxito un elemento en la cola de verificaciones de identidad

FALSE Si la cola está llena y no se puede insertar ningún elemento

descripción:

observaciones:

Como para el resto de buffers (estructuras de tipo cola diseñados en esta implementación), suponemos que el buffer de verificaciones de identidad tiene una longitud suficientemente grande y nunca se desborda

NOMBRE: RecuperaBverificacionesId

definición:

UBYTE RecuperaBverificacionesId( )

parámetros de entrada:

parámetros de salida:

código devuelto:

Devuelve el primer IET de la conexión de Enlace de Datos que ha pedido que se realice una verificación de su identidad

descripción:

Esta función auxiliar nos devuelve el valor de IET de la primera conexión de

Enlace de Datos que está esperando una verificación de identidad

observaciones:

Modifica la variable global frenteBverificaciones incrementándola apropiadamente

NOMBRE: GuardaBpeticionesId

definición:

UWORD GuardaBpeticionesId(numRef)

parámetros de entrada:

numRef Valor del número de referencia que se quiere guardar en la cola de peticiones de identidad

parámetros de salida:

código devuelto:

TRUE Si logra insertar con éxito un elemento (numRef) en la cola de peticiones de identidad

FALSE Si la cola está llena y no se puede insertar ningún elemento

descripción:

Inserta un elemento (numRef) en la cola de peticiones de identidad en espera y devuelve TRUE. Si la cola está llena devuelve FALSE

observaciones:

Incrementa la variable global frenteBpeticionesId de forma adecuada

5.8.3 Proceso multiplexor o concentrador del lado del TR

A partir del fichero mux.c se ejecutará el proceso multiplexor o concentrador. Es un proceso intermedio entre las entidades-procesos de Capa de Enlace de Datos del lado del Terminal de Red y la capa física.

Sus funciones son:

- **Dirigir las tramas generadas por las entidades-procesos de Capa de Enlace de Datos a la capa física.**
- **Dirigir las tramas que se reciben desde la capa física a las correspondientes entidades-procesos destino de capa de Enlace de Datos.**

5.8.3.1 Descripción de funciones

En este apartado describimos las funciones correspondientes al fichero a partir del cual se crea el proceso multiplexor o concentrador (del lado del Terminal de Red):

NOMBRE: n2MuxTRmain

definición:

void n2MuxTRmain(pipeLectura, idPuerto)

parámetros de entrada:

UWORD pipeLectura Pipe para la comunicación con los procesos (entidades-procesos) de capa de Enlace de Datos del lado del Terminal de Red. Estos procesos escribirán en este pipe los mensajes que el proceso multiplexor leerá

UWORD idPuerto Identificador puerto físico que controla (con el que se comunica)

parámetros de salida:

código devuelto:

descripción:

Esta función es la llamada al código que representa el proceso multiplexor o concentrador intermedio entre las entidades de Capa de Enlace de Datos del lado del Terminal de Red y la Capa Física. Las tareas que realiza son:

- Solicitar al Kernel que indique cúal es el identificador de proceso que se ha creado a partir de la ejecución de esta función
- Ejecutar función que inicializa parámetros o variables globales
- Entrar en el bucle infinito "while(TRUE)" que implica que el proceso vivirá para siempre

observaciones:

NOMBRE: InicializarParametros

definición:

void InicializarParametros( )

parámetros de entrada:

parámetros de salida:

código devuelto:

descripción:

Su función es inicializar determinados parámetros (variables globales) que se utilizarán en la comunicación con el puerto físico, concretamente con el Controlador de Comunicaciones Serie (SCC). También inicializa los elementos del vector "vecMux" y realiza la inicialización de los índices de los buffers (estructuras tipo cola) utilizados

observaciones:

NOMBRE: TratarFIdatosRecibidos

definición:

UWORD TratarFIdatosRecibidos(indiceTRmaster)

parámetros de entrada:

UBYTE indiceTRmaster Indice que indica cúal de los elementos del vector "vecMux" pertenece al proceso TR master

parámetros de salida:

código devuelto:

TRUE Se diseñó para que siempre devolviese este valor para simplificar la programación. Este valor indica que se ha ejecutado correctamente

descripción:

Se ejecutará esta función cada vez que se reciba un signal FI_DATOS_RECIBIDOS o FI_ERROR_IND porque se ha recibido una trama del nivel físico. Se procesa la trama sino hubo error al recibirla y según el tipo de trama se enviará al proceso adecuado: al proceso TR master si se trata de una trama de información sin confirmar o una trama dirigida al Gestor de Enlace de Datos (del lado del Terminal de Red), o al proceso TR que corresponda con el IET utilizado en el campo de dirección de esa trama recibida.

Si la trama recibida es del tipo SABME, se enviará al proceso con IET asociado igual al IET recibido o a un proceso TR libre. Si no hay ningún proceso libre, se descarta. Si es una trama no reconocida según la implementación realizada, se descarta (se libera la memoria que ocupa)

observaciones:

Si se produjo algún error en la recepción de la trama, ésta se descarta. Si se produce un error de desactivación del nivel físico, se indicará a todos los procesos de nivel 2 del lado TR (que estén comunicándose con su respectivo remoto), que se ha caído en "enlace de nivel físico" y por tanto, deberán pasar al estado "EnlaceLiberado". Se modificaría en este caso los elementos del vector "vecMux" que estén en comunicación con su remoto. Cuando se libere la trama recibida, se

modificará, obviamente, la variable global "header".

NOMBRE: EnviarTramaAlTRdestino

definición:

void EnviarTramaAlTRdestino(indiceTRdestino)

parámetros de entrada:

UWORD indiceTRdestino

parámetros de salida:

código devuelto:

descripción:

Se envía un mensaje (mensajeMuxTR) al proceso-entidad de capa de Enlace de Datos del lado del Terminal de Red (a el indicado por la variable "indiceTRdestino") y se le indica con el signal MUX_TR_DATOS_IND que equivale a el signal FI_DATOS_RECIBIDOS o FI_ERROR_IND que produce el nivel físico

observaciones:

NOMBRE: EnviarTramaUlalTRmaster

definición:

void EnviarTramaUlalTRmaster(indiceTRmaster)

parámetros de entrada:

UWORD indiceTRmaster Indica la posición relativa en el vector "vecMux" en la que se encuentra el elemento correspondiente al proceso de Capa de Enlace de Datos master

parámetros de salida:

código devuelto:

descripción:

Se envía un mensaje (mensajeMuxTR) al proceso-entidad master conteniendo la trama UI recibida del nivel físico y que se encuentra en las variables globales header y lengHeader (trama recibida y su longitud)

observaciones:

NOMBRE: FormarMensajeMuxTR

definición:

void FormarMensajeMuxTR(messageMuxTR, tipoMessage, idProcesoTR, header, lengHeader, info, longInfo)

parámetros de entrada:

UWORD tipoMessage Identifica el tipo de mensaje (repetición del signal asociado a este mensaje) que se enviará a la entidad-proceso de nivel 2

UBYTE idProcesoTR

UBYTE *heder Trama que se va a enviar a la entidad-proceso de nivel 2

UWORD lengHeader Longitud de la trama que se va a enviar a la entidad-proceso de nivel 2

UBYTE *info El valor de este parámetro será NULL

UWORD longInfo El valor de este parámetros será cero

parámetros de salida:

MENSAJE_MUX_TR *messageMuxTR

código devuelto:

descripción:

Esta función sirve para formar un mensaje (mensajeMuxTR) que luego será enviado a la entidad-proceso de Capa de Enlace de Datos correspondiente

observaciones:

NOMBRE: IdentificaTramaRecibida

definición:

UWORD IdentificaTramaRecibida(frame, frameLength, numIETrecibido)

parámetros de entrada:

UBYTE *frame	Trama recibida
UWORD frameLength	Longitud de la trama recibida

parámetros de salida:

UBYTE *numIETrecibido	Esta variable tomará el valor del Identificador de Equipo Terminal (IET) del campo de dirección de la trama recibida

código devuelto:

Devolverá el tipo de trama identificada, es decir, uno de los siguientes tipos con su respectivo significado (los lugares de destino

TRAMA_TR_UI	La trama recibida es del tipo información sin confirmar (UI)
SABME	Tipo SABME
TRAMA_TR	Trama de Supervisión, de Transferencia de Información o No Numerada dirigida al respectivo proceso TR (entidad de Capa de Enlace de Datos) según el valor del IET recibido

TRAMA_GED_U Trama de información sin numerar destinada al proceso Gestor de Enlace de Datos del lado del Terminal de Red

DESCONOCIDO La trama recibida es desconocida para la implementación del protocolo LAP-D desarrollada

descripción:

Identifica el tipo de trama recibida del nivel físico y extrae el valor del IET del campo de dirección de dicha trama recibida

observaciones:

NOMBRE: CheckTRdisponible

definición:

UWORD CheckTRdisponible(numIETrecibido, indiceTRdestino)

parámetros de entrada:

UBYTE numIETrecibido

parámetros de salida:

UWORD *indiceTRdestino

código devuelto:

DISPONIBLE Si existe un proceso-entidad de Capa de Enlace de Datos comunicándose con un proceso remoto con el IET recibido en esa trama SABME; o existe un proceso-entidad de Capa de Enlace de Datos libre para poder establecer una comunicación con el extremo que está solicitando una conexión de Enlace de Datos utilizando el IET recibido en dicha trama SABME

NO_DISPONIBLE Si no existe ningún proceso-entidad de Capa de Enlace de Datos libre

descripción:

Se ejecuta esta función cada vez que se recibe una trama SABME

1.- Si existe un proceso-entidad de Capa de Enlace de Datos que está comunicándose con un proceso par del lado del ET con un IET igual al recibido en la trama, la enviamos a ese proceso

2.- Si no se cumple lo anterior, pero existe un proceso del lado del TR que no está comunicándose con ningún proceso del lado ET, se asignará a la nueva conexión de Enlace de Datos este IET recibido y se asociará ese IET con ese proceso que no tiene ninguna conexión asociada.

3.- Si no se cumple ninguna de las dos anteriores posibilidades, la función devuelve el valor "NO_DISPONIBLE"

observaciones:

En el segundo supuesto se modificará el campo numIETasignado del elemento del vector vecMux tomando el valor del IET recibido (numIETrecibido)

NOMBRE: CheckIETrecibido

definición:

UWORD CheckIETrecibido(numIETrecibido, inidiceTRdestino)

parámetros de entrada:

UBYTE numIETrecibido

parámetros de salida:

UWORD *indiceTRdestino

código devuelto:

INCLUIDO Si existe algún proceso-entidad de Capa de Enlace de Datos que esté comunicándose con su par remoto utilizando el IET recibido

NO_INCLUIDO Si no hay ningún proceso-entidad de Capa de Enlace de Datos que esté comunicándose con su par remoto utilizando el IET recibido

descripción:

Comprueba si alguno de los procesos-entidad de Capa de Enlace de Datos del lado del Terminal de Red está comunicándose con su remoto utilizando el valor del IET recibido. Si es así, devuelve el índice de ese TR en la variables indiceTRdestino y la función toma el valor INCLUIDO. En caso contrario, la función toma el valor NO_INCLUIDO

observaciones:

NOMBRE: GuardarBtxMux

definición:

UWORD GuardarBtxMux(idProcesoTRemisor)

parámetros de entrada:

UWORD idProcesoTRemisor

parámetros de salida:

código devuelto:

TRUE Si logra insertar un elemento (identificador del TR emisor de dicha trama) en el buffer de transmisión del multiplexor

FALSE Si el buffer está lleno

descripción:

Inserta un elemento (idProcesoTRemisor) en la cola "bufferTxMux" y devuelve TRUE. Si la cola está llena, devuelve FALSE

observaciones:

Suponemos que el bufferTxMux está dimensionado de tal manera que en ningún caso se llenará

NOMBRE: RecuperarBtxMux

definición:

UWORD RecuperarBtxMux( )

parámetros de entrada:

parámetros de salida:

código devuelto:

TRUE Si se la cola bufferTxMux no está vacía y se logra enviar la correspondiente señal de indicación de haber enviado la trama a través del nivel físico para así poder liberar la memoria correspondiente al cabecero y la información, si es el caso

FALSE Si la cola bufferTxMux está vacía

descripción:

Se ejecutará esta función cada vez que se reciba un signal FI_DATOS_ENVIADOS. Se enviará al proceso-entidad emisor de la trama el signal MUX_TR_DATOS_ENVIADOS_IND para indicar que el nivel físico ya envió la trama correspondiente

observaciones:

NOMBRE: GuardarBtramasAtx

definición:

UWORD GuardarBtramasAtx(idProcesoTRemisor, header, lengHeader, info,

longInfo)

parámetros de entrada:

UBYTE idProcesoTRemisor Identificador proceso-entidad de Capa de Enlace de Datos emisor de la trama

UBYTE *header Cabecero de la trama a enviar

UWORD lengHeader Longitud del cabecero

UBYTE *info Campo de información de la trama a enviar, si es una trama de información, sino se pasará el valor NULL a este parámetro

UWORD longInfo Longitud del campo de información de la trama. Si la trama no es información, se pasará el valor cero a este parámetro

parámetros de salida:

código devuelto:

TRUE

FALSE

descripción:

Inserta un elemento en el buffer de tramas a transmitir ,"bufferTramasAtx", (identificador del proceso emisor, cabecero y su longitud, información y su longitud; y devuelve TRUE. Si la cola está llena devuelve FALSE

observaciones:

Suponemos que la cola "bufferTramasAtx" está dimensionada de tal forma que nunca se llena

NOMBRE: RecuperarBtramasAtx

definición:

UWORD RecuperarBtramasAtx(idProcesoTRemisor, header, lengHeader, info, longInfo)

parámetros de entrada:

parámetros de salida:

UYBTE *idProcesoTRemisor Identificador proceso-entidad de Capa de Enlace de Datos emisor de la trama

UBYTE **header Cabecero de la trama a transmitir

UWORD *lengHeader Longitud del cabecero de la trama a transmitir

UBYTE **info Campo de información de la trama a transmitir

UWORD *longInfo Longitud del campo de información de la trama a transmitir

código devuelto:

TRUE Si la cola bufferTramasAtx está vacía

FALSE Si se logra recuperar un elemento de la cola bufferTramasAtx

descripción:

Cuando se ha recibido un signal FI_DATOS_ENVIADOS del nivel físico y se puede transmitir una trama en cola de espera, se recupera esa trama (cabecero e información con sus respectivas longitudes y el identificador del proceso que quiere enviarla -proceso o entidad de Capa de Enlace de Datos emisor-)

observaciones:

Capítulo 6: Entorno de Pruebas en MS-DOS. Explicación del software

6.1 GENERAL

Se diseñaron una serie de funciones para simular los servicios ofrecidos por el módulo Kernel y por el nivel físico en el entorno MS-DOS. Con la ayuda de estas funciones podremos ejecutar cada uno de los programas creados en el entorno MS-DOS.

Las comunicaciones con la capa o nivel 3 y con la capa o nivel físico se simulan de la siguiente manera:

Las señales (procedentes del nivel 3 o del nivel físico), los mensajes (procedentes del nivel 3) o las tramas (procedentes del nivel físico) que recibiría el proceso de nivel 2 en cada caso, se introducen por el teclado cuando corresponda leerlos del pipe de señales, de mensajes o del nivel físico respectivamente.

Cuando sea el propio proceso de nivel 2 al que corresponda el turno de enviar una señal (al nivel 3), un mensaje (al nivel 3) o una trama (al nivel físico) según los requerimientos del flujo del algoritmo, éstos se mostrarán en la pantalla.

Para solicitar y liberar memoria dinámica durante el desarrollo de los algoritmos (por ejemplo, para formar los cabeceros de las tramas LAP-D), se utilizaron las funciones "malloc()" y "free()".

De esta forma ejecutamos paso a paso cada uno de los programas creados comprobando todas las posibles opciones de los algoritmos y verificando su correcto funcionamiento según los criterios de diseño.

Los resultados de las pruebas conjuntas realizadas en el entorno MS-DOS y sobre el sistema empotrado se describen en el capítulo 8 (Banco de Pruebas).

6.2 DESCRIPCION DE FUNCIONES

A continuación se realiza una descripción de las funciones implementadas para la simulación de los servicios suministrados por el módulo Kernel y por la capa física.

6.2.1 Funciones que simulan el módulo Kernel

NOMBRE: Espera

definición:

void Espera( )

parámetros de entrada:

parámetros de salida:

código devuelto:

descripción:

Al ejecutar esta función el programa se detiene hasta que se pulse una tecla. Sirve para realizar una parada en la ejecución del código en la simulación para que el usuario pueda observar lo que hay en la pantalla

observaciones:

NOMBRE: DisplayData

definición:

void DisplayData(data, dataLength, tipoDisplay)

parámetros de entrada:

UBYTE *data Puntero a la zona de memoria donde están los octetos o bytes que se quieren mostrar

UWORD dataLength Longitud de los datos (número de octetos que se quiere visualizar)

char tipoDisplay Indica la forma de visualización de los octetos: como pareja de números hexadecimales o en decimal

parámetros de salida:

código devuelto:

descripción:

Muestra por pantalla los bytes u octetos que hay a partir de la posición de memoria a la que apunta el puntero data. El número de octetos que se visualizará viene determinado por el valor dataLength referenciado

observaciones:

Utilizaremos siempre la opción de visualización de octetos como pareja de números hexadecimales

NOMBRE: DisplaySignal

definición:

void DisplaySignal(signal)

parámetros de entrada:

UWORD signal Valor numérico de la señal que se quiere visualizar en la pantalla

parámetros de salida:

código devuelto:

descripción:

Muestra el tipo de señal por pantalla como cadena de caracteres, la misma cadena de caracteres utilizada en las definiciones realizadas con los "define"

observaciones:

NOMBRE: EligeSignal

definición:

void EligeSignal(signal)

parámetros de entrada:

parámetros de salida:

UWORD signal Tipo de señal elegida

código devuelto:

descripción:

Esta función se utiliza para simplificar la selección de una señal o signal. Se presenta primeramente un menú donde se requiere seleccionar uno de los siguientes tipos de señal:

- ED: Señales intercambiadas através del interfaz de Capa de Enlace de Datos
- GED: Señales intercambiadas entre la Entidad de Gestión de Capa de Enlace de Datos y la entidad de Capa de Enlace de Datos
- FI: Señales intercambiadas entre la entidad de Capa de Enlace de Datos y la Capa Física
- MUX_TR: Señales intercambiadas entre el proceso multiplexor o concentrador y las entidades de Capa de Enlace de Datos
- O señal del teclado (en este caso se introducirá el valor numérico de la señal desde el teclado)

Las cuatro primeras opciones se desglosan a su vez en un submenú donde se

requiere seleccionar una señal concreta dentro del tipo elegido

observaciones:

NOMBRE: EligeError

definición:

void EligeError(error)

parámetros de entrada:

parámetros de salida:

UWORD *error Tipo de error elegido

código devuelto:

descripción:

Función auxiliar a la función FRecibeDatos. Se utiliza par poder seleccionar el tipo de error asociado a una trama que podría recibirse del nivel físico. Se solicita seleccionar entre los siguientes posibles opciones:

KNO_ERROR Ningún error recibido

ERROR_DESACT Error nivel físico desactivado

Opción "cualquier otro tipo de error" que engloba a los siguientes tipos de errores indicados por el nivel físico a la simulación de proceso de nivel 2:

ERROR_OVERRUN Error de overrun o "sobremuestreo"

ERROR_FRAMING Error de "framing"

ERROR_PARITY Error en la comprobación de la paridad

ERROR_BREAK Error de "ruptura"

ERROR_ABORT Error por trama considerada como abortada

ERROR_NONOCTET Error que indica recepción de una trama formada por un número de bits no múltiplo de ocho (trama no formada por un número entero de octetos)

EROR_CRC Error al comprobar el código redundante cíclico recalculado con el de la trama recibida

observaciones:

NOMBRE: KSignal

definición:

UWORD KSignal(proceso, signal)

parámetros de entrada:

UBYTE signal Indica el tipo de señal que se quiere hacer llegar al proceso indicado en la variable del mismo nombre ("proceso")

UWORD proceso Identificador del proceso al que se quiere enviar el señal

parámetros de salida:

código devuelto:

KNO_ERROR Todo correcto, indicaría que la señal se escribió en el pipe de señales del proceso referenciado correctamente

descripción:

Muestra el número de proceso (identificador de proceso) al que va dirigida la señal y el valor hexadecimal de la señal y la representación en caracteres

observaciones:

NOMBRE: KLeeSignal

definición:

UWORD KLeeSignal(signal)

parámetros de entrada:

parámetros de salida:

UWORD *signal Valor de la señal que se quiere leer del teclado, simulando su llegada desde la entidad-proceso de capa 3 o desde la capa física

código devuelto:

KNO_ERROR Indica lectura correcta de la señal

descripción:

Para simular la lectura de una señal del pipe de señales recibidas. Se pedirá la selección de la señal (por teclado) empleando la función auxiliar "EligeSignal"

observaciones:

NOMBRE: KWait

definición:

UWORD KWait( )

parámetros de entrada:

parámetros de salida:

código devuelto:

KINTERRUP Indicaría que ha llegado una señal

descripción:

Cuando un proceso utiliza este servicio del módulo Kernel, el proceso pasa a un

"estado dormido o inactivo" hasta que reciba una señal y se despierte. En esta simulación se devuelve directamente el código que indica que se ha recibido una señal pues no tiene sentido simular el estado dormido ya que sólo tratamos con un proceso

observaciones:

NOMBRE: KIDProceso

definición:

UBYTE KIDProceso( )

parámetros de entrada:

parámetros de salida:

código devuelto:

Valor del identificador de proceso que solicitó este servicio (rango 0 a 127)

descripción:

Cuando un proceso utiliza este servicio del módulo Kernel está solicitando que se le indique cuál es el identificador asociado con el proceso que representa. En nuestra simulación se solicita un valor (rango 0 a 127) para identificar el proceso simulado

observaciones:

NOMBRE: KSolicitaContador

definición:

UWORD KSolicitaContador(contador)

parámetros de entrada:

parámetros de salida:

UWORD *contador Identificador del contador que el Kernel devolvería, en nuestro caso es el valor que el usuario introduce por el teclado

código devuelto:

KNO_ERROR Todo correcto

descripción:

El módulo Kernel entregaría el valor de un contador de tiempo o temporizador; en esta simulación se solicita introducir desde el teclado un valor (rango 0 a 65535) para identificar el contador utilizado

observaciones:

NOMBRE: KActivaContador

definición:

UWORD KActivaContador(contador, tiempo)

parámetros de entrada:

UWORD Contador Valor del temporizador que se quiere activar para empezar una nueva cuenta

UWORD tiempo Tiempo en segundos que se quiere que el contador cuente o espere

parámetros de salida:

código devuelto:

KNO_ERROR Todo correcto

descripción:

El Kernel activaría el contador indicado para que cuente o espere el número de segundos que se especifica en el parámetro de entrada "tiempo". En nuestra simulación simplemente se muestra en pantalla el valor del contador y el tiempo que se quiere que cuente antes de que la llamada a la función "KEsperaContador" devuelva KNO_EROR indicando la finalización de la cuenta (la finalización del tiempo)

observaciones:

NOMBRE: KEsperaContador

definición:

UWORD KEsperaContador(contador)

parámetros de entrada:

UWORD Contador Valor que identifica el temporizador o contador de tiempo por el que habría que esperar a que terminara su cuenta o a que llegara alguna señal

parámetros de salida:

código devuelto:

KNO_EROR Para indicar que ha terminado la cuenta del temporizador

KINTERRUP Para indicar que ha llegado una señal antes de que la cuenta haya terminado

descripción:

En el módulo Kernel al solicitar este servicio el proceso pasa a estado "dormido o inactivo" hasta que termine el contador (hasta que transcurra el tiempo de la cuenta) y entonces se activa el proceso devolviendo el código KNO_ERROR. El

proceso puede también activarse si recibe alguna señal; en ese caso se devuelve el código KINTERRUP

observaciones:

NOMBRE: KPideMemoria

definición:

UWORD KPideMemoria(longDatos, datos)

parámetros de entrada:

UWORD longDatos Tamaño de la cantidad de memoria solicitada

parámetros de salida:

UWORD **datos Puntero a la zona de memoria dinámica

código devuelto:

KNO_ERROR

KNO_MEMORIA Si no hay disponible memoria dinámica del tamaño solicitado

descripción:

Esta función devuelve una cantidad de memoria dinámica a la que apuntará el puntero datos. Esta cantidad de memoria dinámica es el número de octetos especificado en el parámetro longDatos. Se mostrará en pantalla el número de octetos solicitado y la posición de memoria a partir de la cual se almacenan

observaciones:

NOMBRE: KLiberaMemoria

definición:

UWORD KLiberaMemoria(datos)

parámetros de entrada:

UWORD *datos Puntero a la zona de memoria que se desea liberar

parámetros de salida:

código devuelto:

KNO_ERROR Todo correcto

descripción:

Libera la zona de memoria a la que apunta el puntero datos

observaciones:

Los bloques liberados pueden volver a ser utilizados. Este servicio o función se comporta de forma impredecible si el descriptor o puntero recibido no fue inicializado por el servicio KPideMemoria

NOMBRE: KEscribePipe

definición:

UWORD KEscribePipe(pipe, data, longData, espera)

parámetros de entrada:

UWORD pipe Identificador del pipe donde se quiere escribir los datos (el mensaje)

void *data Puntero a los datos que se quiere escribir en el pipe

UWORD longData Longitud de los datos o mensaje que se

quiere escribir en el pipe

UWORD espera Este parámetro indicaría al módulo Kernel si se quiere esperar o no a escribir los datos si no hay espacio en el pipe referenciado

parámetros de salida:

código devuelto:

KNO_ERROR Todo correcto

descripción:

El servicio del módulo Kernel guardaría los bytes uno a continuación del otro en el pipe referenciado. En nuestra simulación se escribe en pantalla los distintos campos del mensaje que se quiere guardar en el pipe. Así se visualizará el valor del pipe referenciado, el tipo de mensaje (que generalmente será una señal y se indicará su tipo en caracteres ayudándonos de la función auxiliar "DisplaySignal"), la longitud de los datos y se presentará los datos como parejas de números hexadecimales con la ayuda de la función DisplayData.

Por último se mostrará en pantalla la modalidad de escritura en el pipe: KWAIT o KNOWAIT.

observaciones:

Esta implementación del protocolo LAP-D siempre utiliza la modalidad KWAIT

NOMBRE: KLeePipe

definición:

UWORD KLeePipe(pipe, data, longData, espera)

parámetros de entrada:

UWORD pipe Identificador del pipe de donde se leerán los datos (el mensaje)

UWORD espera Este parámetro indicaría la modalidad de espera

UWORD longData Longitud de los datos que se quiere leer del pipe

parámetros de salida:

void *data Puntero a la zona de memoria donde se encontrará los datos (o mensaje) leídos del pipe

código devuelto:

KNO_ERROR Todo correcto

descripción:

El servicio del módulo Kernel leería del pipe referenciado todos los bytes uno a continuación del otro. En esta simulación se leerá del teclado el supuesto mensaje que, utilizando el servicio del Kernel, se leería del pipe. Se solicitará memoria dinámica para los datos que simuladamente nos envía el nivel 3

observaciones:

6.2.2 Funciones que simulan la Capa Física

Las cuatro siguientes funciones corresponden a la simulación de los servicios ofrecidos por la capa o nivel físico:

NOMBRE: FIniciaSCC

definición:

UWORD FIniciaSCC(scc, protocolo, param)

parámetros de entrada:

UWORD scc Identificador del Controlador de Comunicaciones Serie que se inicializará (en ocasiones, al SCC le llamaremos puerto

físico)

UWORD protocolo Identificador del protocolo con el que se quiere inicializar el SCC (en nuestro caso sería el protocolo HDLC)

PARAM_SCC *param Puntero a la estructura param que contendrá una serie de campos donde se especifican ciertos parámetros para la comunicación con el nivel físico (con el SCC, "a través" del nivel físico)

parámetros de salida:

código devuelto:

TRUE Todo correcto

descripción:

Esta función serviría para iniciar los parámetros del Controlador Serie de Comunicaciones. En esta simulación se muestra en la pantalla los parámetros con los que se iniciaría el puerto físico o SCC

observaciones:

NOMBRE: FEnviaDatos

definición:

UWORD FEnviaDatos(scc, datos1, longitud1, datos2, longitud2)

parámetros de entrada:

UWORD scc Identificador del SCC al que se envían los datos para ser enviados al bus pasivo

void *datos1 Puntero a la zona de memoria donde se encuentra la primera secuencia de octetos (generalmente se tratará de la cabecera de la trama)

UWORD longitud1 Longitud de la cabecera de la trama

void* datos2 Puntero a la zona de memoria donde se encuentra la segunda secuencia de datos (campo de información cuando lo haya, sino apuntará a NULL)

UWORD longitud2 Longitud del campo de información de la trama (si no lo hay, valdrá cero)

parámetros de salida:

código devuelto:

TRUE Si se quiere indicar que se logró enviar los datos a través del SCC

FALSE Si se quiere indica que no se logró enviar los datos a través del SCC

descripción:

El objetivo de la simulación de esta función es mostrar los datos (trama con su cabecero y campo de información, si lo hay) que se enviaría al SCC destino

observaciones:

NOMBRE: FRecibeDatos

definición:

UWORD FRecibeDatos(scc, datos, longitud, error)

parámetros de entrada:

UWORD scc Identificador del SCC del que se quiere recibir los datos

parámetros de salida:

void **datos Descriptor a la zona de memoria donde se encontrarán los datos recibidos del SCC

UWORD *longitud Longitud de los datos recibidos del SCC

UWORD *error Tipo de error que se recibe (o indicación de que no hay error)

código devuelto:

TRUE Todo correcto

descripción:

El objetivo de la simulación de esta función es leer del teclado la supuesta trama recibida del nivel físico y el tipo de error que se lee utilizando la función auxiliar EligeError (leyéndose KNO_EROR -no hubo error- o alguno de los errores contemplados)

observaciones:

NOMBRE: FActivaSCC

definición:

UWORD FActivaSCC(scc)

parámetros de entrada:

UWORD scc Valor del Controlador de Comunicaciones Serie (SCC) que se pretende activar

parámetros de salida:

código devuelto:

TRUE Si se quiere indicar que se pudo activar el nivel físico

FALSE Si se quiere indicar que no se pudo activar el nivel físico

descripción:

Con esta función se quiere simular la activación de un SCC referenciado. Se pedirá la selección de una de las dos siguientes posibilidades: TRUE o FALSE, es decir, para simular si se pudo o no activar el nivel físico

observaciones:

Capítulo 7: Entorno de Pruebas en Sistema Empotrado. Explicación del Software

7.1 GENERAL

Se creó un nuevo fichero, "n3_sim.c", para la simulación de una entidad de capa o nivel 3 teniendo encuenta únicamente los intercambios de señales (signals) y mensajes que se producirían entre esta entidad de nivel 3 y una entidad de Capa de Enlace de Datos. Es decir, se implementa únicamente la parte de la entidad de nivel 3 que se comunica con una entidad de Capa de Enlace de Datos, pero no con la aplicación. Tampoco se implementa las recomendaciones concretas del CCITT para esta entidad de nivel 3.

El proceso creado a partir del fichero n3_sim.c nos permitirá simular una entidad de nivel 3 "de respuesta lenta" y así poder decidir la dimensión de los pipes de señales y de mensajes entre los procesos: entidad-proceso de Capa de Enlace de Datos del lado del Equipo Terminal y la entidad-proceso de capa o nivel 3. A partir de estos valores dimensionaremos también los tamaños de los pipes de señales y de mensajes entre los procesos de nivel 2 y nivel 3 del lado del Terminal de Red.

El objetivo principal de esta nueva implementación es comprobar de forma práctica el correcto funcionamiento de la implementación del protocolo LAP-D diseñado.

El proceso de simulación del nivel 3 se creará ejecutándose a partir de la función principal del fichero "n3_sim.c" (función "SimulacionN3main").

Las entradas o salidas de esta simulación del nivel 3 desde / hacia el terminal asíncrono se harán de la siguiente forma:

- **La salida de la función "printf()" se mostrará en la pantalla del terminal asíncrono (PC).**
- **La entrada a la función "scanf()" se leerá del teclado del terminal asíncrono.**

La comunicación entre el proceso-entidad de nivel 3 y el terminal asíncrono se realizará a través del puerto serie RS-232 (entre el terminal y el MC68302).

En la figura 7.1 podemos observar un esquema de cómo se realizaría esta simulación.

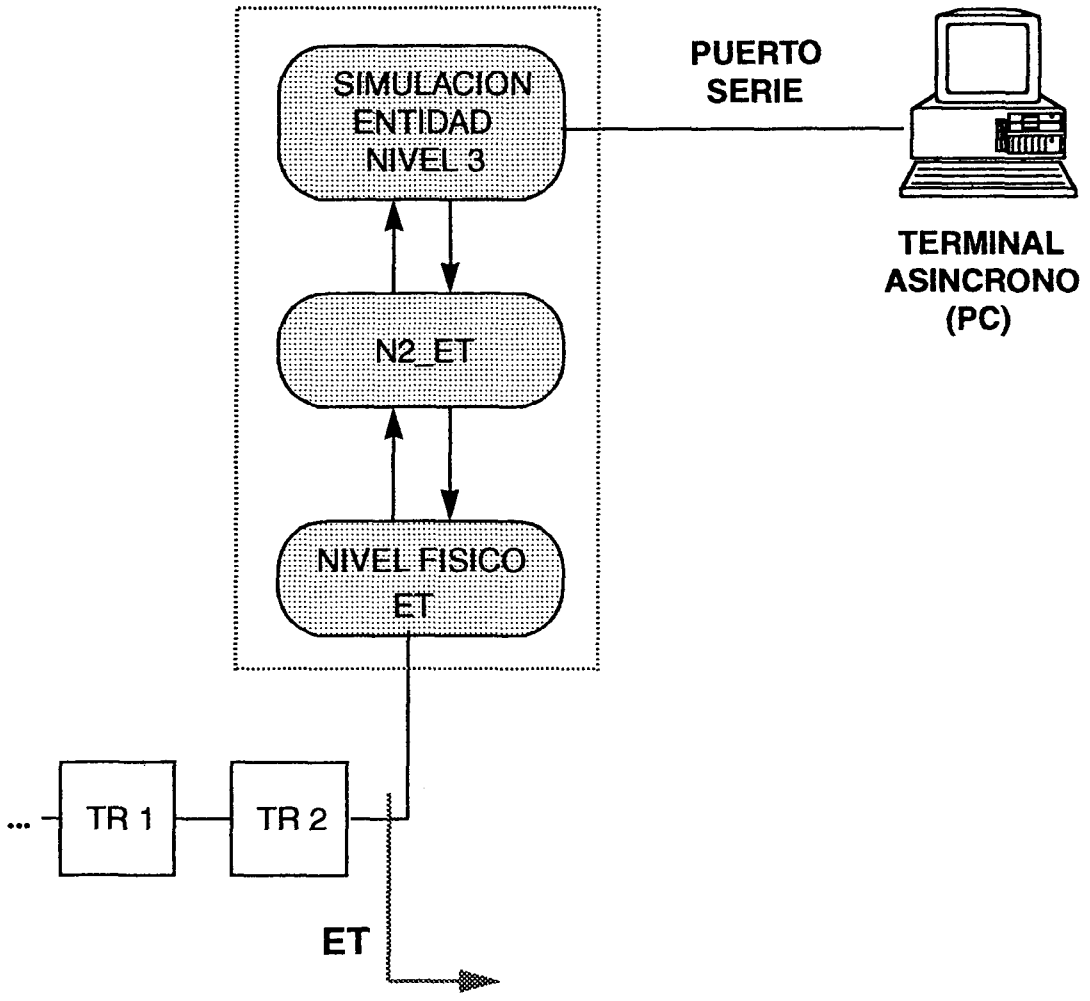


Figura 7.1 Esquema de la simulación del nivel 3 del lado ET en el Sistema Empotrado

En el menú de la simulación podemos seleccionar las siguientes opciones:

- **Leer señal del pipe de señaless (señales procedentes de la entidad-proceso de Capa de Enlace de Datos)**
- **Leer del pipe de lectura los mensajes escritos o enviados por la entidad de Capa de Enlace de Datos**
- **Establecer la conexión de Capa de Enlace de Datos**
- **Liberar la conexión de Capa de Enlace de Datos**
- **Enviar información sin confirmación (información en un mensaje asociado a la señal ED_UNIDAD_DATOS_REQ)**
- **Enviar información con confirmación (información en un mensaje asociado al señal ED_DATOS_REQ)**

7.2 DESCRIPCION DE FUNCIONES

Este fichero está compuesto únicamente por tres funciones: "SimulacionN3main" (función principal a partir de la cual se creará y ejecutará el proceso), "PresentaPantalla" (visualiza el menú de opciones de esta simulación) y "FormarMensaje23" que es la misma función utilizada en los ficheros n2_et.c y n2_tr.c (comentada ya anteriormente) para asignar a los campos de la estructura mensaje23 los valores que se quieren enviar a la entidad de Capa de Enlace de Datos.

A continuación se describen las dos primeras funciones:

NOMBRE: SimulacionN3main

definición:

void SimulacionN3main( )

parámetros de entrada:

UBYTE identificadorN2 Valor del identificador de la entidad de

Capa de Enlace de Datos

UWORD pipeEscrituraN3N2 Identificador del pipe de comunicación entre las entidades de capa 3 y capa 2. La entidad de capa 3 escribe sobre este pipe los mensajes dirigidos a la entidad de capa de Enlace de Datos, que posteriormente ésta leerá.

UWORD pipeLecturaN3N2 Identificador del pipe de comunicaciones entre las entidades de capa 3 y capa 2. La entidad de capa 3 leerá de este pipe los mensajes enviados por la entidad de capa de Enlace de Datos

parámetros de salida:

código devuelto:

descripción:

Esta función, que es la principal del fichero n3_sim.c, consiste en la implementación de la entidad de capa 3 vista desde la capa 2, es decir, realiza las tareas necesarias de interfaz para con la entidad de Capa de Enlace de Datos. El proceso que representa la entidad de capa 3 simulada se creará a partir de esta función, ejecutando esta función

observaciones:

Esta función constituye la simulación de una entidad de capa 3 teniendo encuenta únicamente los intercambios de señales y mensajes que se producirán entre esta entidad de capa 3 y una entidad de Capa de Enlace de Datos

NOMBRE: PresentaPantalla

definición:

void PresentaPantalla( )

parámetros de entrada:

parámetros de salida:

código devuelto:

descripción:

Esta función realiza la presentación en la pantalla del menú de simulación. En este menú se puede seleccionar las siguientes opciones:

1.- Leer señal del pipe de señales (señales procedentes de la entidad-proceso de Capa de Enlace de Datos)

2.- Leer del pipe de lectura los mensajes escritos o enviados por la entidad de Capa de Enlace de Datos

3.- Establecer la conexión de Capa de Enlace de Datos

4.- Liberar la conexión de Capa de Enlace de Datos

5.- Enviar información sin confirmación (información en un mensaje asociado a la señal ED_UNIDAD_DATOS_REQ)

6.- Enviar información con confirmación (información en un mensaje asociado a la señal ED_DATOS_REQ)

observaciones:

Capítulo 8: Banco de Pruebas

8.1 OBJETO

En este capítulo se describe de forma global los resultados de las pruebas realizadas para cada uno de los procesos en el entorno MS-DOS y sobre el sistema empotrado.

El objeto de estas pruebas es realizar una serie de comprobaciones del correcto funcionamiento de los procesos implementados según los criterios de diseño seguidos.

La descripción de estas pruebas se realiza de una forma conceptual para destacar mejor las ideas generales, haciendo relevancia especial en las entradas y salidas a cada proceso.

Las entradas y salidas, en general, harán referencia a las señales o tramas recibidas, así como a las acciones, señales y/o tramas enviadas en concepto de salida. Así una entrada podrá ser una señal, o explícitamente, una trama recibida en el mensaje asociado a dicha señal. Una salida podrá ser una señal o trama que sea necesario enviar o determinadas acciones tomadas como consecuencia de la entrada.

8.2 PROCESOS-ENTIDADES DE CAPA DE ENLACE DE DATOS

8.2.1 Proceso-entidad de Capa de Enlace de Datos del lado del Equipo Terminal

8.2.1.1 Pruebas Establecimiento de la Conexión de Enlace de Datos

Entrada:

Se recibe la señal ED_ESTABLEC_REQ procedente de la entidad de capa 3

Salida:

Se indica a la entidad de gestión del lado del Equipo Terminal que inicie el procedimiento de asignación de IET enviándole la señal GED_ASIGNACION_IET_IND

Entrada:

Recibo como primera señal la señal ED_UNIDAD_DATOS_REQ

Salida:

Igual que en el caso anterior se indica a la entidad de gestión del lado del Equipo Terminal que inicie el procedimiento de asignación de IET enviándole la señal GED_ASIGNACION_IET_IND

Entrada:

Si recibimos más señales ED_UNIDAD_DATOS_REQ antes de recibir la respuesta de la entidad de gestión de asignando o no un IET

Salida:

Se indicará a la entidad de capa 3 que ya ha sido enviada la información correspondiente a esa señal

Entrada:

Se recibe la señal GED_ASIGNACION_IET_REQ como respuesta a la petición de un IET a la entidad de gestión

Salida:

Se enviará una trama SABME utilizando el IET recibido a la entidad de capa de Enlace de Datos par y se esperará una trama UA

Entrada:

En oposición al caso anterior, se recibe una señal GED_NO_DISPONIBLE_IET_REQ con la que la entidad de gestión de capa me indica que no hay ningún IET disponible para establecer la conexión de Enlace de Datos

Salida:

Se indica a la entidad de Capa 3 que la conexión de Enlace de Datos no se pudo establecer enviándole la señal ED_LIBERACION_IND

8.2.1.1.1 Establecimiento de la conexión pendiente

En esta situación se envió una trama SABME y se está esperando una trama UA

Entrada:

Se recibe una trama UA confirmando el establecimiento de la conexión de Enlace de Datos

Salida:

Se ha establecido la conexión de Enlace de Datos y se pasaría al estado "EnlaceEstablecido" y dentro de él al subestado "NoWaitingAck_RR"

Entrada:

En vez de recibir una trama UA, recibimos una trama SABME

Salida:

Enviamos una trama UA y se espera hasta que recibamos una trama UA

8.2.1.2 Pruebas liberación de la conexión de Enlace de Datos

Entrada:

Se recibe una señal ED_LIBERACION_REQ de la entidad de Capa 3

Salida:

Se libera el buffer de tramas de información pendientes de envio, se libera la estructura ventana si se está transmitiendo la trama I de la ventana. Se envía una trama DISC y por tanto la conexión de Enlace de Datos estará pendiente de liberación

Entrada:

Si se recibe una señal GED_SUPRESION_IET_REQ

Salida:

Se indica a la entidad de Capa 3 que la conexión de Enlace de Datos se ha liberado y se liberan los buffers de tramas UI, de tramas I y la estructura ventana si hay una trama I en la ventana pendiente de confirmación. En este caso se considera la conexión de Enlace de Datos liberada

Entrada:

No recibimos trama UA después de tres retransmisiones de la trama SABME separadas un segundo

Salida:

Se indica a la entidad de nivel 3 que el enlace se ha liberado

8.2.1.2.1 Liberación de la conexión pendiente

En esta situación se envió una trama DISC a la entidad de Capa de Enlace de Datos par y se está esperando una trama UA

Entrada:

Recibimos una trama UA o DM como respuesta a la trama SABME enviada

Salida:

Se confirma que la conexión de Enlace de Datos está liberada

Entrada:

En vez de recibir una trama UA o DM, recibimos una trama DISC

Salida:

Enviamos una trama UA y esperaremos a recibir una trama UA para considerar la conexión de Enlace de Datos liberada

Entrada:

No recibimos trama UA después de tres retransmisiones separadas un segundo de la trama DISC

Salida:

Se indica a la entidad de nivel 3 que el enlace se ha liberado

8.2.1.3 Pruebas de transferencia de información

Estas pruebas son comunes tanto para la entidad de Capa de Enlace de Datos del lado del Equipo Terminal como para la entidad de Capa de Enlace de Datos del lado del Terminal de Red.

Entrada:

Llega una señal ED_DATOS_REQ y podemos formar y enviar una trama I

Salida:

Enviamos la trama I y esperamos la llegada de una trama confirmando

Entrada:

Llega una señal ED_DATOS_REQ y no podemos enviar la información asociada formando una trama I

Salida:

Se guarda en el buffer de tramas I pendientes de envío para su posterior transmisión

Entrada:

Recibimos una trama I procedente de la entidad de Capa de Enlace de Datos par y no tenemos ninguna trama I en espera de transmisión o no podemos transmitir una trama I temporalmente por estar esperando confirmación de una trama I previamente enviada

Salida:

Enviamos la trama I a la entidad de Capa 3 indicándoselo con la señal ED_DATOS_IND y enviamos una trama RR confirmando

Entrada:

Recibimos una trama I procedente de la entidad de Capa de Enlace de Datos par y tenemos una trama I en espera de transmisión

Salida:

Enviamos dicha trama I confirmando la trama I recibida

Entrada:

Llega una trama REJ rechazando la trama I enviada

Salida:

Se reenvía la trama I

Entrada:

Si hemos enviado la trama I y no se ha recibido respuesta confirmando después de un segundo

Salida:

Se retransmite la trama I hasta tres veces, luego se libera la conexión de Enlace de Datos si todavía no llegó respuesta confirmando

A continuación se muestra una gráfica que representa el intercambio de tramas I tramas RR confirmando entre dos entidades de Capa de Enlace de Datos. Se indica también las variables asociadas de cada una de las entidades:

vs: variable de estado en transmisión

va: variable de estado de acuse de recibo

vr: variable de estado en recepción

También se indica los valores del número secuencial en transmisión (ns) y número secuencial en recepción (nr) de las tramas intercambiadas. En esta tabla se observa como van cambiando los valores de dichas variables.

Podemos observar como una trama I puede ser utilizada para enviar información y confirmar tramas recibidas simultáneamente; y una trama RR se utilizará para confirmar tramas recibidas.

Entidad de Capa de Enlace de Datos (lado A)			Trama enviada			Entidad de Capa de Enlace de Datos (lado B)		
vs	va	vr	tipo trama y sentido	ns	nr	vs	va	vr
0	0	0				0	0	0
1	0	0	I →	0	0	0	0	1
1	1	0	RR ←	0	1	0	0	1
2	1	0	I →	1	0	0	0	2
2	2	0	RR ←	0	2	0	0	2
3	2	0	I →	2	0	0	0	3
3	3	1	I ←	0	3	1	0	3
4	3	1	I →	3	1	1	1	4
4	4	2	I ←	1	4	2	1	4

8.2.2 Proceso-entidad de Capa de Enlace de Datos del lado del Terminal de Red

8.2.2.1 Pruebas establecimiento de la conexión

Entrada:

Recibo trama SABME procedente de la entidad de Capa de Enlace de Datos par

Salida:

Envío señal UA confirmando el establecimiento de la conexión de Enlace de Datos

8.2.2.2 Pruebas liberación de la conexión

Son análogas a las realizadas para el caso de la entidad de Capa de Enlace de Datos del lado del Equipo Terminal. La única diferencia es el siguiente caso:

Entrada:

El multiplexor nos indica la pérdida del enlace físico

Salida:

Se libera la conexión de Enlace de Datos evolucionando al estado "EnlaceLiberado" y se indica a la entidad de Capa 3 que se ha liberado la conexión de Enlace de Datos. Además se liberarán los buffers de tramas I y tramas UI y la estructura ventana si había alguna trama I pendiente de confirmación

8.2.2.3 Pruebas de transferencia de información

Las pruebas de transferencia de información para la entidad de Capa de Enlace de Datos del lado del Equipo Terminal y del lado del Terminal de Red son las mismas puesto que su funcionamiento es el mismo para este caso concreto de la transferencia de información.

8.3 PROCESOS-ENTIDADES DE GESTIÓN DE CAPA DE LA CAPA DE ENLACE DE DATOS

8.3.1 Proceso-entidad de gestión del lado del Equipo Terminal

8.3.1.1 Procedimiento de asignación de IET en el lado del ET

Entrada:

Llega una señal GED_ASIGNACION_IET_IND de la entidad proceso de Capa de Enlace de Datos solicitando que le sea asignado un IET para establecer una conexión de Enlace de Datos

Salida:

Se envía a la entidad de gestión par un mensaje de petición de identidad (PETICION_IDENTIDAD) y se activa el temporizador (1 sg) para esperar la respuesta

8.3.1.1.1 Identidad Asignada

Entrada:

En esta situación de espera de respuesta a la petición de identidad llega un mensaje IDENTIDAD_ASIGNADA asignando un IET a la conexión de Enlace de Datos

Salida:

Se envía un mensaje a la entidad de Capa de Enlace de Datos indicándole el IET asignado

8.3.1.1.2 Identidad Rechazada

Entrada:

En la situación de espera de respuesta a la petición de identidad llega un mensaje IDENTIDAD_RECHAZADA y termina la cuenta del temporizador

Salida:

Se incrementa el contador de transmisiones del mensaje de petición de identidad y se transmite otro mensaje de petición de identidad

Entrada:

En esta situación de espera de respuesta a la petición de identidad se realizan hasta N202 (3) transmisiones del mensaje petición de identidad sin recibir respuesta asignando identidad

Salida:

Se indicará a la entidad de Capa de Enlace de Datos que no hay ningún IET disponible para asignació con la señal GED_NO_DISPONIBLE_IET_REQ

8.3.1.2 Procedimiento de prueba de IET en el lado del ET

Entrada:

Llega un mensaje petición prueba de identidad de la entidad de gestión par

Salida:

Se envía un mensaje respuesta de prueba de identidad indicando el IET utilizado por la capa de Enlace de Datos en su conexión de Enlace de Datos

8.3.1.3 Procedimiento de supresión de IET en el lado del ET

Entrada:

Llega un mensaje de supresión de identidad (SUPRESION_IDENTIDAD) procedente de la entidad de gestión par indicando que la conexión de Enlace de Datos debe liberarse para dejar libre el IET utilizado

Salida:

Se envía a la entidad de Capa de Enlace de Datos la señal GED_SUPRESION_IET_REQ para que libere la conexión

8.3.1.4 Petición de liberación de memoria

Entrada:

Llega una señal GED_LIBERACION_MEMO_IND y en el mensaje asociado el puntero a la zona de memoria que debe liberarse

Salida:

Se libera la memoria correspondiente

Observaciones:

Puede llegar también cuando estemos esperando la respuesta a la petición de identidad y responderemos de igual forma (liberando la memoria correspondiente)

8.3.2 Proceso-entidad de gestión del lado del Terminal de Red

A continuación se describe las posibles situaciones dentro de los tres estados en los que se puede encontrar el proceso gestor del lado del Terminal de Red.

Contemplamos los tres estados ya mencionados: "Idle", "EsperaRespPruebaIdIETdeGrupo" y "EsperaRespPruebaIdIETconcreto".

Entrada:

En cualquier estado puede llegar una petición de liberación de memoria

Salida:

Se libera la zona de memoria a la que apunta el puntero recibido en el mensaje asociado

8.3.2.1 Estado "Idle"

Entrada:

Se recibe un mensaje de petición de identidad (PETICION_IDENTIDAD)

Salida:

Se asigna el primer IET disponible del rango 64 a 126 y se permanece en este estado "Idle"

Entrada:

Se recibe un mensaje de petición de identidad (PETICION_IDENTIDAD)

Salida:

En este caso no hay ningún IET disponible para asignación y por tanto se enviará un mensaje de petición de prueba de identidad (PETICION_PRUEBA_IDENTIDAD) para verificar todos los valores IET (indicando el IET de grupo, es decir, 127). Se evoluciona al estado "EsperaRespPruebaIdIETdeGrupo"

Entrada:

Estando en el estado "Idle" se recibe un mensaje de solicitud de verificación de

identidad

Salida:

Se pasa al estado "EsperaRespPruebaIdIETconcreto" y en dicho estado se enviará un mensaj de petición de prueba de identidad con el IET referenciado y se inicia el termporizador T201 (1 sg)

8.3.2.2 Estado "EsperaRespPruebaIdIETconcreto"

Entrada:

Se recibe un mensaje respuesta prueba de identidad

Salida:

Se incrementa la variable que cuenta el número de respuestas de prueba de identidad

Entrada:

Se recibe un mensaje de petición de identidad

Salida:

Como hay algún IET disponible, se responde con un mensaje de asignación de identidad

Entrada:

Se recibe un mensaje de petición de identidad

Salida:

Como no hay ningún IET disponible, se guardará en la cola de peticiones el número de referencia enviado en el mensaje de petición de identidad

Entrada:

Se termina la primera cuenta del temporizador

Salida:

En este caso, sólo ha llegado un mensaje respuesta a la prueba de identidad y por tando se termina la prueba de identidad. Como hay otra petición de prueba de identidad, se atenderá enviando un mensaje de petición de identidad y rearrancando el temporizador

Entrada:

Se termina la primera cuenta del temporizador y no ha llegado ninguna respuesta de prueba de identidad

Salida:

Se repetirá una vez la petición de prueba de identidad y se rearrancará el temporizador T201 (1 sg)

Entrada:

Si terminó la segunda cuenta del temporizador y no se recibió ninguna respuesta de prueba de identidad

Salida:

Se supone que el valor IET está libre y disponible para reasignación

Entrada:

Se recibió más de una respuesta de prueba de identidad durante la segunda cuenta del temporizador

Salida:

Se considera que existe múltiple asignación de IET y se enviará dos veces

seguidas un mensaje de supresión de identidad

Entrada:

Terminada la primera o segunda cuenta y hay peticiones de identidad pendientes

Salida:

Se enviará una petición de prueba de identidad con IET de grupo y se evoluciona al estado EsperandoRespPruebaIdIETdeGrupo

8.3.2.3 Estado "EsperaRespPruebaIdIETdeGrupo"

Entrada:

Llega una petición de identidad

Salida:

Se guarda en la cola de peticiones de identidad

Entrada:

Llega una petición de verificación de identidad

Salida:

Se guarda en la cola de verificaciones de identidad

Entrada:

Llega una respuesta a la petición de prueba de identidad y no ha llegado antes una respuesta haciendo referencia al mismo IET

Salida:

Se anota ese IET como asignado

Entrada:

Llega una respuesta a la petición de prueba de identidad con un indicador de acción que hace referencia a un IET anotado previamente como asignado

Salida:

Se envía dos veces consecutivas el mensaje supresión de identidad a la entidad par de gestión y se indica a la entidad de capa de Enlace de Datos que utilice ese IET que libere la conexión enviándole la señal GED_SUPRESION_IET_REQ

Entrada:

Si no se recibe ninguna respuesta de prueba de identidad dentro del primer periodo T201 (1 sg)

Salida:

Se repetirá el envío del mensaje de petición de prueba de identidad con el IET de grupo

Entrada:

Al terminar el primer o segundo periodo de temporización (porque ha llegado más de una respuesta de prueba de identidad), y hay alguna petición de identidad pendiente

Salida:

Se responde a cada una de ellas asignándole un IET

Entrada:

Si hay alguna petición de identidad pendiente en el caso anterior y no se puede asignar más IETs

Salida:

Se envía una petición de prueba de identidad con IET de grupo y se libera los buffers de peticiones de identidad y de peticiones de verificación de identidad pendientes

8.4 PROCESO MULTIPLEXOR DEL LADO DEL TERMINAL DE RED

A continuación se describe las entradas y sus salidas de cada una de las situaciones principales en las que puede encontrarse el proceso multiplexor en su ejecución. Diferenciaremos los grupos de posibles situaciones tal y como se indica.

8.4.1 Llegada identificación entidad-proceso de Capa de Enlace de Datos

Entrada:

Mensaje MUX_TR_ID_PROCESO_REQ.

Al crearse las entidades-procesos de Capa de Enlace de Datos, éstas envían un primer mensaje al proceso multiplexor indicándole su identificador de proceso, el pipe de donde leerán los mensajes escritos por el proceso multiplexor y si son o no la entidad-proceso master

Salida:

Esta entrada se utiliza sólo para enviar información de identificación y no se genera ningún tipo de acción

8.4.2 Tratamiento de tramas recibidas

Entrada:

Se recibe una trama errónea

Salida:

El proceso multiplexor liberará la memoria correspondiente ignorando la trama

Entrada:

Se indica un error de desactivación del nivel físico al leer la supuesta trama recibida

Salida:

Se indicará a todas las entidades-procesos de Capa de Enlace de Datos que tengan una conexión establecida con sus respectivos remotos, que se ha perdido el enlace físico y por tanto deberán liberar esa conexión de Enlace de Datos

Entrada:

Se recibe una trama de información no numerada (trama UI) o una trama de información no numerada destinada a la entidad de gestión

Salida:

Se envía un mensaje al proceso-entidad master conteniendo dicha trama UI recibida del nivel físico

Entrada:

Se recibe una trama SABME con un Identificador de Equipo Terminal (IET) en su campo de dirección que no se utiliza en ninguna de las conexiones de Enlace de Datos que hay establecidas en ese determinado momento

Salida:

Se envía a una entidad-proceso de Capa de Enlace de Datos que no tenga una conexión de Enlace de Datos establecida o si todas las entidades-procesos de Capa de Enlace de Datos tienen una respectiva conexión de enlace establecida, se libera la trama SABME igonorándola

Entrada:

Se recibe una trama SABME con un IET que está siendo utilizado para identificar una de las conexiones de Enlace de Datos establecidas en ese momento (entre una entidad-proceso de Capa de Enlace de Datos y su correspondiente entidad par)

Salida:

Se envía dicha trama SABME a la entidad-proceso que está utilizando ese IET para identificar la conexión de Enlace de Datos

Entrada:

Se recibe cualquier tipo de trama soportada por esta implementación del protocolo LAP-D que no sea de los tipos SABME o información no numerada

Salida:

Se envía a la entidad-proceso que esté utilizando el IET de la trama recibida para identificar la conexión de Enlace de Datos mantenida con su entidad par

Entrada:

Se recibe una trama no soportada por esta implementación del protocolo LAP-D

Salida:

Se descarta liberando la memoria correspondiente

8.4.3 Indicación trama enviada

Entrada:

Se recibe la señal FI_DATOS_ENVIADOS indicando que el nivel físico ya envió la primera trama que tenía pendiente

Salida:

Se indica a la entidad-proceso que emitió dicha trama que el nivel físico ya la ha enviado (enviando la señal MUX_TR_DATOS_ENVIADOS_IND) y de esta forma se pueda liberar la memoria correspondiente. Además si se puede enviar la primera trama en espera a ser enviada, ésta se guarda en el buffer de transmisión

8.4.4 Indicación liberación de la conexión de Enlace de Datos

Entrada:

Se recibe la señal MUX_TR_ENLACE_LIBERADO_REQ indicando la desconexión de la conexión de Enlace de Datos que mantenía una determinada entidad-proceso de Capa de Enlace de Datos con su par

Salida:

Se toma referencia de esa entidad-proceso que se encontrará en el estado IET no asignado (IETnoAsignado) para un posible envío de trama SABME (con un nuevo IET) posterior

8.4.5 Indicación trama a enviar

Entrada:

Llega la señal MUX_TR_DATOS_REQ indicando que una determinada entidad-proceso de Capa de Enlace de Datos tiene alguna trama para transmitir al nivel físico

Salida:

Si no hay ninguna trama en el buffer de tramas en espera de transmisión, se intentará transmitir. Si se pudo transmitir se guarda en el buffer de transmisión. Si no se pudo transmitir o hay alguna trama en espera de ser transmitida, se guardará en el buffer de tramas en espera de ser transmitidas

ANEXOS

```
/*****************************************************************************/
/*                                INICIO.H                                   */
/*****************************************************************************/


/*---------- Constantes globales: ------------------------------------------*/

#define NNO_ERROR       (UWORD)  0  /* Servicio ejecutado correctamente */
#define NID_PUERTO_NOK  (UWORD)  1  /* Indice fuera de rango */
#define NNO_PROCESO     (UWORD)  2  /* No hay mas procesos */



/*---------- Principio y fin de la memoria: --------------------------------*/

#define NIBLOQUE1  0x800000   /* Inicio del bloque 1 de memoria */
#define NFBLOQUE1  0X80ffff   /* Fin del bloque 1 de memoria */

#define NIBLOQUE2  0x400000   /*  Inicio del bloque 2 de memoria */
#define NFBLOQUE2  0X41ffff   /*  Fin del bloque 2 de memoria */



/*---------- Estructuras de datos utilizadas: ------------------------------*/

#define NDATOS_INI        5    /*  Numero de dtos iniciales para el proceso */
#define NMAX_PROCPUERTO   3    /*  Número máximo de procesos por puerto     */

typedef struct
  {
  VOID  (*arranque)();        /* Direccion de comienzo      */
  UWORD tDatos;               /* Numero de bytes para datos */
  UWORD tPila;                /* Numero de bytes para pilas */
  ULONG datos[NDATOS_INI];    /* Datos iniciales            */
  UBYTE ocupado;              /* TRUE -> está utilizada     */
  } NPROCESOS;                /* Los datos sobre los procesos son pasados al
                                 kernel en esta estructura */


/*---------- Servicio de Inicialización: -----------------------------------*/

/*
typedef NPROCESOS *PNPROCESOS;

VOID  NMain();
UWORD NObtieneProceso(UWORD,PNPROCESOS);
UWORD NProcesosPuerto(UWORD,PUWORD,PUWORD);
*/

VOID  NMain();
UWORD NObtieneProceso();
UWORD NProcesosPuerto();
UWORD NBorraprocesosPuerto();
UWORD NCreaProcesosPuerto();
VOID  NLeeMemoria();
```

MODULO: KERNEL

DESCRIPCION:

Contiene todos los servicios del kernel. Controla la inicialización del hardware. Lleva el control de procesos, de pipes, de contadores, de memoria dinámica y del reloj en tiempo real.

FICHEROS:

fuente: kernel.c

definiciones propias: kernel.hp

definiciones globales: kernel.h

librerías generadas:

librerías utilizadas:

Se linka junto con los procesos 02, 03, 04, 06, 07 y 11.

CONSTANTES GLOBALES

NOMBRE: KNO_ERROR

valor: 0

descripción: Código devuelto cuando no hay error

NOMBRE: KID_PIPE_NOK

valor: 1

descripción: Devuelto cuando no existe el pipe referenciado

NOMBRE: KINTERRUP

valor: 2

descripción: Devuelto cuando el proceso que ha llamado tiene una indicación de interrupción pendiente.

NOMBRE: KNOFREE

valor: 3

descripción: Valor devuelto cuando se accede a un recurso en la modalidad no espera (KNOWAIT) y el recurso no está disponible.

NOMBRE: KNO_MEMORIA

valor: 4

descripción: Código devuelto por el servicio de solicitud de memoria cuando no hay la cantidad de memoria solicitada disponible.

NOMBRE: KID_CONT_NOK

valor: 5

descripción: Código devuelto por el servicio de contadores cuando no existe el identificador referenciado.

NOMBRE: KID_PROC_NOK

valor: 6

descripción: Código devuelto por el servicio de control de procesos para indicar que l identificador de proceso recibido no es válido.

NOMBRE: KCONTANDO

valor: 7

descripción: Valor devuelto por el servicio de consulta de contador para indicar que el contador no ha terminado de contar.

NOMBRE: KFINALIZADO

valor: 8

descripción: Valor devuelto por el servicio de consulta de contador para indicar que el contador ha terminado de contar.

NOMBRE: KWAIT

valor: 1

descripción: Parámetro que indica que el proceso quiere esperar si el recurso no está libre (lectura sobre un pipe vacio ...).

NOMBRE: KNOWAIT

valor: 0

descripción: Parámetro que indica que el proceso no quiere esperar cuando el recurso no está libre.

NOMBRE: KSALIDA

valor: 0x0100

descripción: En el servicio de espera multievento se puede definir en lugar de un pipe un puerto de E/S. Para ello en vez de pasar el identificador de pipe se pasa el identificador de puerto OR con este valor.

NOMBRE: KNO_UTIL

valor: 0XFFFF

descripción: En el servicio de espera multievento se utiliza este valor para indicar que dicho evento no se debe de tener en cuenta.

NOMBRE:
KME_1PS
KME_2PS
KME_1PE
KME_2PE
KME_C

valor:
0x80
0x81
0x82
0x83
0x84 (respectivamente).

descripción: En el servicio de espera multievento se devuelve uno de estos códigos para indicar el evento producido. Los eventos son los siguientes:

KME_1PS primer pipe (o puerto) para escritura
KME_2PS segundo " " " "
KME_1PE primer " " " lectura
KME_2PE segundo " " " "
KME_C contador

ESTRUCTURAS DE DATOS

NOMBRE: KESTADO_PIPE

definición:

```
typedef struct
 {
 UBYTE          numProcesosLectura;
 UBYTE          numProcesosEscritura;
 UWORD          bytesOcupados;
 UBYTE          bytesLibres;
 } KESTADO_PIPE;
```

descripción:

En el servicio de lectura del estado de un pipe devuelve los datos del pipe:

numProcesosLectura: número de los procesos que están durmiendo en espera de una lectura.

numProcesosEscritura: número de los procesos que están durmiendo en espera de escribir.

bytesOcupados: número de bytes que están ocupados.

bytesLibres: número de bytes que están libres.

NOMBRE: KHORA_FECHA

definición:

```
typedef struct
 {
 UBYTEanno;
 UBYTEmes;
 UBYTEdia;
 UBYTEhora;
 UBYTEminuto;
 UBYTEsegundo;
 } KHORA_FECHA;
```

descripción:

Contiene la fecha y hora en tiempo real del sistema. Para los servicios de puesta en hora y consulta se utiliza esta estructura como parámetro.

El año viene dado a partir de 1980. Es decir, si vale 1 es el año 1981, si vale 255 es el año 2235.

NOMBRE: KMULTIEVENT

definición:

```
typedef estruct
 {
 UWORD      pipeOut1;
 UWORD      bytesPipeOut1;
 UWORD      pipeOut2;
 UWORD      bytesPipeOut2;
 UWORD      pipeIn1;
 UWORD      bytesPipeIn1;
 UWORD      pipeIn2;
 UWORD      bytesPipeIn2;
 UWORD      contador;
 } KMULTIEVENT;
```

descripción:

Es la estructura que se utiliza como parámetro para el servicio de espera multievento.

Las cuatro primeras variables son los pipes por los que se está esperando para escribir y los bytes que se necesitan escribir.

Las cuatro siguientes son los pipes por los que se está esperando para leer y los bytes que se desean leer.

Si en ves de esperar un pipe se espera un puerto se rellena la variable (pipe[In I Out][1 I 2]) con el OR del indice del puerto y la constante KSALIDA. En este caso el número de bytes no se utiliza.

Los contadores son los índices de los contadores por los cuales se espera.

Si alguno de los eventos no se utiliza su variable se rellena con la constante KNO_UTIL.

SERVICIOS

NOMBRE: KSolicitaPipe

definición:

UWORD KSolicitaPipe(pipe)

parámetros entrada:

parámetros salida:

UWORD	*pipe	Indice pipe a utilizar

código devuelto:

KNO_ERROR	Todo correcto
KNO_PIPE	No quedan pipes libres

descripción:

Para solicitar un pipe

observaciones:

NOMBRE: KLiberaPipe

definición:

UWORD KLiberaPipe(pipe)

parámetros entrada:

UWORD	pipe	Indice del pipe a liberar

parámetros salida:

código devuelto:

descripción:

Libera el pipe referenciado si ya no es necesario

observaciones:

NOMBRE: KEscribePipe

definición:

UWORD KEscribePipe(pipe,datos,longitud,espera)

parámetros entrada:

UWORD	pipe	indice del pipe
VOID	*datos	buffer de los datos a escribir
UWORD	longitud	número de bytes a escribir
UWORD	espera	bandera de espera (KWAIT o KNOWAIT)

parámetros salida:

código devuelto:

KNO_ERROR	Todo correcto
KID_PIPE_NOK	Indice del pipe no válido
KINTERRUP	Recibida una interrupción para el proceso
KNOFREE	En modalidad no espera, no hay espacio suficiente en el pipe.

descripción:

La operación de escritura se realiza atómicamente. Es decir, en el pipe se guardan todos los bytes uno a continuación del otro.

Si se utiliza la modalidad KWAIT y no hay espacio suficiente, el proceso se duerme hasta que lo haya y entonces se realiza la escritura. Si está en modo KNOWAIT y no hay espacio se devuelve el código KNOFREE.

observaciones:

NOMBRE: KLeePipe

definición:

UWORD KLeePipe(pipe,datos,longitud,espera)

parámetros entrada:

UWORD	pipe	indice del pipe
UWORD	longitud	número de bytes a leer

UWORD	espera	bandera de espera (KWAIT o KNOWAIT)

parámetros salida:

VOID	*datos	buffer donde se escriben los datos

código devuelto:

KNO_ERROR	Todo correcto
KID_PIPE_NOK	Indice del pipe no válido
KINTERRUP	Recibida una interrupción para el proceso
KNOFREE	En modalidad no espera, no hay datos suficientes en el pipe.

descripción:

La operación de lectura se realiza atómicamente. Es decir, se leen del pipe todos los bytes uno a continuación del otro.

Si se utiliza la modalidad KWAIT y no hay datos suficientes, el proceso se duerme hasta que los haya y entonces se realiza la lectura. Si está en modo KNOWAIT y no hay datos suficientes se devuelve el código KNOFREE.

observaciones:

NOMBRE: KSignal

definición:

UWORD KSignal(proceso, signal)

parámetros entrada:

UBYTE	proceso	Proceso que recibirá la señal
UWORD	signal	Señal que recibirá el proceso

parámetros salida:

código devuelto:

KID_PROC_NOK	Identificador de proceso erroneo
KSIG_ERROR	Buffer de señales lleno
KNO_ERROR	Todo correcto

descripción:

Envía una señal al proceso referenciado.

Si el proceso está esperando por algo (pipe, contador o en KWait) es despertado con el código KINTERRUP

El valor de la señal se guarda en una cola FIFO

observaciones:

NOMBRE: KLeeSignal

definición:

UWORD KLeeSignal(signal)

parámetros entrada:

parámetros salida:

UWORD	*signal	Señal recibida por el proceso

código devuelto:

KSIG_ERROR	Si no hay señal esperando
KNO_ERROR	Devuelta la primera señal del buffer

descripción:

Se lee la primera señal del buffer del proceso que llama a la función

observaciones:

NOMBRE: KWait

definición:

UWORD KWait(void)

parámetros entrada:

parámetros salida:

código devuelto:

KINTERRUP	El proceso se ha despertado por una señal

descripción:

El proceso queda dormido hasta que llegue una señal. Si cuando se llama al servicio, el buffer de señales no está vacío se retorna inmediatamente.

observaciones:

NOMBRE: KCreaProceso

definición:

UWORD KCreaProceso(funcion, datos, pila, numParam, param)

parámetros entrada:

(VOID)	(*funcion)()	Puntero de la función de arranque
UWORD	datos	Espacio de su segmento de datos
UWORD	pila	Espacio pars su pila
UWORD	numParam	Número de parámetros de "funcion"

parámetros salida:

código devuelto:

NILPROC	No se puede crear el proceso
otro valor	Identificador del proceso creado

descripción:

Se crea un nuevo proceso que empieza su ejecución en la función "funcion".
Esta función recibe un número de parámetros determinado por numParam.
El tamaño del espacio de datos del proceso se tiene que determinar através del espacio que ocupan sus variables estáticas.
El tamaño de la pila se define de tal forma que sea imposible su desbordamiento

observaciones:

NOMBRE: KPideMemoria

definición:

UWORD KPideMemoria(longitud,descriptor)

parámetros entrada:

UWORD	longitud	número de bytes a pedir

parámetros salida:

UWORD	**descriptor	puntero a la memoria solicitada

código devuelto:

KNO_ERROR	Todo correcto
KNO_MEMORIA	No hay suficiente memoria

descripción:

Los bloques de memoria siempre empiezan en posiciones pares.

observaciones:

NOMBRE: KNice

definición:

UWORD KNice(void)

parámetros entrada:

parámetros salida:

código devuelto:

descripción:

Permite que otro proceso use la CPU. No se duerme sino que se encola el último en la cola de procesos activos.

observaciones:

NOMBRE: KLiberaMemoria

definición:

UWORD KLiberaMemoria(descriptor)

parámetros entrada:

UWORD	*descriptor	bloque de memoria a liberar

parámetros salida:

código devuelto:

KNO_ERROR	Todo correcto

descripción:

Los bloques liberados pueden volver a ser usados.

observaciones:

El servicio se comporta de forma impredecible si el descriptor recibido no fue inicializado por el servicio KPideMemoria.

NOMBRE: KSolicitaContador

definición:

UWORD KSolicitaContador(indiceContador)

parámetros entrada:

parámetros salida:

UWORD *indiceContador	Indice del contador solicitado

código devuelto:

KNO_ERROR	Asignado contador
KNOFREE	No hay contador libre

descripción:

Devuelve el índice de un contador que puede ser utilizado en exclusiva. Cualquier contador que se desee utilizar tiene que haber sido solicitado con este servicio.

NOMBRE: KLiberaContador

definición:

VOID KLiberaContador(indiceContador)

parámetros entrada:

UWORD indiceContador	Contador a liberar

parámetros salida:

código devuelto:

descripción:

Cuando un contador obtenido con KSolicitaContador no se desea utilizar más se libera con este servicio.

observaciones:

Si se intenta liberar un contador no asignado o con índice fuera de rango no se hace nada.

NOMBRE: KActivaContador

definición:

UWORD KActivaContador(contador,tiempo)

parámetros entrada:

UWORD	contador	Indice del contador a activar
UWORD	tiempo	Tiempo en segundos para el contador

parámetros salida:

código devuelto:

KNO_ERROR	Todo correcto
KID_CONT_NOK	El indice del contador no es correcto

descripción:

Se activa el contador indicado para que cuente el número de segundos que se especifica en "tiempo". Durante ese "tiempo" las consultas al contador devuelven KCONTANDO, cuando termina devuelven KFINALIZADO.

Si se utiliza el servicio KEsperaContador el proceso queda dormido hasta que pase el tiempo.

observaciones:

NOMBRE: KConsultaContador

definición:

UWORD KConsultaContador(contador)

parámetros entrada:

UWORD	contador	Indice del contador

parámetros salida:

código devuelto:

KID_CONT_NOK	Indice del contador no valido
KCONTANDO	El contador no ha llegado a cero
KFINALIZADO	El contador ya llegó a cero

descripción:

Miestran el contador sea distinto de cero se devuelve KCONTADO, desde que esté a cero deja de contar y se devuelve KFINALIZADO.

observaciones:

Si se consulta el estado de un contador no inicializado se devuelve KFINALIZADO.

NOMBRE: KEsperaContador

definición:

UWORD KEsperaContador(contador)

parámetros entrada:

UWORD	contador	Indice del contador

parámetros salida:

código devuelto:

KNO_ERROR	Todo correcto
KID_CONT_NOK	El indice del contador no es correcto

descripción:

Si el contador está a cero se devuelve el control inmediatamente con el código KNO_ERROR. Si está contado se durme el proceso hasta que termine el contador y entonces se activa el proceso con el código KNO_ERROR.

observaciones:

NOMBRE: KConsultaHora

definición:

UWORD KConsultahora(hora)

parámetros entrada:

parámetros salida:

KHORA_FECHA	*hora	fecha y horas actuales

código devuelto:

KNO_ERROR

descripción:

Devuelve la hora y la fecha de tiempo real. El año lo da referenciado a 1980.

observaciones:

NOMBRE: KPuestaHora

definición:

UWORD KPuestaHora(hora)

parámetros entrada:

KHORA_FECHA *hora datos actuales de la fecha y hora

parámetros salida:

código devuelto:

KNO_ERROR

descripción:

Se actualizan los datos de la fecha y la hora. Si alguno de los datos está fuera de rango no se realiza ninguna actualización.

observaciones:

El año está entre 0 y 255.
El mes entre 1 y 12
El día entre 1 y 31
La hora entre 0 y 23
El minuto entre 0 y 59

NOMBRE: KEsperaMulti

definición:

UWORD KEsperaMulti(multi)

parámetros entrada:

KMULTIEVENT *multi datos de los eventos esperados

parámetros salida:

código devuelto:

KINTERRUP	Recibida una interrupción para el proceso
KME_1PS	Espacio en el primer pipe
KME_2PS	Idem en el segundo
KME_1PE	Datos suficientes en el primer pipe
KME_2PE	Idem en el segundo
KME_1C	Contador terminado de contar

descripción:

Este servicio espera a que alguno de los eventos especificados (ver estructura de datos KMULTIEVENT) se cumpla.

observaciones:

Para el caso de los pipes o puertos la operación de entrada salida no se ejecuta. Por lo tanto se tiene que llamar a continuación al servicio correspondiente. No se garantiza que cuando se llame al servicio todavia se cumpla la condición de salida de la espera multievento.

Si se espera por la entrada de un puerto se devuelve el código indicando una entrada cuando la patilla dcd de dicho puerto cambia de estado.

NOMBRE: KInterrupcion

definición:

UWORD KInterrupcion(proceso)

parámetros entrada:

UWORD	proceso	Indice del proceso a interrumpir

parámetros salida:

código devuelto:

KNO_ERROR	Todo correcto
KID_PROC_NOK	El indice del proceso no es correcto

descripción:

Si el proceso a interrumpir está en una cola de lectura/escritura de pipe se despierta con el indicador de interrupción.

Si no, se guarda la indicación de interrupción y en la primera solicitud del proceso de una lectura/escritura en pipe se le devuelve el código de interrupción.

observaciones:

NOMBRE: KDeshabilitarProceso

definición:

UWORD KDeshabilitarProceso(proceso)

parámetros entrada:

UWORD	proceso	Indice del proceso a dormir

parámetros salida:

código devuelto:

KNO_ERROR	Todo correcto
KID_PROC_NOK	El indice del proceso no es correcto

descripción:

Se encola el proceso referenciado en una cola especial en espera que se le despierte con el servicio KHabilitarProceso.

observaciones:

El proceso referenciado puede ser el mismo que ejecuta el servicio.

NOMBRE: KHabilitarProceso

definición:

UWORD KHabilitarProceso(proceso)

parámetros entrada:

UWORD	proceso	Indice del proceso a habilitar

parámetros salida:

código devuelto:

KNO_ERROR	Todo correcto
KID_PROC_NOK	El indice del proceso no es correcto

descripción:

Se saca el proceso de la cola de espera y se pone en la cola de procesos activos.

observaciones:

Si el proceso referenciado no se encuentra en la cola de espera no se realiza ninguna acción.

NOMBRE: KComparaContador

definición:

UWORD KComparaContador(cont1,cont2)

parámetros entrada:

UWORD	cont1	Indice del primer contador
UWORD	cont2	Indice del segundo contador

parámetros salida:

código devuelto:

KIGUALES	Son iguales
K1MENOR	El primero tiene menos tiempo
K1MAYOR	El primero tiene más tiempo

descripción:

Compara dos contadores según el tiempo que les falta por contar.

observaciones:

IMPLEMENTACION

Los servicios tienen que ser utilizados por cualquier proceso. Esto incluye a los procesos compilados y linkados por separado.

Como estas llamadas son a dirección absoluta y a un código que puede haber sido compilado y linkado por separado se implementan a través de una tabla de punteros a función.

Esta tabla se implementa en la tabla de vectores de excepción. Se realiza la llamada mediante indirección. Y se implementan con una macro.

Estas macros se encuentran en el fichero "KERNEL.H".

Fichero kernel.h:

```
/*****************************************************************************/
/*                              KERNEL.H                                     */
/*        Este fichero contiene las definiciones globales que pueden ser     */
/*                    manejadas por otros procedimientos                     */
/*****************************************************************************/

#ifndef KERNEL_H
#define KERNEL_H

#ifdef UNIX
#define asm(a) Asm(a)
#endif


/*----------------- Constantes globales: ------------------------------------*/

#define SYSERROR       (UWORD) 100    /* critical error */
#define KNO_ERROR      (UWORD) 0      /* No hay error */
#define KID_PIPE_NOK   (UWORD) 1      /* no existe el pipe referenciado */
#define KINTERRUP      (UWORD) 2      /* indicacion de interrupcion pendiente */
#define KNOFREE        (UWORD) 3      /* recurso no disponible */
#define KNO_MEMORIA    (UWORD) 4      /* no hay memoria disponible */
#define KID_CONT_NOK   (UWORD) 5      /* no existe el contador referenciado */
#define KID_PROC_NOK   (UWORD) 6      /* no existe el proceso referenciado */
#define KCONTANDO      (UWORD) 7      /* no se ha terminado la cuenta */
#define KFINALIZADO    (UWORD) 8      /* se ha terminado la cuenta */
#define KIGUALES       (UWORD) 9      /* comparación de contadores */
#define K1MAYOR        (UWORD) 10
#define K1MENOR        (UWORD) 11
#define KSIG_ERROR     (UWORD) 12     /* no se puede enviar la señal */
#define KNO_PIPE       (UWORD) 13     /* no quedan pipes libres */
#define KWAIT          (UWORD) 1      /* el proceso espera si el recurso no esta
                                         disponible */
#define KNOWAIT        (UWORD) 0      /* el proceso no espera */
```

```
/*--------------- ESTRUCTURAS DE DATOS: ------------------------------------*/

/* Tipo que se utiliza para indicar el estado de un pipe */
typedef struct
  {
  UBYTE numProcesosLectura;        /* Procesos en espera de lectura */
  UBYTE numProcesosEscritura;      /* Procesos en espera de escritura */
  UWORD bytesOcupados;             /* Bytes ocupados */
  UBYTE bytesLibres;               /* Bytes libres */
  }  KESTADO_PIPE;

/* Tipo que se utiliza para indicar HORA y FECHA */
typedef struct
  {
  UBYTE anno;
  UBYTE mes;
  UBYTE dia;
  UBYTE hora;
  UBYTE minuto;
  UBYTE segundo;
  } KHORA_FECHA;


/*  Macros para desactivar y activar las interrupciones */

#define  CLI1  asm("  MOVE.W SR,-(SP)");\
               asm("  ORI.W #$100,SR")

#define  CLI2  asm("  MOVE.W SR,-(SP)");\
               asm("  ORI.W #$200,SR")

#define  CLI3  asm("  MOVE.W SR,-(SP)");\
               asm("  ORI.W #$300,SR")

#define  CLI4  asm("  MOVE.W SR,-(SP)");\
               asm("  ORI.W #$400,SR")

#define  CLI5  asm("  MOVE.W SR,-(SP)");\
               asm("  ORI.W #$500,SR")

#define  CLI6  asm("  MOVE.W SR,-(SP)");\
               asm("  ORI.W #$600,SR")

#define  CLI7  asm("  MOVE.W SR,-(SP)");\
               asm("  ORI.W #$700,SR")

#define  STI asm("  MOVE.W (SP)+,SR")


/*------------------------- Servicios del Kernel ---------------------------*/
#ifdef UNIX
UWORD KSolicitaPipe(UWORD *);
UWORD KLiberaPipe(UWORD);
UWORD KEstadoPipe(UWORD, KESTADO_PIPE *);
UWORD KEscribePipe(UWORD, VOID *, UWORD, UWORD);
UWORD KLeePipe(UWORD, VOID *, UWORD, UWORD);
UWORD KPideMemoria(UWORD, UWORD **);
UWORD KLiberaMemoria(UWORD *);
UWORD KActivaContador(UWORD, UWORD);
UWORD KConsultaContador(UWORD);
UWORD KEsperaContador(UWORD);
UWORD KSolicitaContador(UWORD *);
UWORD KLiberaContador(UWORD);
UWORD KComparaContador(UWORD, UWORD);
UWORD KSignal(UBYTE, UWORD);
UWORD KLeeSignal(UWORD *);
UWORD KWait(VOID);
UBYTE KCreaProceso(VOID (*)(), UWORD, UWORD, UWORD, ULONG []);
```

```
VOID  KMain(VOID);
VOID  KNice(VOID);
UWORD KConsultaHora(KHORA_FECHA *);
UWORD KPuestaHora(KHORA_FECHA *);
UBYTE KIDProceso(VOID);
VOID  Asm(BYTE *);   /* para eliminar el ensamblador */

#else

UWORD KSolicitaPipe();
UWORD KLiberaPipe();
UWORD KEstadoPipe();
UWORD KEscribePipe();
UWORD KLeePipe();
UWORD KPideMemoria();
UWORD KLiberaMemoria();
UWORD KActivaContador();
UWORD KConsultaContador();
UWORD KEsperaContador();
UWORD KSolicitaContador();
UWORD KLiberaContador();
UWORD KComparaContador();
UWORD KSignal();
UWORD KLeeSignal();
UWORD KWait();
UBYTE KCreaProceso();
VOID  KMain();
VOID  KNice();
UWORD KConsultaHora();
UWORD KPuestaHora();
UBYTE KIDProceso();
#endif

#endif /*KERNEL_H*/
```

```
/*****************************************************************************/
/*                              NIVEL1.H                                     */
/*         Definiciones para usar los servicios del nivel físico             */
/*****************************************************************************/
/*#include "..\kernel\basico.h"*/

/* puertos conectados a cada uno de los scc */
#define   CANAL_D         1
#define   CANAL_B1        2
#define   CANAL_B2        3
#define   CANAL_DIRECTO   4
#define   CANAL_NOCONECT  5

/* tipo de protocolo soportado por cada scc */
#define   PROTOCOLO_HDLC  1
#define   PROTOCOLO_UART  2
#define   PROTOCOLO_TRANS 3

/* baudios para las conexiones NMSI (no IDL)   */
#define   B150            1
#define   B300            2
#define   B600            3
#define   B1200           4
#define   B2400           5
#define   B4800           6
#define   B9600           7
#define   B19200          8
#define   B38400          9

/* definiciones para los bits de stop de la UART */
#define   STOP_1          1
#define   STOP_15         2
#define   STOP_2          3

/* definiciones para la paridad de la UART */
#define   PARIDAD_PAR     1
#define   PARIDAD_IMPAR   2
#define   PARIDAD_NO      3

/* definiciones de los errores de los scc */
#define   ERROR_OVERRUN    0x0001    /* 0000 0000 0000 0001 */
#define   ERROR_FRAMING    0x0002    /* 0000 0000 0000 0010 */
#define   ERROR_PARITY     0x0004    /* 0000 0000 0000 0100 */
#define   ERROR_BREAK      0x0008    /* 0000 0000 0000 1000 */
#define   ERROR_ABORT      0x0010    /* 0000 0000 0001 0000 */
#define   ERROR_NONOCTET   0x0020    /* 0000 0000 0010 0000 */
#define   ERROR_CRC        0x0040    /* 0000 0000 0100 0000 */
#define   ERROR_DESACT     0X0080    /* 0000 0000 1000 0000 */

typedef struct
        {
        UWORD scc1;         /* tipo de configuracion del puerto serie 1 */
        UWORD scc2;
        UWORD scc3;
        UBYTE mascaraBitsB1;    /* indica los bits utiles del canal B1 */
        UBYTE mascaraBitsB2;    /* indica los bits utiles del canal B2 */
        } CONF_NIVEL1;

typedef struct
        {
        UBYTE IDProceso; /* proceso dueño del scc */
        UWORD signal[3]; /* señales que envia el nivel 1 al proceso dueño */

        /* [0] enviada trama; [1] recibida; [2] error    */
        UWORD baudios;
        UWORD xonXof;      /* TRUE  si se usa Xon Xof    */
        UWORD bitStop;     /* valores definidos: 1, 1.5 y 2 */
        UWORD paridad;     /* valores definidos: par, impar, sin paridad, 1 y 0 */
        UWORD caracRecep; /* numero de caracteres en recepcion para interrumpir
*/
```

```
        } PARAM_UART;

typedef struct
        {
        UBYTE IDProceso;  /* proceso dueño del scc */
        UWORD signal[3];  /* señales que envia el nivel 1 al proceso dueño */

        /* [0] enviada trama; [1] recibida; [2] error    */
        UWORD caracRecep; /* numero MAXIMO de caracteres de un paquete */
        } PARAM_HDLC;

typedef struct
        {
        UBYTE IDProceso;  /* proceso dueño del scc */
        UWORD signal[3];  /* señales que envia el nivel 1 al proceso dueño */

        /* [0] enviada trama; [1] recibida; [2] error    */
        UWORD caracRecep; /* numero de caracteres en recepcion para un paquete */
        } PARAM_TRANS;

typedef union
        {
        PARAM_HDLC  hdlc;
        PARAM_UART  uart;
        PARAM_TRANS trans;
        } PARAM_SCC;


/*--------- PROTOTIPOS DE FUNCIONES QUE IMPLEMENTAN EL NIVEL FISICO: -------*/

void  Finicia68302( void);
UWORD FIniciaCP( CONF_NIVEL1 *param);
UWORD FIniciaSCC( UWORD scc, UWORD protocolo, PARAM_SCC *param);
void  FControlTransSCC( UWORD scc, UWORD ActivaRec, UWORD ActivaTrans,
                        UWORD tbreak);
UWORD FEnviaDatos( UWORD scc, void *datos1, UWORD longitud1,
                   void *datos2, UWORD longitud2);

UWORD FRecibeDatos( UWORD scc, void **datos, UWORD *longitud, UWORD *error);
UWORD FActivaSCC( UWORD scc);
```

```
/*****************************************************************************/
/*                              N2N3.H                                       */
/*        Fichero de definiciones compartidas entre los niveles 2 y 3        */
/*                                                                           */
/*              Indice de proceso = 0x0200 (Indica nivel 2)                  */
/*        Este indice estará en el byte de mayor peso de los signals         */
/*****************************************************************************/

#define INDICE_NIVEL2           (UWORD)(0x0200)


/*------------ Primitivas entre el nivel 2 y el 3: --------------------------*/
#define ED_ESTABLEC_REQ         (UWORD)(INDICE_NIVEL2+0x00)   /* NOTA */
#define ED_ESTABLEC_IND         (UWORD)(INDICE_NIVEL2+0x01)   /* NOTA */
#define ED_ESTABLEC_CONF        (UWORD)(INDICE_NIVEL2+0x02)   /* NOTA */
#define ED_LIBERACION_REQ       (UWORD)(INDICE_NIVEL2+0x03)   /* NOTA */
#define ED_LIBERACION_IND       (UWORD)(INDICE_NIVEL2+0x04)   /* NOTA */
#define ED_LIBERACION_CONF      (UWORD)(INDICE_NIVEL2+0x05)   /* NOTA */
#define ED_DATOS_REQ            (UWORD)(INDICE_NIVEL2+0x06)
#define ED_DATOS_IND            (UWORD)(INDICE_NIVEL2+0x07)
#define ED_UNIDAD_DATOS_REQ     (UWORD)(INDICE_NIVEL2+0x08)
#define ED_UNIDAD_DATOS_IND     (UWORD)(INDICE_NIVEL2+0x09)
/* NOTA :  Signals que no precisan comunicación a traves del pipe. */


/*------------ Primitivas auxiliares para liberación de memoria: -----------*/
#define ED_LIBERACION_MEMO_REQ  (UWORD)(INDICE_NIVEL2+0x17)
#define ED_LIBERACION_MEMO_IND  (UWORD)(INDICE_NIVEL2+0x18)


/*---------------------------------------------------------------------------
  Signals intercambiados entre la entidad de nivel 2 y la entidad de gestión
  de capa para la correcta liberación de la memoria dinámica solicitada para
  el envío de las tramas GED-UI:
---------------------------------------------------------------------------*/
#define GED_LIBERACION_MEMO_REQ   (UWORD)(INDICE_NIVEL2+0x20)
#define GED_LIBERACION_MEMO_IND   (UWORD)(INDICE_NIVEL2+0x21)


/*------- Estructura para comunicación entre los niveles 2 y 3: ------------*/
typedef struct{
                UWORD tipo;
                UWORD longDatos;
                UBYTE *datos;
              }MENSAJE23;

#define LONG_MENSAJE23  (UWORD)(sizeof(MENSAJE23))
```

```
/*************************************************************************/
/*                              NIVEL2.HP                                */
/*            Fichero de definiciones propias al Nivel 2                 */
/*                                                                       */
/*            Indice de proceso = 0x0200 (Indica nivel 2)                */
/*       Este indice estará en el byte de mayor peso de los signals      */
/*************************************************************************/


/*---- Definiciones de los estados del nivel 1 y de los bits I/R y P/F: ----*/
#define ACTIVO   1
#define INACTIVO 0


/*-------------------------------------------------------------------------
  Definiciones Estados de los Puntos Extremo de Conexion de datos pto a pto;
  Corresponden con el comportamiento de la capa de Enlace de Datos tal como
  es observado por la capa 3:
-------------------------------------------------------------------------*/
#define IET_NO_ASIGNADO            (UWORD)(INDICE_NIVEL2+0x50)
/*  Estados Estables: */
#define ENLACE_ESTABLECIDO         (UWORD)(INDICE_NIVEL2+0x51)
#define ENLACE_LIBERADO            (UWORD)(INDICE_NIVEL2+0x52)
/*  Estados de Transición: */
#define ESTABLECIMIENTO_PENDIENTE  (UWORD)(INDICE_NIVEL2+0x53)
#define LIBERACION_PENDIENTE       (UWORD)(INDICE_NIVEL2+0x54)


/*------------ Subestados del estado ENLACE_ESTABLECIDO ----------------------
  Corresponden a las posibles combinaciones de los dos siguientes casos:
  1.- Espera o no de confirmación de trama I enviada
  2.- Situación del remoto (RR ó RNR, es decir, Receptor Dispuesto o
      No Dispuesto).
-------------------------------------------------------------------------*/
#define NO_WAITING_ACK_RR          (UWORD)(INDICE_NIVEL2+0x55)
#define NO_WAITING_ACK_RNR         (UWORD)(INDICE_NIVEL2+0x56)
#define WAITING_ACK_RR             (UWORD)(INDICE_NIVEL2+0x57)
#define WAITING_ACK_RNR            (UWORD)(INDICE_NIVEL2+0x58)


/*-------------------------------------------------------------------------
  Parametros del Sistema asociados con cada Punto de Acceso al Servicio PAS:
-------------------------------------------------------------------------*/
#define T200  1
  /* tiempo (en sg) hasta retransmisión de trama;
  cuando finaliza, se incrementa contador retransmisiones */
#define T203  8
  /* tiempo maximo permitido sin intercambio de tramas
  (La norma indica un máximo de 10 sg) */
#define N200  3
  /* maximo número de retransmisiones de una trama */
#define N201  260
  /* maximo número de octetos en el campo de información de una trama */


/*------------ Definiciones de los tipos de trama protocolo LAP-D: ---------*/
#define I             (UWORD)  0
#define UI            (UWORD)  1
#define RR            (UWORD)  2
#define RNR           (UWORD)  3
#define REJ           (UWORD)  4
#define DM            (UWORD)  5
#define SABME         (UWORD)  6
#define DISC          (UWORD)  7
#define UA            (UWORD)  8
#define DESCONOCIDO   (UWORD)  9


/*Número de bytes que contendrá una trama que se intercambia con el nivel 1:*/
#define LONG_MAX_TRAMA_I    (UWORD) ((LONG_HEADER)  + (N201))
```

```
#define LONG_MAX_TRAMA_UI   (UWORD) ((SHORT_HEADER) + (N201))
#define LONG_CERO           (UWORD) 0


/*------- Tipos de "signals" que se intercambian con el nivel físico: ------*/
#define FI_DATOS_ENVIADOS   (UWORD) 0xa1
#define FI_DATOS_RECIBIDOS  (UWORD) 0xa2
#define FI_ERROR_IND        (UWORD) 0xa3


/*------------ Estructuras del cabecero de trama: --------------------------*/

/*------------ Definiciones y macros para formar el campo de direccion: ----*/
#define IPAS_SHIFT      (UBYTE) 2
#define IET_SHIFT       (UBYTE) 1
#define BIT_IR_SHIFT    (UBYTE) 1
#define LSBIT_LSB_DIR   (UBYTE) 1
#define IPAS_GED        (UBYTE) 63
#define IET_DE_GRUPO    (UBYTE) 127   /*0x7f*/

/* macro que genera el byte mas significativo del campo de direccion: */
#define MSB_DIR(numIPAS,bitIR)\
  (UBYTE)((numIPAS << IPAS_SHIFT) | (bitIR << BIT_IR_SHIFT))

/* genera el byte menos significativo del campo de direccion: */
#define LSB_DIR(numIET)\
  (UBYTE)(((numIET) << (IET_SHIFT)) | ((LSBIT_LSB_DIR)))


/*------------ Definiciones y macros para formar el campo de control: ------*/
#define SABME_MASK      (UBYTE) 0x6f
#define DM_MASK         (UBYTE) 0X0F
#define UI_MASK         (UBYTE) 0x03
#define DISC_MASK       (UBYTE) 0x43
#define UA_MASK         (UBYTE) 0x63
#define MSB_CTRL_RR     (UBYTE) 0x01
#define MSB_CTRL_RNR    (UBYTE) 0x05
#define MSB_CTRL_REJ    (UBYTE) 0x09
#define BIT_PF_SHIFT    (UBYTE) 4
#define NX_SHIFT        (UBYTE) 1     /* Desplazamiento para NS y NR */
#define ADR_LENGTH      (UBYTE) 2
#define LONG_CTRL       (UBYTE) 2
#define SHORT_CTRL      (UBYTE) 1
#define LONG_HEADER     (UWORD) ((ADR_LENGTH) + (LONG_CTRL))
#define SHORT_HEADER    (UWORD) ((ADR_LENGTH) + (SHORT_CTRL))

/* macro que genera el campo de control de las tramas de supervision: */
#define CTRL_TRAMAS_U(MASCARA,bitPF)\
  (UBYTE)((MASCARA) | ((bitPF) << (BIT_PF_SHIFT)))

/* genera el byte de menor peso del campo de control
  de las tramas de supervision y de informacion: */
#define LSB_CTRL_TRAMAS_SI(nr,bitPF)\
  (UBYTE)(((nr) << (NX_SHIFT)) | (bitPF))

/* genera el byte de mayor peso de las tramas de informacion:*/
#define MSB_CTRL_TRAMAS_I(ns)\
  (UBYTE)((ns) << (NX_SHIFT))



/*-------------------------------------------------------------------------
              Definiciones y macros para identificar tipo trama
              (para identificar bit I/R, bit P/F, IET, IPAS)
-------------------------------------------------------------------------*/
#define BIT_IR_MASK           (UBYTE) 0x02
#define BIT_ED_MASK           (UBYTE) 0x01
#define BIT_PF_MASK           (UBYTE) 0x10
#define TRAMAS_U_MASK         (UBYTE) 0xef
#define BIT_PF_TRAMAS_SI_MASK (UBYTE) 0x01
```

```
#define I_MASK                  (UBYTE) 0x01
#define IPAS_CERO               (UBYTE) 0

/* macro que nos devuelve el valor del bit I/R: */
#define GET_BIT_IR(MSBadr)\
  (UBYTE) (((MSBadr) & (BIT_IR_MASK)) >> (BIT_IR_SHIFT))

/* devuelve el valor del bit ED (bit de menor peso de los octetos que forman
   el campo de dirección (1 ó 0): */
#define GET_BIT_ED(octeto)  (UBYTE) ((octeto) & (BIT_ED_MASK))

/* devuelve el valor IPAS (Identificador Punto de Acceso al Servicio: */
#define GET_IPAS(MSBadr)(UBYTE) ((MSBadr) >> (IPAS_SHIFT))

/* devuelve el valor del IET (Identificador Equipo Terminal: */
#define GET_IET(LSBadr) (UBYTE) ((LSBadr) >> (IET_SHIFT))

/* devuelve valor del bit P/F de las tramas U (tramas no numeradas): */
#define GET_BIT_PF_TRAMAS_U(ctrl)\
  (UBYTE) (((ctrl) & (BIT_PF_MASK)) > BIT_PF_SHIFT)

/* devuelve el valor del bit P/F de las tramas S e I:*/
#define GET_BIT_PF_TRAMAS_SI(octeto)\
  (UBYTE) ((octeto) & (BIT_PF_TRAMAS_SI_MASK))

/* devuelve numero secuencial ns o nr, segun el caso: */
#define GET_NX(octeto)  (UBYTE) ((octeto) >> (NX_SHIFT))

/* elimina el bit P/F para encontrar tipo trama U: */
#define GET_CTRL_TRAMAS_U(octeto)\
  (UBYTE) ((octeto) & (TRAMAS_U_MASK))

/* devuelve 1 ó 0 según sea trama UI o otra cualquiera posible: */
#define ES_TRAMA_UI(octeto)\
  (UBYTE) ((((octeto) & (UI_MASK)) == (UI_MASK)) ? TRUE : FALSE)

/* devuelve valor del bit menos significatico de un byte: */
#define LSBIT(octeto)   (UBYTE) ((octeto) & (I_MASK))

/*-------- Macro para saber si un nro. secuencia de rx. es válido o no: ---*/
#define NR_VALIDO( nr, va, vs)\
  ((((va) <= (nr)) && ((nr) <= (vs))) ? TRUE : FALSE)


/*------- Definiciones auxiliares a los distintos buffers utilizados: --------
 Buffer de Tramas Transmitidas:   bufferTx
 Buffer de información confirmada pendiente de envío: bufferTramasI
 Buffer de información sin confirmación(UI) pendiente de envío: bufferTramasUI
------------------------------------------------------------------------------*/

#define LONG_BUF_TX             8
#define LONG_BUF_TRAMAS_I      20
#define LONG_BUF_TRAMAS_UI     20
#define LONG_BUF_TRAMAS_CTRL    4

/* Definición tamaño de la ventana (en Acceso Básico: k = ventana = 1) */
#define K   1

/* Definiciones de tipos de estructuras que componen los distintos buffers:
   bufferTx, bufferTramasI, bufferTramasUI, y ventana: */
typedef struct{
                UWORD tipo;
                UBYTE *header;
                UBYTE *info;
              }TRAMA_BUFFER_TX;

typedef struct{
                UBYTE *info;
                UWORD longInfo;
              }TRAMA_BUFFER_I;
```

```
typedef struct{
                  UBYTE *info;
                  UWORD longInfo;
                  UBYTE numIPAS;
                }TRAMA_BUFFER_UI;

typedef struct{
                 UBYTE *header;
                 UWORD lengHeader;
                }TRAMA_BUFFER_CTRL;

typedef struct{
                  UBYTE *header;
                  UWORD lengHeader;
                  UBYTE *info;
                  UWORD longInfo;
                }TRAMA_VENTANA;

/* Definiciones de los tipos de tramas que se indicarán dentro de la
   estructura bufferTx: */
#define TIPO_I              (UBYTE)   40
#define TIPO_UI             (UBYTE)   41
#define TIPO_GED_UI         (UBYTE)   42
#define TIPO_NOT_I_NOR_UI   (UBYTE)   43
```

```
/*****************************************************************************/
/*                              GED.HP                                       */
/*        Fichero de definiciones propias para el tratamiento de             */
/*        la Entidad de Gestión de Enlace del Nivel 2                        */
/*                                                                           */
/*              Indice de proceso = 0x0200 (Indica nivel 2)                  */
/*        Este indice estará en el byte de mayor peso de los signals         */
/*****************************************************************************/


/*---------------------------------------------------------------------------
  Signals intercambiados entre la entidad (proceso) de nivel 2 y la entidad
  de gestion de capa:
---------------------------------------------------------------------------*/
/* Signals entre los procesos N2_ET y GED_ET: */
#define GED_ASIGNACION_IET_IND     (UWORD)(INDICE_NIVEL2+0x30)
#define GED_ASIGNACION_IET_REQ     (UWORD)(INDICE_NIVEL2+0x31)
#define GED_NO_DISPONIBLE_IET_REQ  (UWORD)(INDICE_NIVEL2+0x32)
#define GED_SUPRESION_IET_REQ      (UWORD)(INDICE_NIVEL2+0x33)
#define GED_ERROR_IND              (UWORD)(INDICE_NIVEL2+0x34)
/* Signals entre los procesos N2_ET y GED_ET ó N2_TR y GED_TR: */
#define GED_UNIDAD_DATOS_REQ       (UWORD)(INDICE_NIVEL2+0x35)
#define GED_UNIDAD_DATOS_IND       (UWORD)(INDICE_NIVEL2+0x36)


/* Identificador de la Entidad de Gestión de Capa: */
#define IDENTIFICADOR_GED       (UBYTE) 0x0f

/* Estados posibles en el proceso de Gestión de Enlace de Datos del lado del
   Terminal de Red: */
#define IDLE                                    (UWORD) (0x0000)
#define ESPERA_RESP_PRUEBA_ID_IET_DE_GRUPO      (UWORD) (0x0001)
#define ESPERA_RESP_PRUEBA_ID_IET_CONCRETO      (UWORD) (0x0002)


/*---------------------------------------------------------------------------
  Parámetros del Sistema asociados con cada Punto de Acceso al Servicio PAS
  Estos parámetros se utilizan en el proceso de Gestión de Enlace de Datos
---------------------------------------------------------------------------*/
#define T201  T200
/* tiempo mínimo (en sg) entre retransmisiones de mensajes de prueba de
   identidad IET */
#define T202  2
/* tiempo mínimo entre la transmisión de mensajes de petición de identidad
   IET */
#define N202  3
/* número maximo de transmisiones de un mensaje de petición de identidad */
#define N204_BY_ME  2
/* Este es un parámetro definido por mi (necesario para cumplir requisitos
   de la norma Q.921, aunque está no lo defina específicamente).
   Representa el número de transmisiones de mensajes de petición de prueba
   de identidad y de verificación de identidad */

#define MAX_IETS_ASIGNADOS  62
#define DIFERENCIA_RANGO    64

#define ASIGNADO          1
#define NO_ASIGNADO       0
#define POSICION_1ER_INDICADOR_ACCION ((ADR_LENGTH) + (LONG_GED_UI_INFO) - 1)
#define IDENTIFICADO      1
#define NO_IDENTIFICADO   0

/*------------ Definiciones de los tipos de mensajes ------------------------
  El tipo del mensaje intercambiados entre las Entidades de Gestión de Capa
  del lado de Usuario y del lado de Red son los siguientes:
---------------------------------------------------------------------------*/
#define PETICION_IDENTIDAD          (UBYTE) 0x01  /*de Usuario a Red*/
#define IDENTIDAD_ASIGNADA          (UBYTE) 0x02  /*de Red a Usuario*/
#define IDENTIDAD_RECHAZADA         (UBYTE) 0x03  /*de Red a Usuario*/
#define PETICION_PRUEBA_IDENTIDAD   (UBYTE) 0x04  /*de Red a Usuario*/
```

```
#define RESPUESTA_PRUEBA_IDENTIDAD (UBYTE) 0x05  /*de Usuario a Red*/
#define SUPRESION_IDENTIDAD        (UBYTE) 0x06  /*de Red a Usuario*/
#define VERIFICACION_IDENTIDAD     (UBYTE) 0x07  /*de Usuario a Red*/


/*-------------------------------------------------------------------------
  Definiciones y macros para formar las tramas UI que se intercambian entre
  la entidad de gestión de capa del lado de usuario y la entidad de gestión
  de capa del lado de red:
-------------------------------------------------------------------------*/
#define IPAS_CERO         (UBYTE)   0
#define IET_CERO          (UBYTE)   0
#define IET_GED           (UBYTE) 127
#define LONG_GED_UI_INFO  (UWORD)   5
#define BYTE_SHIFT        (UBYTE)   8

#define MSB_MASK                (UWORD) 0xff00
#define LSB_MASK                (UWORD) 0X00ff
#define LSBIT_MASK              (UBYTE) 0X01
#define INDICADOR_ACCION_SHIFT  (UBYTE) 0x01
#define LSBIT_INDICADOR_ACCION  (UBYTE) 0x01

#define MAKE_MSB_NUM_REF(Ri) \
  (UWORD) (((Ri) & (MSB_MASK)) \
  >> (BYTE_SHIFT))

#define MAKE_LSB_NUM_REF(Ri)  (UWORD) ((Ri) & (LSB_MASK))

#define MAKE_BYTE_INDICADOR_ACCION(Ai) \
  (UBYTE) (((Ai) << (INDICADOR_ACCION_SHIFT)) \
  + (LSBIT_INDICADOR_ACCION))


/*-------------------------------------------------------------------------
  Definiciones y macros para identificar las tramas UI que se intercambian
  la entidad de gestión de capa del lado de usuario y la entidad de gestión
  de capa del lado de red:
-------------------------------------------------------------------------*/
#define OCTETO_NULO  (UBYTE) 0

/* Macro que a partir del número de referencia dividido en dos UBYTES
   (dos octetos) forma un UWORD (palabra de 16 bits): */
#define GET_NUM_REF( numRefMSB, numRefLSB) \
  (UWORD) (((numRefMSB) << (BYTE_SHIFT)) \
  + (numRefLSB))

/* Devuelve el indicador de acción contenido en un octeto de entrada: */
#define GET_INDICADOR_ACCION(Ai) \
  (UBYTE) ((Ai) >> (INDICADOR_ACCION_SHIFT))

/* Devuelve el valor del bit menos significativo del octeto indicador de
   acción: */
#define GET_LSBIT_OCTETO_INDICADOR_ACCION(Ai) \
  (UBYTE) ((Ai) & (LSBIT_MASK))


/*-------------------------------------------------------------------------
   Definiciones máximos rangos de los buffers que guardan las peticiones y
   las verificaciones de identidad en espera de ser atendidas:
-------------------------------------------------------------------------*/
#define LONG_BUF_PETICIONES_ID      4 /*20*/
#define LONG_BUF_VERIFICACIONES_ID  4 /*20*/

#define LONG_VEC_NUM_REF            4 /*8*/
```

```
/*****************************************************************************/
/*                              MUX.HP                                       */
/*        Fichero de definiciones propias correspondientes al                */
/*        Proceso Multiplexor (lado del Terminal de Red)                     */

/*****************************************************************************/

typedef struct{
                UBYTE libre;
                UBYTE numIETasignado;
                UBYTE idProcesoTR;
                UWORD pipeEscMuxTR;
              }ELEM_VEC_MUX;

#define NUM_PROCESOS_N2_TR 12   /*Longitud vector vecMux*/


#define DISPONIBLE      1
#define NO_DISPONIBLE   0

#define TRAMA_GED_UI    1
#define TRAMA_TR_UI     2
#define TRAMA_TR        3

#define INCLUIDO        1
#define NO_INCLUIDO     0


/* Definición estructura correspondiente a los elementos del bufferTxMux: */
typedef struct{
                 UWORD idProcesoTRemisor;
              }TRAMA_BUFFER_TX_MUX;

#define LONG_BUF_TX_MUX  8


/*Definición estructura correspondiente a los elementos del bufferTramasAtx:*/
typedef struct{
                UBYTE idProcesoTRemisor;
                UBYTE *header;
                UWORD lengHeader;
                UBYTE *info;
                UWORD longInfo;
              }ELEM_BUF_TRAMAS_A_TX;

#define LONG_BUF_TRAMAS_A_TX  30
```

```
/*************************************************************************/
/*                          MUX_TR.H                                     */
/*     Fichero de definiciones compartidas entre el proceso Multiplexor  */
/*     y los procesos que representan al Terminal de Red:                */
/*************************************************************************/


/*-----------------------------------------------------------------------
   Signals y/o tipos de mensajes intercambiados entre el proceso Multiplexor
   y los procesos (entidades) que representan el Terminal de Red:
-----------------------------------------------------------------------*/
#define MUX_TR_DATOS_REQ           (UWORD)(INDICE_NIVEL2+70) /*signal y
                                                               tipoMensaje*/
#define MUX_TR_DATOS_IND           (UWORD)(INDICE_NIVEL2+71)  /*tipoMensaje*/
#define MUX_TR_ENLACE_LIBERADO_REQ (UWORD)(INDICE_NIVEL2+72)  /*tipoMensaje*/
#define MUX_TR_ENLACE_LIBERADO_IND (UWORD)(INDICE_NIVEL2+73)  /*signal*/
#define MUX_TR_ID_PROCESO_REQ      (UWORD)(INDICE_NIVEL2+74)  /*tipoMensaje*/
#define MUX_TR_DATOS_ENVIADOS_IND  (UWORD)(INDICE_NIVEL2+75)  /*signal*/


/*-----------------------------------------------------------------------
   Estructura de los mensajes intercambiados entre el proceso multiplexor
   y las entidades (procesos) de nivel 2 que representan al Terminal de Red:
-----------------------------------------------------------------------*/
typedef struct{
                UWORD tipoMensaje;
                UBYTE idProcesoTR;
                UBYTE *header;
                UWORD lengHeader;
                UBYTE *info;
                UWORD longInfo;
              }MENSAJE_MUX_TR;

#define LONG_MENSAJE_MUX_TR  (UWORD) (sizeof(MENSAJE_MUX_TR))


/*-----------------------------------------------------------------------
   Valor del parámetro que se pasa en el momento de la creación a los procesos
   de nivel 2 del lado del TR:
     1.- Sólo uno de ellos recibirá el valor que lo identifica como proceso
         master
     2.- El proceso master es el único que se comunica con el proceso gestor
         de enlace de datos del lado del terminal de red (GED_TR); además
         todas las tramas tipo UI provenientes de los equipos terminales (ETs)
         serán enviadas al proceso master y éste a su vez las pasará al nivel3
-----------------------------------------------------------------------*/
#define PROCESO_MASTER       (UWORD) 1
#define PROCESO_NO_MASTER    (UWORD) 0
```

```
/*****************************************************************************/
/*                              N2_ET.C                                      */
/*          TRABAJO FIN DE CARRERA  SANTIAGO MARTIN ROMANI:                  */
/*              IMPLEMENTACION PROTOCOLO LAP-D PARA EL                       */
/*            MICROCONTROLADOR DE COMUNICACIONES MC68302                     */
/*                                                                           */
/*      IMPLEMENTACION PROCESO NIVEL 2 PARA EL LADO DEL EQUIPO TERMINAL      */
/*****************************************************************************/

#define UNIX
/*#define $CODESEG S105*/
/*#define $INITSEG S205*/
/*#define $DATASEG S305*/


/*------------ FICHEROS INCLUIDOS: ------------------------------------------*/
#include "basico.h"    /* ..\kernel\basico.h */
#include "kernel.h"    /* ..\kernel\kernel.h */
#include "inicio.h"
#include "n2n3.h"
#include "ged.hp"
#include "nivel2.hp"
#include "nivel1.h"


/*------------ DECLARACION DE VARIABLES GLOBALES: --------------------------*/
UWORD signal;
UWORD estado; /* Estado ptos extremos de conexion de enlace datos pto a pto */

UWORD estadoEnTx;
/*---------------------------------------------------------------------------
  Estado en el proceso de transmisión (subestado del estado EnlaceEstablecido
  en el que se encuentra la conexión de Enlace de Datos en el proceso de tx/rx
---------------------------------------------------------------------------*/

UBYTE estadoNivel1; /* Estado capa o nivel físico */
UWORD timer;  /* temporizador: funcionará como contador hasta T200 ó T203 */

/*---------- Variable puntero trama; se utiliza de dos formas: ---------------
  1.- Como puntero al cabecero de una trama en el proceso de formación y
      envío de una trama
  2.- Como puntero a la zona de memoria que contiene una trama recién
      recibida por el nivel físico
---------------------------------------------------------------------------*/
UBYTE *trama;

UWORD longTrama;  /* Longitud de la trama en bytes */
UWORD n200;      /* número de retransmisiones */
UWORD n200bis;   /* número de transmisiones (incluyendo retransmisiones) */
UWORD n201;      /* número de octetos en campo de información de trama*/

/*---------------------------------------------------------------------------
  Estructura para la comunicación con la entidad de nivel 3 y con la entidad
    de gestión de capa de Capa de Enlace de Datos
  La información intercambiada através de los pipes utilizará esta estructura
    es decir, en el pipe de escritura se escribirá un mensaje de información
    cuyos campos coincidirán con los de esta estructura
---------------------------------------------------------------------------*/
MENSAJE23 mensaje23;

/*---------------------------------------------------------------------------
  Estructura que define los valores de los parametros del puerto y del
  protocolo HDLC que se intercambian con el nivel físico:
---------------------------------------------------------------------------*/
PARAM_SCC parametros;

UBYTE numIETasignado, numIPASasignado;
UBYTE numIETrecibido, numIPASrecibido;

/*---------------------------------------------------------------------------
  Indica si hay una trama de información (trama I) esperando en la ventana
   para ser enviada
```

```
/*-------------------------------------------------------------------------
  Buffer de ED_DATOS_REQ (mensajes entre entidades pares de la capa
  solicitante) recibidos del nivel 3: esta información se enviará
  de forma transparente al remoto dentro de las tramas I
-------------------------------------------------------------------------*/
TRAMA_BUFFER_I  bufferTramasI[LONG_BUF_TRAMAS_I];
UWORD           frenteBtramasI, finalBtramasI;

/*-------------------------------------------------------------------------
  Buffer de ED_UNIDAD_DATOS_REQ recibidos del nivel 3 y GED_UNIDAD_DATOS_REQ
  recibidos del Gestor de Enlace de Datos del lado ET (son mensajes sin acuse
  de recibo): esta información se enviará de forma transparente al remoto
  dentro de las tramas UI
-------------------------------------------------------------------------*/
TRAMA_BUFFER_UI bufferTramasUI[LONG_BUF_TRAMAS_UI];
UWORD           frenteBtramasUI, finalBtramasUI;

/*------- Buffer de tramas de control en espera de ser transmitidas --------*/
TRAMA_BUFFER_CTRL bufferTramasCtrl[LONG_BUF_TRAMAS_CTRL];
UWORD             frenteBtramasCtrl, finalBtramasCtrl;

/*-------------------------------------------------------------------------
   Tramas de Información enviadas en espera de confirmación:
   PARTICULARIDAD: Para el Acceso Básico sólo 1 trama ya que
   (k = ventana = 1)
-------------------------------------------------------------------------*/
TRAMA_VENTANA ventana;


/*------------ Variables secuenciales de funcionamiento multitrama: --------*/
UBYTE vs;
/* V(S): nº secuencia siguiente trama I que debe transmitirse ( 0 a n-1)
   V(S) < V(A) + K */
UBYTE va;
/* V(A): nº última trama de la que el par del ECED haya acusado recibo */
UBYTE ns;
/* N(S): nº secuencial en emisión de las tramas I transmitidas */
UBYTE vr;
/* V(R): nº secuencia de la próxima trama I que se espera recibir */
UBYTE nr;
/* N(R): nº secuencia en rx de la próxima trama I que se espera recibir
   V(A) <= N(R) <= V(S) */



/*------------ DEFINICIONES DE PROTOTIPOS DE FUNCIONES: --------------------*/
void  n2_ETmain( UWORD pipeLecturaN2N3, UWORD pipeLecturaN2GED,
  UWORD idPuerto);
void  InicializaParametros( void);
UWORD IETnoAsignado( void);
UWORD EnlaceLiberado( void);
UWORD EstablecimientoPendiente( void);
UWORD LiberacionPendiente( void);
UWORD EnlaceEstablecido( void);
UWORD NoWaitingAck_RR( void);
UWORD NoWaitingAck_RNR( void);
UWORD WaitingAck_RR( void);
UWORD WaitingAck_RNR( void);

UWORD IdentificaTrama( UBYTE *frame, UWORD frameLength,
  UBYTE *bitioIR, UBYTE *bitioPF,
  UBYTE *numIETrecibido, UBYTE *numIPASrecibido);
UWORD FormarCabeceroTrama( UBYTE **header, UWORD *headerLength,
  UWORD tipoTrama, UBYTE bitIR, UBYTE bitPF);
void  FormarMensaje23( MENSAJE23 *mensaje23, UWORD tipoMens, UWORD longDatos,
  UBYTE *datos);

UWORD GuardaBtx( UWORD tipo, UBYTE *header, UBYTE *info);
UWORD RecuperaBtx(void);
void  LiberaBtx(void);
```

```
UWORD GuardaBtramasI( UBYTE *info, UWORD longInfo);
void  LiberaBtramasI( void);
UWORD GuardaBtramasUI( UBYTE *info, UWORD longInfo, UBYTE numIPAS);
void  LiberaBtramasUI( void);
UWORD GuardaBtramasCtrl( UBYTE *header, UWORD lengHeader);
void  LiberaBtramasCtrl( void);
void  LiberaVentana( void);

UWORD EnviarTramaCtrl( UWORD frameType, UBYTE bitioIR, UBYTE bitioPF);
UWORD EnviarTramaDelBufferUI( void);
UWORD EnviarTramaDelBufferCtrl( void);
UWORD EnviarTramaVentana( void);

void  TratarTramaUIrecibida( UBYTE numIETrecibido, UBYTE numIPASrecibido);
UWORD TratarEDunidadDatosReq( void);
UWORD TratarEDdatosReq( void);
UWORD TratarFIdatosEnviados( void);
UWORD TratarGEDunidadDatosReq( void);



/*------------ DECLARACION DE FUNCIONES:  ----------------------------------*/

void  n2_ETmain( pipeLecturaN2N3, pipeLecturaN2GED, idPuerto)
UWORD pipeLecturaN2N3;    /* Pipes para la communicacion con el nivel 3 */
UWORD pipeLecturaN2GED;
UWORD idPuerto;           /* Identificador puerto físico que controla */
{
  /*-------------------------------------------------------------------------
     Parámetros de Entrada al Proceso que representa a la entidad de nivel de
     Enlace de Datos (nivel 2) del lado del Terminal del Equipo Terminal:
  -------------------------------------------------------------------------*/
  pipeLecN2N3   = pipeLecturaN2N3;
  pipeLecN2GED  = pipeLecturaN2GED;
  idSCC         = idPuerto;

  /* Se solicita al Kernel que nos indique cuál es núestro identificador de
     proceso: */
  idProcesoN2 = KIDProceso();

  /*---- Procesado primer mensaje que recibimos del nivel 3: ----------------
    Se utiliza para identificar el pipe de escritura (pipe lectura para el
    nivel 3); también se indica el identificador de proceso de nivel 3 con el
    que nos comunicaremos:
  -------------------------------------------------------------------------*/
  KLeePipe( pipeLecN2N3, (VOID *)(&mensaje23), LONG_MENSAJE23, KWAIT);
  idProcesoN3 = mensaje23.tipo;
  pipeEscN2N3 = mensaje23.longDatos;

  /*--- Procesado 1er mensaje recibido de la Entidad de Gestión de Capa: -----
    Se utiliza para identificar el pipe de escritura (pipe lectura para la
    Entidad de Gestión de Capa); también se indica el identificador de proceso
    de dicha Entidad 3 con la que nos comunicaremos:
  -------------------------------------------------------------------------*/
  KLeePipe( pipeLecN2GED, (VOID *)(&mensaje23), LONG_MENSAJE23, KWAIT);
  idProcesoGED = mensaje23.tipo;
  pipeEscN2GED = mensaje23.longDatos;

  /* Ejecutar función que inicializa parámetros (variables globales) */
  InicializaParametros();

  /* Se solicita al Kernel el temporizador que utilizaremos en el resto del
     código del proceso */
  KSolicitaContador( &timer);

  /*--------- Inicializaciacion estado punto extremo de conexion: ----------*/
  estado = IET_NO_ASIGNADO;
  while(TRUE)    /* El proceso vive para siempre */
  {
    switch(estado)
    {
```

```
      case IET_NO_ASIGNADO:
        estado = IETnoAsignado();
        break;
      case ENLACE_LIBERADO:
        estado = EnlaceLiberado();
        break;
      case ESTABLECIMIENTO_PENDIENTE:
        estado = EstablecimientoPendiente();
        break;
      case ENLACE_ESTABLECIDO:
        estado = EnlaceEstablecido();
        break;
      case LIBERACION_PENDIENTE:
        estado = LiberacionPendiente();
        break;
    }  /*switch(estado)*/
  }  /*while(TRUE)*/
}  /*n2_ETmain*/


void InicializaParametros( void)
{
  UWORD i;
  /*--------- Inicializar el puerto fisico: --------------------------------*/
  parametros.hdlc.IDProceso  = idProcesoN2;
  parametros.hdlc.signal[0]  = FI_DATOS_ENVIADOS;
  parametros.hdlc.signal[1]  = FI_DATOS_RECIBIDOS;
  parametros.hdlc.signal[2]  = FI_ERROR_IND;
  parametros.hdlc.caracRecep = LONG_MAX_TRAMA_I;
  FIniciaSCC( idSCC, (UWORD)(PROTOCOLO_HDLC), &parametros);

  /*--------- Inicialización índices buffers: ------------------------------*/
  frenteBtx         = finalBtx         = 0;
  frenteBtramasUI   = finalBtramasUI   = 0;
  frenteBtramasI    = finalBtramasI    = 0;
  frenteBtramasCtrl = finalBtramasCtrl = 0;

  /*-------------------------------------------------------------------------
    Inicialización variable "número IPAS asignado" de la conexión de Enlace
     de Datos a cero (0)
  -------------------------------------------------------------------------*/
  numIPASasignado = IPAS_CERO;

  /*-------------------------------------------------------------------------
    Inicialización variable esperando trama de supervisión respuesta
    Se activa (tomará el valor TRUE), cuando hayamos enviado una trama de
    supervisión y pasamos a esperar una trama de supervisión respuesta
  -------------------------------------------------------------------------*/
  esperandoRespuesta = FALSE;
}



UWORD GuardaBtx( tipo, header, info)
UWORD tipo;
UBYTE *header, *info;
{
  /*-------------------------------------------------------------------------
    Inserta un elemento (tipo trama, cabecero e información, en su caso)
    en la cola de tramas de transmisión (bufferTx) y devuelve TRUE.
    Si la cola está llena, devuelve FALSE

      ACLARACION IMPORTANTE: El bufferTx junto con las funciones RecuperaBtx
    y LiberaBtx son necesarias para tener un control secuencial de las tramas
    enviadas al nivel físico que esperan recibir un signal FI_DATOS_ENVIADOS
    para liberar la memoria dinámica utilizada para formar el cabecero.
  -------------------------------------------------------------------------*/
  /* Comprobamos si cola llena: */
  if ((frenteBtx == finalBtx+1) ||
    ((frenteBtx == 0) && (finalBtx == LONG_BUF_TX-1)))
    return(FALSE);
  else
```

```
  {
    bufferTx[finalBtx].tipo   = tipo;
    bufferTx[finalBtx].header = header;
    bufferTx[finalBtx].info   = info;
    if ((++finalBtx) == LONG_BUF_TX)
      finalBtx = 0;
    return(TRUE);
  }
} /*GuardaBtx*/


UWORD RecuperaBtx(void)
{
  /*----- Por cada signal FI_DATOS_ENVIADOS, se ejecutará esta función: -----
    1.- Se comprueba tipo de la trama que corresponde: Si es distinto de
        TIPO_I, se libera la memoria dinámica asociada al cabecero de la trama
    2.- Si la trama es TIPO_UI, se indicará al nivel 3 que libere la memoria
        dinámica asociada a la información recibida con el signal
        ED_UNIDAD_DATOS_REQ correspondiente.
    3.- Si la trama es TIPO_GED_UI, se indicará a la entidad de gestión de
        enlace de datos del lado del Equipo Terminal que libere la memoria
        asociada con la trama enviada (sin campo de dirección) con el signal
        GED_UNIDAD_DATOS_REQ correspondiente.
    4.- Para cualquier tipo de tramas (TIPO_UI, TIPO_I ó TIPO_NOT_I_NOR_UI)
        se incrementa el índice frenteBtx de forma apropiada.
    NOTA:          La función devuelve FALSE si el buffer está vacío.
    ADVERTENCIA:   Sólo existirá una trama TIPO_I en espera de confirmación y
                   se guarda en su propia estructura: estructura "ventana"
                   (RECORDAR: En el Acceso Básico la ventana es uno)
        La función RecuperaBtx es necesaria para mantener un control del
        orden en el que se libera la memoria correspondiente a las tramas
        entregadas al nivel físico y confirmadas por este con cada signal
        FI_DATOS_ENVIADOS
  ------------------------------------------------------------------------*/
  /* Comprobación cola vacía: */
  if (frenteBtx == finalBtx)
    return(FALSE);
  else
  {
    if (bufferTx[frenteBtx].tipo != TIPO_I)
      KLiberaMemoria( (UWORD *)(bufferTx[frenteBtx].header));
    if (bufferTx[frenteBtx].tipo == TIPO_UI)  /*IPAS = 0 e IET = 127*/
    {
      FormarMensaje23( &mensaje23, ED_LIBERACION_MEMO_IND, LONG_CERO,
        bufferTx[frenteBtx].info);
      KEscribePipe( pipeEscN2N3, (void *)&mensaje23, LONG_MENSAJE23, KWAIT);
      KSignal( idProcesoN3, ED_LIBERACION_MEMO_IND);
    }
    else
      if (bufferTx[frenteBtx].tipo == TIPO_GED_UI)
      {
        /* Significa bufferTx[frenteBtx].tipo == TIPO_GED_UI, es decir,
         IPAS = IPAS_GED = 63  e  IET = IET_DE_GRUPO = 127*/
        FormarMensaje23( &mensaje23, GED_LIBERACION_MEMO_IND, LONG_CERO,
          bufferTx[frenteBtx].info);
        KEscribePipe( pipeEscN2GED, (void *)&mensaje23, LONG_MENSAJE23, KWAIT);
        KSignal( idProcesoGED, GED_LIBERACION_MEMO_IND);
      }
      /* El resto de tramas sólo tiene cabecero y, por tanto, no hay que
         liberar un campo de información */
    if ((++frenteBtx) == LONG_BUF_TX)
      frenteBtx = 0;
    return(TRUE);
  } /*else if (frenteBtx == finalBtx)*/
} /*RecuperaBtx*/


void LiberaBtx(void)
{
  /*------------------------------------------------------------------------
    Realiza los mismos pasos que la función RecuperaBtx, pero para cada una de
      las tramas (ver comentario secuencia de pasos en la propia función
```

```
      RecuperaBtx)
    Se ejecutará está función, es decir, se liberará el bufferTx o buffer de
      transmisión sólo cuando el nivel físico se desactive
  ------------------------------------------------------------------------*/
  while (frenteBtx != finalBtx)
  {
    if (bufferTx[frenteBtx].tipo != TIPO_I)
      KLiberaMemoria( (UWORD *)(bufferTx[frenteBtx].header));

    if (bufferTx[frenteBtx].tipo == TIPO_UI)  /*IPAS = 0 e IET = 127*/
    {
      FormarMensaje23( &mensaje23, ED_LIBERACION_MEMO_IND, LONG_CERO,
        bufferTx[frenteBtx].info);
      KEscribePipe( pipeEscN2N3, (void *)&mensaje23, LONG_MENSAJE23, KWAIT);
      KSignal( idProcesoN3, ED_LIBERACION_MEMO_IND);
    }
    else
      if (bufferTx[frenteBtx].tipo == TIPO_GED_UI)
      {
        /* Significa bufferTx[frenteBtx].tipo == TIPO_GED_UI, es decir,
           IPAS = IPAS_GED = 63  e  IET = IET_DE_GRUPO = 127*/
         FormarMensaje23( &mensaje23, GED_LIBERACION_MEMO_IND, LONG_CERO,
           bufferTx[frenteBtx].info);
         KEscribePipe( pipeEscN2GED, (void *)&mensaje23, LONG_MENSAJE23,
           KWAIT);
         KSignal( idProcesoGED, GED_LIBERACION_MEMO_IND);
      }
      /* El resto de tramas sólo tiene cabecero y, por tanto, no hay que
         liberar un campo de información */
    if ((++frenteBtx) == LONG_BUF_TX)
      frenteBtx = 0;
  } /*while (frenteBtx != finalBtx)*/
} /*LiberaBtx*/


UWORD GuardaBtramasI( info, longInfo)
UBYTE *info;
UWORD longInfo;
{
  /*--- Por cada signal ED_DATOS_REQ recibido, se ejecutará esta función: ---
    1.- Se añade al bufferTramasI la información del mensaje proveniente
        del nivel 3.
    2.- Se incrementa apropiadamente el índice finalBtramasI

    NOTA 1: La función devuelve FALSE si el buffer está lleno.
    NOTA 2: Cuando la ventana esté libre y el remoto esté dispuesto a recibir,
            formaremos una trama I (añadiendóle el correspondiente cabecero) y
            la enviaremos al nivel físico.
  ------------------------------------------------------------------------*/
  /* Comprobamos si cola llena:*/
  if ((frenteBtramasI == finalBtramasI+1) ||
    ((frenteBtramasI == 0) && (finalBtramasI == LONG_BUF_TRAMAS_I-1)))
    return(FALSE);
  else
  {
    bufferTramasI[finalBtramasI].info       = info;
    bufferTramasI[finalBtramasI].longInfo   = longInfo;
    if ((++finalBtramasI) == LONG_BUF_TRAMAS_I)
      finalBtramasI = 0;
    return(TRUE);
  }
}


void LiberaBtramasI(void)
{
  /*------------------------------------------------------------------------
     Libera la información enviada en los mensajes asociados con los signals
       ED_DATOS_REQ pendiente de envío
     Será necesario liberar el buffer de tramas I cuando se desactive el nivel
       físico o cuando se libere el enlace de datos
  ------------------------------------------------------------------------*/
```

```
  while(frenteBtramasI != finalBtramasI)
  {
    FormarMensaje23( &mensaje23, ED_LIBERACION_MEMO_IND, LONG_CERO,
      bufferTramasI[frenteBtramasI].info);
    KEscribePipe( pipeEscN2N3, (void *)&mensaje23, LONG_MENSAJE23, KWAIT);
    KSignal( idProcesoN3, ED_LIBERACION_MEMO_IND);
    if (++frenteBtramasI == LONG_BUF_TX)
      frenteBtramasI = 0;
  }
}


UWORD GuardaBtramasUI( info, longInfo, numIPAS)
UBYTE *info, numIPAS;
UWORD longInfo;
{
  /*-------------------------------------------------------------------------
    Por cada signal ED_UNIDAD_DATOS_REQ o GED_UNIDAD_DATOS_REQ recibidos,
    se ejecutará esta función:
      1.- Se añade al bufferTramasUI la información del correspondiente
          mensaje: información, su longitud y el IPAS que me identifica
          de dónde proviene la trama UI:
            Si numIPAS = IPAS_CERO = 0, proviene del nivel 3
            Si numIPAS = IPAS_GED = 63, proviene del Gestor de Enlace de Datos
      2.- Se incrementa apropiadamente el índice finalBtramasUI
    NOTA: La función devuelve FALSE si el buffer está lleno.
  -------------------------------------------------------------------------*/
  /* Comprobamos si cola llena:*/
  if ((frenteBtramasUI == finalBtramasUI+1) ||
    ((frenteBtramasUI == 0) && (finalBtramasUI == LONG_BUF_TRAMAS_I-1)))
    return(FALSE);
  else
  {
    bufferTramasUI[finalBtramasUI].info       = info;
    bufferTramasUI[finalBtramasUI].longInfo   = longInfo;
    bufferTramasUI[finalBtramasUI].numIPAS    = numIPAS;
    if ((++finalBtramasUI) == LONG_BUF_TRAMAS_I)
      finalBtramasUI = 0;
    return(TRUE);
  }
}


void LiberaBtramasUI(void)
{
  /*-------------------------------------------------------------------------
     Libera la información enviada en los mensajes asociados con los signals
       ED_UNIDAD_DATOS_REQ o GED_UNIDAD_DATOS_REQ pendientes de envío
     Será necesario liberar el buffer de tramas UI sólo si se desactiva el
       nivel físico
  -------------------------------------------------------------------------*/
  while(frenteBtramasUI != finalBtramasUI)
  {
    if (bufferTramasUI[finalBtramasUI].numIPAS == IPAS_CERO)
    {
      FormarMensaje23( &mensaje23, ED_LIBERACION_MEMO_IND, LONG_CERO,
        bufferTramasUI[frenteBtramasUI].info);
      KEscribePipe( pipeEscN2N3, (void *)&mensaje23, LONG_MENSAJE23, KWAIT);
      KSignal( idProcesoN3, ED_LIBERACION_MEMO_IND);
    }
    else /*bufferTramasUI[finalBtramasUI].numIPAS == IPAS_GED*/
    {
      FormarMensaje23( &mensaje23, GED_LIBERACION_MEMO_IND, LONG_CERO,
        bufferTramasUI[frenteBtramasUI].info);
      KEscribePipe( pipeEscN2GED, (void *)&mensaje23, LONG_MENSAJE23, KWAIT);
      KSignal( idProcesoGED, GED_LIBERACION_MEMO_IND);
    }
    if (++frenteBtramasUI == LONG_BUF_TX)
      frenteBtramasUI = 0;
  } /*while(frenteBtramasUI != finalBtramasUI)*/
}
```

```
UWORD GuardaBtramasCtrl( header, lengHeader)
UBYTE *header;
UWORD lengHeader;
{
  /*-------------------------------------------------------------------------
    Será necesario guardar una trama de control que no pueda ser enviada en un
    momento determinado, para hacerlo posteriormente, cuando el nivel físico
    asi lo permita.
  -------------------------------------------------------------------------*/
  /* Comprobamos si cola llena:*/
  if ((frenteBtramasCtrl == finalBtramasCtrl+1) ||
    ((frenteBtramasCtrl == 0) &&
    (finalBtramasCtrl == LONG_BUF_TRAMAS_CTRL-1)))
    return(FALSE);
  else
  {
    bufferTramasCtrl[finalBtramasCtrl].header       = header;
    bufferTramasCtrl[finalBtramasCtrl].lengHeader   = lengHeader;
    if ((++finalBtramasCtrl) == LONG_BUF_TRAMAS_CTRL)
      finalBtramasCtrl = 0;
    return(TRUE);
  }
}


void LiberaBtramasCtrl(void)
{
  /* Será necesario liberar el buffer de tramas de control sólo si se
     desactiva el nivel físico */
  while(frenteBtramasCtrl != finalBtramasCtrl)
  {
    KLiberaMemoria( (UWORD *)(bufferTramasCtrl[frenteBtramasCtrl].header));
    if (++frenteBtramasCtrl == LONG_BUF_TRAMAS_CTRL)
      frenteBtramasCtrl = 0;
  }
}


void LiberaVentana(void)
{
  /*-------------------------------------------------------------------------
    Libera la memoria dinámica correspondiente al cabecero y a la información
    de la trama I enviada en espera de confirmación que forma la ventana.
    NOTA:  Se ejecutará esta función cuando se confirma la trama I que forma
           la ventana, o cuando se reinicializa o se pierde el enlace de
           nivel 2 o la conexión física y existe una trama I por confirmar
           en la ventana.
  -------------------------------------------------------------------------*/
  KLiberaMemoria( (UWORD *)ventana.header);
  FormarMensaje23( &mensaje23, ED_LIBERACION_MEMO_IND, LONG_CERO,
    ventana.info);
  KEscribePipe( pipeEscN2N3, (void *)&mensaje23, LONG_MENSAJE23, KWAIT);
  KSignal( idProcesoN3, ED_LIBERACION_MEMO_IND);
}


void FormarMensaje23( mensaje23, tipoMens, longDatos, datos)
MENSAJE23 *mensaje23;
UWORD tipoMens, longDatos;
UBYTE *datos;
{
  /*-------------------------------------------------------------------------
    Forma un mensaje, es decir, actualiza la estructura mensaje con los campos
    apropiados que se quieren transmitir. (Posteriormente el mensaje será
    enviado a la entidad de nivel 3 o al gestor de enlace de datos
  -------------------------------------------------------------------------*/
  mensaje23->tipo      = tipoMens;
  mensaje23->longDatos = longDatos;
  mensaje23->datos     = datos;
}
```

```
UWORD FormarCabeceroTrama( header, headerLength, tipoTrama, bitIR, bitPF)
UBYTE **header, bitIR, bitPF;
UWORD *headerLength, tipoTrama;
{
  /*-------------------------------------------------------------------------
     Esta función forma el cabecero de la trama que se solicita con los
     parámetros adecuados a cada tipo de trama
  -------------------------------------------------------------------------*/
  switch(tipoTrama)
  {
    case SABME:
      *headerLength = SHORT_HEADER;
      if (KPideMemoria( *headerLength, (UWORD **)(header)) == KNO_ERROR)
      {
        (*header)[0] = MSB_DIR( numIPASasignado, INACTIVO);
        (*header)[1] = LSB_DIR( numIETasignado);
        (*header)[2] = CTRL_TRAMAS_U( SABME_MASK, ACTIVO);
        return(TRUE);
      }
      else return(FALSE);
    case DM:
      *headerLength = SHORT_HEADER;
      if (KPideMemoria( *headerLength, (UWORD **)(header)) == KNO_ERROR)
      {
        (*header)[0] = MSB_DIR( numIPASasignado, ACTIVO);
        (*header)[1] = LSB_DIR( numIETasignado);
        (*header)[2] = CTRL_TRAMAS_U( DM_MASK, ACTIVO);
        return(TRUE);
      }
      else return(FALSE);
    case DISC:
      *headerLength = SHORT_HEADER;
      if (KPideMemoria( *headerLength, (UWORD **)(header)) == KNO_ERROR)
      {
        (*header)[0] = MSB_DIR( numIPASasignado, INACTIVO);
        (*header)[1] = LSB_DIR( numIETasignado);
        (*header)[2] = CTRL_TRAMAS_U( DISC_MASK, ACTIVO);
        return(TRUE);
      }
      else return(FALSE);
    case UA:
      *headerLength = SHORT_HEADER;
      if (KPideMemoria( *headerLength, (UWORD **)(header)) == KNO_ERROR)
      {
        (*header)[0] = MSB_DIR( numIPASasignado, ACTIVO);
        (*header)[1] = LSB_DIR( numIETasignado);
        (*header)[2] = CTRL_TRAMAS_U( UA_MASK, ACTIVO);
        return(TRUE);
      }
      else return(FALSE);
    case RR:
      *headerLength = LONG_HEADER;
      if (KPideMemoria( *headerLength, (UWORD **)(header)) == KNO_ERROR)
      {
        (*header)[0] = MSB_DIR( numIPASasignado, bitIR);
        (*header)[1] = LSB_DIR( numIETasignado);
        (*header)[2] = MSB_CTRL_RR;
        (*header)[3] = LSB_CTRL_TRAMAS_SI( nr, bitPF);
        return(TRUE);
      }
      else return(FALSE);
    case RNR:
      *headerLength = LONG_HEADER;
      if (KPideMemoria( *headerLength, (UWORD **)(header)) == KNO_ERROR)
      {
        (*header)[0] = MSB_DIR( numIPASasignado, bitIR);
        (*header)[1] = LSB_DIR( numIETasignado);
        (*header)[2] = MSB_CTRL_RNR;
        (*header)[3] = LSB_CTRL_TRAMAS_SI( nr, bitPF);
```

```
        return(TRUE);
      }
      else return(FALSE);
    case REJ:
      *headerLength = LONG_HEADER;
      if (KPideMemoria( *headerLength, (UWORD **)(header)) == KNO_ERROR)
      {
        (*header)[0] = MSB_DIR( numIPASasignado, bitIR);
        (*header)[1] = LSB_DIR( numIETasignado);
        (*header)[2] = MSB_CTRL_REJ;
        (*header)[3] = LSB_CTRL_TRAMAS_SI( nr, bitPF);
        return(TRUE);
      }
      else return(FALSE);
    case I:
      *headerLength = LONG_HEADER;
      if (KPideMemoria( *headerLength, (UWORD **)(header)) == KNO_ERROR)
      {
        (*header)[0] = MSB_DIR( numIPASasignado, INACTIVO);
        (*header)[1] = LSB_DIR( numIETasignado);
        (*header)[2] = MSB_CTRL_TRAMAS_I( ns);
        (*header)[3] = LSB_CTRL_TRAMAS_SI( nr, ACTIVO);
        return(TRUE);
      }
      else return( FALSE);
    case UI:
      *headerLength = SHORT_HEADER;
      if (KPideMemoria( *headerLength, (UWORD **)(header)) == KNO_ERROR)
      {
        (*header)[0] = MSB_DIR( numIPASasignado, INACTIVO);
        (*header)[1] = LSB_DIR( numIETasignado);
        (*header)[2] = CTRL_TRAMAS_U( UI_MASK, INACTIVO);
        return(TRUE);
      }
      else return(FALSE);
    case TIPO_GED_UI:
      /*-------------------------------------------------------------------------
         En el caso de una trama proveniente del GED (Gestor de Enlace de
         Datos del lado del ET (Equipo Terminal, será la entidad de nivel 2
         la que forme la cabecera con IPAS_GED (63) e IET_DE_GRUPO (127):
      -------------------------------------------------------------------------*/
      *headerLength = (UWORD)ADR_LENGTH;
      if (KPideMemoria( *headerLength, (UWORD **)(header)) == KNO_ERROR)
      {
        (*header)[0] = MSB_DIR( IPAS_GED, INACTIVO);
        (*header)[1] = LSB_DIR( IET_DE_GRUPO);
      }
      else return(FALSE);
    default:
      return(FALSE);
  } /*switch(tipoTrama)*/
} /*FormarCabeceroTrama*/


UWORD IdentificaTrama( frame, frameLength, bitioIR, bitioPF,
  numIETrecibido, numIPASrecibido)
UBYTE *frame, *bitioIR, *bitioPF, *numIETrecibido, *numIPASrecibido;
UWORD frameLength;
{
  /*-------------------------------------------------------------------------
    Esta función identifica el tipo de trama recibida a partir de la trama
    recibida por el nivel físico y de su longitud, teniendo encuenta los
    distintos campos que caracterizan a las tramas (campo de dirección y campo
    de control) y los distintos valores que pueden tomar
  -------------------------------------------------------------------------*/
  if ((GET_BIT_ED(frame[0]) != 0) || (GET_BIT_ED(frame[1]) != 1))
    return(DESCONOCIDO);
  *numIPASrecibido = GET_IPAS(frame[0]);
  *bitioIR         = GET_BIT_IR(frame[0]);
  *numIETrecibido  = GET_IET(frame[1]);
```

```
if (((*numIPASrecibido == numIPASasignado) &&
  (*numIETrecibido == numIETasignado))
  || ((ES_TRAMA_UI(frame[2])) && (*numIETrecibido == IET_DE_GRUPO)))
  /*-------------------------------------------------------------------------
  Si los valores de IET y IPAS de la trama recibida no coinciden con los
  valores locales;o dicha trama no es tipo UI y el IET no es el de difusión,
  dicha trama se descarta:
  -------------------------------------------------------------------------*/
  switch(frameLength)
  {
    case SHORT_HEADER:  /*cabecero de 3 bytes*/
      /* Se trata de tramas tipo U (no numeradas) */
      *bitioPF = GET_BIT_PF_TRAMAS_U(frame[2]);
      switch(GET_CTRL_TRAMAS_U(frame[2]))
      {
        case SABME_MASK:
          if (*bitioIR == INACTIVO)
            return(SABME);
          else
            return(DESCONOCIDO);
        case DM_MASK:
          if (*bitioIR == ACTIVO)
            return(DM);
          else
            return(DESCONOCIDO);
        case DISC_MASK:
          if (*bitioIR == INACTIVO)
            return(DISC);
          else
            return(DESCONOCIDO);
        case UA_MASK:
          if (*bitioIR == ACTIVO)
            return(UA);
          else
            return(DESCONOCIDO);
        default:
          return(DESCONOCIDO);
      }
    case LONG_HEADER: /*cabecero de 4 bytes*/
      /*Se trata de tramas tipo S (Supervision)*/
      *bitioPF = GET_BIT_PF_TRAMAS_SI(frame[3]);
      nr = GET_NX(frame[3]);
      switch(frame[2])
      {
        case MSB_CTRL_RR:
          return(RR);
        case MSB_CTRL_RNR:
          return(RNR);
        case MSB_CTRL_REJ:
          return(REJ);
        default:
          return(DESCONOCIDO);
      }
    default:
      if (LSBIT(frame[2]) == 0)
      {
        /* Se trata de tramas I  (Información): */
        ns = GET_NX(frame[2]);
        nr = GET_NX(frame[3]);
        *bitioPF = GET_BIT_PF_TRAMAS_SI(frame[3]);
        return(I);
      }
      else
        if (GET_CTRL_TRAMAS_U(frame[2]) == UI_MASK)
        {
          /* Se trata de tramas UI  (Información no numerada): */
          *bitioPF = GET_BIT_PF_TRAMAS_U(frame[2]);
          return(UI);
        }
        else
```

```
          {
            if ((*numIPASrecibido == IPAS_GED) &&
              (*numIETrecibido == IET_DE_GRUPO))
              /* Se trata de tramas GED_UI: */
              return(TIPO_GED_UI);
            else
              /* En otro caso, la trama recibida es de un tipo desconocido: */
              return(DESCONOCIDO);
          }
    } /*switch(frameLength)*/
  else /* if (((*numIPASrecibido == numIPASasignado) && ...)*/
    if ((*numIPASrecibido == IPAS_GED) && (*numIETrecibido == IET_DE_GRUPO))
      return( TIPO_GED_UI);
    else
      return(DESCONOCIDO);
} /*IdentificaTrama*/


UWORD EnviarTramaDelBufferUI( void)
{
  /*-------------------------------------------------------------------------
    Se ejecuta cada vez que se quiere enviar la 1ª trama en espera del
    bufferTramasUI:
      Si se logra enviar, se guarda en el bufferTx (buffer de transmisión)
        y se incrementa frenteBtramasUI apropiadamente
      Si no se puede enviar, la función devuelve FALSE
  -------------------------------------------------------------------------*/
  UWORD tipoTrama;
  if (bufferTramasUI[frenteBtramasUI].numIPAS == IPAS_CERO)
    tipoTrama = TIPO_UI;
  else /*(bufferTramasUI[frenteBtramasUI].numIPAS == IPAS_GED)*/
    tipoTrama = TIPO_GED_UI;
  FormarCabeceroTrama( &trama, &longTrama, tipoTrama,
    INACTIVO, INACTIVO);
  if (FEnviaDatos( idSCC, (void *)(trama), longTrama,
    (void *)(bufferTramasUI[frenteBtramasUI].info),
    bufferTramasUI[frenteBtramasUI].longInfo ) == TRUE)
  {
    GuardaBtx( tipoTrama, trama, bufferTramasUI[frenteBtramasUI].info);
    if (++frenteBtramasUI == LONG_BUF_TRAMAS_UI)
      frenteBtramasUI = 0;
    return(TRUE);
  }
  else
  {
    /* Si no se pudo enviar, tenemos que liberar la memoria dinámica
       utilizada para formar el cabecero de la trama: */
    KLiberaMemoria( (UWORD *)trama);
    /* La función devolverá FALSE si no se pudo transmitir la trama UI */
    return(FALSE);
  }
} /*EnviarTramaDelBufferUI()*/


UWORD EnviarTramaDelBufferCtrl( void)
{
  /*-------------------------------------------------------------------------
    Se ejecuta cada vez que se quiere enviar la 1ª trama en espera del
    bufferTramasCtrl:  (función análoga a "EnviarTramaDelBufferUI")
      Si se logra enviar, se guarda en el bufferTx y se incrementa
        frenteBtramasCtrl apropiadamente
      Si no se puede enviar, la función devuelve FALSE
  -------------------------------------------------------------------------*/
  if (FEnviaDatos( idSCC, (void *)bufferTramasCtrl[frenteBtramasCtrl].header,
    bufferTramasCtrl[frenteBtramasCtrl].lengHeader, NULL, LONG_CERO) == TRUE)
  {
    GuardaBtx( TIPO_NOT_I_NOR_UI, bufferTramasCtrl[frenteBtramasCtrl].header,
      NULL);
    if (++frenteBtramasCtrl == LONG_BUF_TRAMAS_CTRL)
      frenteBtramasCtrl = 0;
```

```
    return(TRUE);
  }
  else
    /*La función devolverá FALSE si no se pudo enviar la trama de control*/
    return(FALSE);
} /*EnviarTramaDelBufferCtrl()*/


UWORD EnviarTramaVentana( void)
{
  /*-------------------------------------------------------------------------
    Se ejecuta cuando se quiera enviar la trama I contenida en la ventana de
      transmisión.
    Si se logra enviar se guarda en el bufferTx; si no la función devuelve
      FALSE
  -------------------------------------------------------------------------*/
  if (FEnviaDatos( idSCC, (void *)(ventana.header), ventana.lengHeader,
    (void *)(ventana.info), ventana.longInfo) == TRUE)
  {
    GuardaBtx( TIPO_I, ventana.header, ventana.info);
    return(TRUE);
  }
  else
    return(FALSE);
} /*EnviarTramaVentana*/


void  TratarTramaUIrecibida( numIETrecibido, numIPASrecibido)
UBYTE numIETrecibido, numIPASrecibido;
{
  /*-------------------------------------------------------------------------
     Esta función se ejecuta cada vez que se recibe una trama UI
     (UI o TIPO_GED_UI):
       - Si se trata de una trama UI, se enviará al nivel3
       - Si se trata de una trama TIPO_GED_UI, se enviará a GED
  -------------------------------------------------------------------------*/
  if ((numIPASrecibido == IPAS_GED) && (numIETrecibido == IET_DE_GRUPO))
  {
    FormarMensaje23( &mensaje23, GED_UNIDAD_DATOS_IND, longTrama, trama);
    KEscribePipe( pipeEscN2GED, (void *)&mensaje23, LONG_MENSAJE23, KWAIT);
    KSignal( idProcesoGED, GED_UNIDAD_DATOS_IND);
  }
  else
    if (numIPASrecibido == IPAS_CERO)
    {
      FormarMensaje23( &mensaje23, ED_UNIDAD_DATOS_IND, longTrama, trama);
      KEscribePipe( pipeEscN2N3, (void *)&mensaje23, LONG_MENSAJE23, KWAIT);
      KSignal( idProcesoN3, ED_UNIDAD_DATOS_IND);
    }
    else
      KLiberaMemoria( (UWORD*)trama);
} /*TratarTramaUIrecibida*/


UWORD TratarEDdatosReq( void)
{
  /*-------------------------------------------------------------------------
    Guarda el mensaje de nivel 3 recibido para su posterior envío formando
    una trama I. Se ejecuta cuando estamos en uno de los estados que no
    permiten enviar en ese preciso momento una trama I:
      NoWaitingAck_RNR
      WaitingAck_RR
      WaitingAck_RNR
  -------------------------------------------------------------------------*/
  if (KLeePipe( pipeLecN2N3, (VOID *)&mensaje23, LONG_MENSAJE23, KWAIT)
    == KNO_ERROR)
    /* La función KLeePipe siempre devolverá KNO_ERROR */
  {
    GuardaBtramasI( mensaje23.datos, mensaje23.longDatos);
    return(TRUE);
```

```
  }
  return(FALSE);
}


UWORD TratarEDunidadDatosReq( void)
{
  /*-------------------------------------------------------------------------
    Esta función se ejecuta cada vez que recibo un signal ED_UNIDAD_DATOS_REQ:
      - Si bufferTramasUI está vacío:
          - Si se puede transmitir la información recién llegada formando una
            trama UI, se añade al bufferTx
      - Si no se puede transmitir o si el bufferTramasUI no está vacío:
          - Se añade la información recién llegada al propio bufferTramasUI
            con numIPAS = IPAS_CERO = 0.

    NOTA: Es necesario contemplar el caso que no se haya transmitido ninguna
          trama y llegue un ED_UNIDAD_DATOS_REQ, por eso si el bufferTramasUI
          está vacío se intenta transmitir la información recién llegada.
          Si no se contempla, no llegaría un FI_DATOS_ENVIADOS que permitiese
          transmitir esa 1ª trama UI al comenzar el programa.
  -------------------------------------------------------------------------*/
  if (KLeePipe( pipeLecN2N3, (VOID *)&mensaje23, LONG_MENSAJE23, KWAIT)
    == KNO_ERROR)
  {
    /* La función KLeePipe siempre devolverá KNO_ERROR */
    FormarCabeceroTrama( &trama, &longTrama, UI, INACTIVO, INACTIVO);
    if (frenteBtramasUI == finalBtramasUI)
    {
      if (FEnviaDatos( idSCC, (void *)(trama), longTrama,
         (void *)(mensaje23.datos), mensaje23.longDatos) == TRUE)
      {
        GuardaBtx( TIPO_UI, trama, mensaje23.datos);
        return(TRUE);
      }
    }
    /* Si no se pudo enviar, tenemos que liberar la memoria dinámica
       utilizada para formar el cabecero de la trama: */
    KLiberaMemoria( (UWORD *)trama);
    GuardaBtramasUI( mensaje23.datos, mensaje23.longDatos, IPAS_CERO);
    return(TRUE);
  }
  return(FALSE);
} /*TratarEDunidadDatosReq*/


UWORD TratarGEDunidadDatosReq( void)
{
  /*-------------------------------------------------------------------------
    Esta función se ejecuta cada vez que recibo un signal GED_UNIDAD_DATOS_REQ:
      - Si bufferTramasUI está vacío:
          - Si se puede transmitir la información recién llegada formando una
            trama TIPO_GED_UI, se añade al bufferTx
      - Si no se puede transmitir o si el bufferTramasUI no está vacío:
          - Se añade la información recién llegada al propio bufferTramasUI
            teniendo en cuenta que el número de IPAS destino será el IPAS_GED,
            es decir, el destino es el Gestor de Enlace del Terminal de Red

    NOTA: Es necesario contemplar el caso que no se haya transmitido ninguna
          trama y llegue un GED_UNIDAD_DATOS_REQ, por eso si el bufferTramasUI
          está vacío se transmite la información recién llegada.
          Si no se contempla, no llegaría un FI_DATOS_ENVIADOS que permitiese
          transmitir esa 1ª trama TIPO_GED_UI al comenzar el programa.
  -------------------------------------------------------------------------*/
  if (KLeePipe( pipeLecN2GED, (VOID *)&mensaje23, LONG_MENSAJE23, KWAIT)
    == KNO_ERROR)
  {
    /* La función KLeePipe siempre devolverá KNO_ERROR */
    FormarCabeceroTrama( &trama, &longTrama, TIPO_GED_UI, INACTIVO, INACTIVO);
    if (frenteBtramasUI == finalBtramasUI)
```

```
    {
      if (FEnviaDatos( idSCC, (void *)(trama), longTrama,
        (void *)(mensaje23.datos), mensaje23.longDatos) == TRUE)
      {
        GuardaBtx( TIPO_GED_UI, trama, mensaje23.datos);
        return(TRUE);
      }
    }
    /* Si no se pudo enviar, tenemos que liberar la memoria dinámica
       utilizada para formar el cabecero de la trama: */
    KLiberaMemoria( (UWORD *)trama);
    GuardaBtramasUI( mensaje23.datos, mensaje23.longDatos, IPAS_GED);
    return(TRUE);
  }
  return(FALSE);
} /*TratarGEDunidadDatosReq*/


UWORD TratarFIdatosEnviados( void)
{
  /*-------------------------------------------------------------------------
    Se ejecutará esta función por cada signal FI_DATOS_ENVIADOS:
    - Se ejecuta la función RecuperaBtx y después:
    - Si bufferTramasCtrl no está vacío:
        - Enviar la 1ª trama de control en espera
    - Si bufferTramasCtrl está vacío:
        - Si (ventanaEsperandoTx = TRUE)
            - Enviar trama I de la ventana y (ventanaEsperandoTx = FALSE)
        - Si no (ventanaEsperandoTx = TRUE)
            - Enviar la 1ª trama del bufferTramasUI
  -------------------------------------------------------------------------*/
  RecuperaBtx();
  /*-------------------------------------------------------------------------
    Se ejecutará la función RecuperaBtx cada vez que se reciba un signal
    FI_DATOS_ENVIADOS: Esta función es necesaria para mantener un control
    del orden en el que se libera la memoria correspondiente a las tramas
    entregadas al nivel físico y que son confirmadas por éste con cada
    signal FI_DATOS_ENVIADOS:
  -------------------------------------------------------------------------*/
  if (frenteBtramasCtrl != finalBtramasCtrl)
    EnviarTramaDelBufferCtrl();
  else
    if (ventanaEsperandoTx == TRUE)
    {
      if( EnviarTramaVentana() == TRUE)
      {
        numTxTramaI++;
        //ventanaEsperandoTx = FALSE;
      }
    }
    else
      if (frenteBtramasUI != finalBtramasUI)
        EnviarTramaDelBufferUI();
  return(TRUE);
} /*TratarFIdatosEnviados*/


UWORD EnviarTramaCtrl( frameType, bitioIR, bitioPF)
UWORD frameType;
UBYTE bitioIR, bitioPF;
{
  /*-------------------------------------------------------------------------
    - Si el bufferTramasCtrl está vacío:
        - Si se puede transmitir, se guarda en el bufferTx
    - Si no se pudo transmitir o si el bufferTramasCtrl no está vacío, se
      guarda en el bufferTramasCtrl para su posterior transmisión
  -------------------------------------------------------------------------*/
  FormarCabeceroTrama( &trama, &longTrama, frameType, bitioIR, bitioPF);
  if (frenteBtramasCtrl == finalBtramasCtrl)
  {
```

```
    if (FEnviaDatos( idSCC, (VOID *)(trama), longTrama, NULL, LONG_CERO)
      == TRUE)
    {
      GuardaBtx( TIPO_NOT_I_NOR_UI, trama, NULL);
      return(TRUE);
    }
  }
  GuardaBtramasCtrl( trama, longTrama);
  return(TRUE);
}


UWORD IETnoAsignado( void)
{
  /*--- Definiciones variables locales a este "estado" (a esta funcion): ---*/
  UWORD signal;
  UWORD tipoTrama;
  /* Tipo de trama del protocolo LAP-D que se ha recibido o que se enviará */
  UBYTE bitIR, bitPF;
  UBYTE numIETrecibido, numIPASrecibido;
  UWORD error;
  UWORD establecimientoPendiente;

  establecimientoPendiente = FALSE;
  while(TRUE)
  {
    KWait();
    KLeeSignal(&signal);
    switch(signal)
    {
      case ED_ESTABLEC_REQ:
        KSignal( idProcesoGED, GED_ASIGNACION_IET_IND);
        establecimientoPendiente = TRUE;
        break;
      case ED_UNIDAD_DATOS_REQ:
        if (establecimientoPendiente == FALSE)
        {
          KSignal( idProcesoGED, GED_ASIGNACION_IET_IND);
          establecimientoPendiente = TRUE;
          KLeePipe( pipeLecN2N3, (VOID *)&mensaje23, LONG_MENSAJE23, KWAIT);
          GuardaBtramasUI( mensaje23.datos, mensaje23.longDatos, IPAS_CERO);
        }
        else
        {
          /*-------------------------------------------------------------------
             Si recibo señales ED_UNIDAD_DATOS_REQ antes de establecer el
             enlace, le indicaré al proceso de nivel 3 que ya fueron enviadas
             porque si las guardo en el bufferTramasUI (para enviarlas
             una vez establecido el enlace) no podré diferenciarlas de las
             tramas asociadas a las señales GED_UNIDAD_DATOS_REQ que pueda
             recibir mientras la entidad de gestión está intentando
             comunicarse con su entidad par
          -------------------------------------------------------------------*/
          KLeePipe( pipeLecN2N3, (VOID *)&mensaje23, LONG_MENSAJE23, KWAIT);
          KEscribePipe( pipeEscN2N3, (void *)&mensaje23, LONG_MENSAJE23,
            KWAIT);
          KSignal( idProcesoN3, ED_LIBERACION_MEMO_IND);
        }
        break;
      case GED_ASIGNACION_IET_REQ:
        if (establecimientoPendiente == TRUE)
        {
          establecimientoPendiente = FALSE;
          if (KLeePipe( pipeLecN2GED, (VOID *)&mensaje23, LONG_MENSAJE23,
            KWAIT) == KNO_ERROR)
            /*-----------------------------------------------------------------
               La entidad de Gestión de Enlace de Datos enviará el número
               de IET asignado en el campo tipo de la estructura mensaje que
               se "envía" a través del pipeLecN2GED:
            -----------------------------------------------------------------*/
```

```
        numIETasignado = mensaje23.tipo;
      if (estadoNivel1 == INACTIVO)
        if (FActivaSCC(idSCC) == FALSE)
        {
          LiberaBtx();
          LiberaBtramasUI();
          KSignal( idProcesoN3, ED_LIBERACION_IND);
          break;
        }
      /* El programa continúa por aquí si el nivel físico está activo: */
      EnviarTramaCtrl( SABME, INACTIVO, ACTIVO);
      KActivaContador( timer, T200);
      vs = vr = va = 0;
      return(ESTABLECIMIENTO_PENDIENTE);
    }
    else /*if (establecimientoPendiente == TRUE)*/
      break;
  case GED_NO_DISPONIBLE_IET_REQ:
    if (establecimientoPendiente == TRUE)
    {
      establecimientoPendiente = FALSE;
      LiberaBtramasUI();
      KSignal( idProcesoN3, ED_LIBERACION_IND);
    }
    break;
  case GED_UNIDAD_DATOS_REQ:
    if (estadoNivel1 == INACTIVO)
      if (FActivaSCC(idSCC) == FALSE)
      {
        /* Si se perdió el nivel físico, o lo que es lo mismo, no se pudo
           activar, se liberan los buffers: */
        LiberaBtx();
        LiberaBtramasUI();
        KLeePipe( pipeLecN2GED, (VOID *)&mensaje23, LONG_MENSAJE23,
          KWAIT);
        KEscribePipe( pipeEscN2GED, (void *)&mensaje23, LONG_MENSAJE23,
          KWAIT);
        KSignal( idProcesoGED, GED_LIBERACION_MEMO_IND);
        if (establecimientoPendiente == TRUE)
          establecimientoPendiente = FALSE;
        break;
      }
    /* El programa continúa por aquí si el nivel físico está activo: */
    TratarGEDunidadDatosReq();
    break;
  case FI_DATOS_ENVIADOS:
    TratarFIdatosEnviados();
    /* Permanecemos en el estado actual: */
    break;
  case FI_ERROR_IND:
  case FI_DATOS_RECIBIDOS:
    if (FRecibeDatos( idSCC, (void **)&trama, &longTrama, &error) == TRUE)
    {
      if (error != 0)
      {
        if (trama != NULL)
          KLiberaMemoria( (UWORD *)trama);
        if (error & ERROR_DESACT)
        {
          LiberaBtx();
          LiberaBtramasUI();
          if (establecimientoPendiente == TRUE)
          {
            establecimientoPendiente = FALSE;
            KSignal( idProcesoN3, ED_LIBERACION_IND);
          }
          break;
        } /*if (error & ERROR_DESACT)*/
      }
      else  /*if (error != 0)*/
```

```
          {
            /* Si no hubo error al recibir la trama, se procesa: */
            tipoTrama = IdentificaTrama( trama, longTrama, &bitIR, &bitPF,
              &numIETrecibido, &numIPASrecibido);
            switch(tipoTrama)
            {
              case UI:
              case TIPO_GED_UI:
                TratarTramaUIrecibida( numIETrecibido, numIPASrecibido);
                break;
              default:
                /* Si recibimos cualquier otro tipo de trama, liberamos la
                   memoria correspondiente: */
                KLiberaMemoria( (UWORD *)trama);
                break;
            } /*switch(tipoTrama)*/
          } /* else if(error!=0)*/
        }
        else /*if (FRecibeDatos(..) == TRUE)*/
          break;
      case ED_LIBERACION_MEMO_REQ:
        KLeePipe( pipeLecN2N3, (VOID *)&mensaje23, LONG_MENSAJE23, KWAIT);
        KLiberaMemoria( (UWORD *)(mensaje23.datos));
        /* Permanecemos en el estado actual: */
        break;
      case GED_LIBERACION_MEMO_REQ:
        KLeePipe( pipeLecN2GED, (VOID *)&mensaje23, LONG_MENSAJE23, KWAIT);
        KLiberaMemoria( (UWORD *)(mensaje23.datos));
        /* Permanecemos en el estado actual: */
        break;
      case GED_SUPRESION_IET_REQ:
        establecimientoPendiente = FALSE;
        LiberaBtramasUI();
        KSignal( idProcesoN3, ED_LIBERACION_IND);
        break;
      default:
        break;
    } /*switch(signal)*/
  } /*while(TRUE)*/
} /*IETnoAsignado*/


UWORD EnlaceLiberado( void)
{
  /*--- Definiciones variables locales a este "estado" (a esta funcion): ---
    Estas definiciones son locales a cada una de las siguientes funciones que
    representan a cada uno de los posibles estados de la conexión de enlace
    de datos:
       EnlaceLiberado
       EstablecimientoPendiente
       LiberaciónPendiente
       WaitingAck_RR          |
       WaitingAck_RNR         |  Estos últimos cuatro son subestados
       NoWaitingAck_RR        |  del estado o función EnlaceEstablecido
       NoWaitingAck_RNR       |
  ------------------------------------------------------------------------*/
  UWORD tipoTrama;
  /* Tipo de trama del protocolo LAP-D que se ha recibido o que se enviará */
  UBYTE bitIR, bitPF;
  /*---------- Bit de Instrucción/Respuesta o Comando/Respuesta: ------------
       Es el 2º bit de menor peso del 1er. octeto del campo de dirección
       de todas las tramas del protocolo LAP-D.
     Bit de Polling/Final:
       Es el bit de menor peso del 2º octeto del campo de control de las
       tramas I y S del protocolo LAP-D.
     Los valores de ambos bits (ACTIVO = 1 ó INACTIVO = 0) se utilizarán en el
       proceso de formación y envío, y en el proceso de identificación de las
       tramas intercambiadas através de la conexión de nivel de enlace.
  ------------------------------------------------------------------------*/
  UBYTE numIETrecibido, numIPASrecibido;
```

```
UWORD error;      /* Error devuelto por el nivel físico */

/*-------------------------------------------------------------------------
   En los casos que se haya evolucionado a este estado porque se estaba
   esperando una respuesta y no se recibió después de N200 retransmisión de
   la instrucción, la variable debe ser puesta a FALSE:
-------------------------------------------------------------------------*/
if (esperandoRespuesta == TRUE)
  esperandoRespuesta = FALSE;

KWait();
KLeeSignal(&signal);
switch(signal)
{
  case FI_ERROR_IND:
  case FI_DATOS_RECIBIDOS:
    if (FRecibeDatos( idSCC, (void **)&trama, &longTrama, &error) == TRUE)
    {
      /*-----------------------------------------------------------------
         Si se ha producido algún error en la recepción de la trama, ésta
         se descarta. Los errores que el nivel físico puede detectar e
         indicar al nivel de enlace son los siguientes:
           ERROR_OVERRUN
           ERROR_FRAMING
           ERROR_PARITY
           ERROR_BREAK
           ERROR_ABORT
           ERROR_NONOCTET
           ERROR_CRC
           ERORR_DESACT
      -----------------------------------------------------------------*/
      if (error != 0)
      {
        /* Si se recibió una trama errónea, se liberará la memoria que ocupa
        */
        if (trama != NULL)
          KLiberaMemoria( (UWORD *)trama);
        /*-----------------------------------------------------------------
           Si se produce un error de desactivación del nivel físico,
           se libera la conexión de nivel de enlace y la memoria dinámica
           correspondiente a las tramas que se encuentren en los buffers:
           bufferTx y bufferTramasUI
           NOTA:  Además se indica al nivel 3 la liberación de la conexión
                  del nivel de enlace
        -----------------------------------------------------------------*/
        if (error & ERROR_DESACT)
        {
          /*liberar buffers de memoria de mensajes y tramas*/
          LiberaBtx();
          LiberaBtramasUI();
          LiberaBtramasCtrl();
          return( ENLACE_LIBERADO);
        }
      }
      else  /*if (error != 0)*/
      {
        /* Si no hubo error al recibir la trama, se procesa: */
        tipoTrama = IdentificaTrama( trama, longTrama, &bitIR, &bitPF,
          &numIETrecibido, &numIPASrecibido);
        switch(tipoTrama)
        {
          case UI:
          case TIPO_GED_UI:
            TratarTramaUIrecibida( numIETrecibido, numIPASrecibido);
            return( ENLACE_LIBERADO);
          case SABME:
            KLiberaMemoria( (UWORD*)trama);
            /*-----------------------------------------------------------
              Sólo en la función-estado EnlaceLiberado se comprueba si
                el nivel físico está activo y se intenta activar
```

```
            si no lo está; en el resto de las funciones-estado no.
          Esto se debe a que el nivel físico puede decidir pasar a un
            estado de mínima energía y desactivar la conexión física
            si no se utiliza durante un tiempo determinado.
        ------------------------------------------------------------------*/
        if (bitPF == ACTIVO)
        {
          if (estadoNivel1 == INACTIVO)
            if (FActivaSCC(idSCC) == FALSE)
            {
              LiberaBtx();
              LiberaBtramasUI();
              LiberaBtramasCtrl();
              return(ENLACE_LIBERADO);
            }
          /* El programa continúa por aquí si el nivel físico
             está activo: */
          EnviarTramaCtrl( UA, ACTIVO, ACTIVO);
          KActivaContador( timer, T203);
          vs = vr = va = 0;
          KSignal( idProcesoN3, ED_ESTABLEC_IND);
          return(ENLACE_ESTABLECIDO);
        } /*if (bitPF == ACTIVO)*/
        else return(ENLACE_LIBERADO);
      case DISC:
        KLiberaMemoria( (UWORD*)trama);
        if (bitPF == ACTIVO)
        {
          if (estadoNivel1 == INACTIVO)
            if (FActivaSCC(idSCC) == FALSE)
            {
              LiberaBtx();
              LiberaBtramasUI();
              LiberaBtramasCtrl();
              return(ENLACE_LIBERADO);
            }
          /* El programa continúa por aquí si el nivel físico
             está activo: */
          EnviarTramaCtrl( DM, ACTIVO, ACTIVO);
        }
        /* En cualquier caso, permanecemos en el estado actual: */
        return(ENLACE_LIBERADO);
      default:
        /* Si recibimos cualquier otro tipo de trama, liberamos la
           memoria correspondiente: */
        KLiberaMemoria( (UWORD *)trama);
        return(ENLACE_LIBERADO);
    } /*switch(tipoTrama)*/
  } /*else (error!=0)*/
}
else /*if (FRecibeDatos(...)==TRUE)*/
  return(ENLACE_LIBERADO);
case FI_DATOS_ENVIADOS:
  TratarFIdatosEnviados();
  /* Permanecemos en el estado actual: */
  return(ENLACE_LIBERADO);
case ED_ESTABLEC_REQ:
  if (estadoNivel1 == INACTIVO)
    if (FActivaSCC(idSCC) == FALSE)
    {
      LiberaBtx();
      LiberaBtramasUI();
      LiberaBtramasCtrl();
      KSignal( idProcesoN3, ED_LIBERACION_IND);
      return(ENLACE_LIBERADO);
    }
  /* El programa continúa por aquí si el nivel físico
     está activo: */
  EnviarTramaCtrl( SABME, INACTIVO, ACTIVO);
  KActivaContador( timer, T200);
```

```
      vs = vr = va = 0;
      return(ESTABLECIMIENTO_PENDIENTE);
    case ED_UNIDAD_DATOS_REQ:
      if (estadoNivel1 == INACTIVO)
        if (FActivaSCC(idSCC) == FALSE)
        {
          /* Si se perdió el nivel físico, o lo que es lo mismo, no se pudo
             activar, la conexión de nivel de enlace se libera: */
          LiberaBtx();
          LiberaBtramasUI();
          LiberaBtramasCtrl();
          KEscribePipe( pipeEscN2N3, (void *)&mensaje23, LONG_MENSAJE23,
            KWAIT);
          KSignal( idProcesoN3, ED_LIBERACION_MEMO_IND);
          return( ENLACE_LIBERADO);
        }
      /* El programa continúa por aquí si el nivel físico está activo: */
      TratarEDunidadDatosReq();
      /* En cualquier caso permanecemos en el estado actual: */
      return(ENLACE_LIBERADO);
    case GED_UNIDAD_DATOS_REQ:
      if (estadoNivel1 == INACTIVO)
        if (FActivaSCC(idSCC) == FALSE)
        {
          /* Si se perdió el nivel físico, o lo que es lo mismo, no se pudo
             activar, la conexión de nivel de enlace se libera: */
          LiberaBtx();
          LiberaBtramasUI();
          LiberaBtramasCtrl();
          KEscribePipe( pipeEscN2N3, (void *)&mensaje23, LONG_MENSAJE23,
            KWAIT);
          KSignal( idProcesoN3, ED_LIBERACION_MEMO_IND);
          return( ENLACE_LIBERADO);
        }
      /* El programa continúa por aquí si el nivel físico está activo: */
      TratarGEDunidadDatosReq();
      /* Permanecemos en el estado actual: */
      return(ENLACE_LIBERADO);
    case ED_LIBERACION_MEMO_REQ:
      KLeePipe( pipeLecN2N3, (VOID *)&mensaje23, LONG_MENSAJE23, KWAIT);
      KLiberaMemoria( (UWORD *)mensaje23.datos);
      /* Permanecemos en el estado actual: */
      return(ENLACE_LIBERADO);
    case GED_LIBERACION_MEMO_REQ:
      KLeePipe( pipeLecN2GED, (VOID *)&mensaje23, LONG_MENSAJE23, KWAIT) ;
      KLiberaMemoria( (UWORD *)(mensaje23.datos));
      /* Permanecemos en el estado actual: */
      return(ENLACE_LIBERADO);
    case GED_SUPRESION_IET_REQ:
      LiberaBtramasUI();
      return(IET_NO_ASIGNADO);
    default:
      return(ENLACE_LIBERADO);
  } /*switch(signal)*/
} /*EnlaceLiberado*/


UWORD EstablecimientoPendiente( void)
{
  /*--- Definiciones variables locales a este "estado" (a esta funcion): ---*/
  UWORD signal;
  UWORD tipoTrama;
  UWORD resp;
  UBYTE bitIR, bitPF;
  UBYTE numIETrecibido, numIPASrecibido;
  UWORD error;
  UWORD n200 = 0; /* Inicializo número de retransmisiones de la trama SABME */
  while( n200 < N200)  /*Hasta N200 = 3 retransmisiones de la trama SABME*/
  {
```

```
resp = KEsperaContador(timer);
if (resp == KNO_ERROR)  /*terminó el temporizador*/
{
  /*-------------------------------------------------------------------
     Permanecemos en el estado actual (EstablecimientoPendiente)
     hasta completar las N200 (3) retransmisiones, o al llegar la trama
     adecuada, por evolución a otro estado
  -------------------------------------------------------------------*/
  EnviarTramaCtrl( SABME, INACTIVO, ACTIVO);
  KActivaContador( timer, T200);
  vs = vr = va = 0;
  n200++;
  continue;  /*Salta a evaluar denuevo la condición del while*/
} /*if (resp == KNO_ERROR)*/
if (resp == KINTERRUP)
{
  KLeeSignal(&signal);
  switch(signal)
  {
    case FI_ERROR_IND:
    case FI_DATOS_RECIBIDOS:
      if (FRecibeDatos( idSCC, (void **)&trama, &longTrama, &error)
         == TRUE)
      {
        if (error != 0)
        {
          if (trama != NULL)
            KLiberaMemoria( (UWORD *)trama);
          if (error & ERROR_DESACT)
          {
            /*liberar buffers de memoria de mensajes y tramas*/
            LiberaBtx();
            LiberaBtramasUI();
            LiberaBtramasCtrl();
            KSignal( idProcesoN3, ED_LIBERACION_IND);
            return( ENLACE_LIBERADO);
          }
        }
        else  /*if (error != 0)*/
        {
          /* Si no hubo error al recibir la trama, se procesa: */
          tipoTrama = IdentificaTrama( trama, longTrama, &bitIR, &bitPF,
            &numIETrecibido, &numIPASrecibido);
          switch(tipoTrama)
          {
            case UI:
            case TIPO_GED_UI:
              TratarTramaUIrecibida( numIETrecibido, numIPASrecibido);
              continue;
            case UA:
              KLiberaMemoria( (UWORD *)trama);
              if (bitPF == ACTIVO)
              {
                vs = vr = va = 0;
                KActivaContador( timer, T203);
                KSignal( idProcesoN3, ED_ESTABLEC_CONF);
                return( ENLACE_ESTABLECIDO);
              }
              else continue;
            case SABME:
              KLiberaMemoria( (UWORD *)trama);
              if (bitPF == ACTIVO)
              {
                EnviarTramaCtrl( UA, ACTIVO, ACTIVO);
                KActivaContador( timer, T203);
                vs = vr = va = 0;
                /*-----------------------------------------------------
                  Respondemos enviando una trama UA, pero no pasaremos al
                  estado EnlaceEstablecido hasta que no recibamos una
                  trama UA
```

```
                    ------------------------------------------------------*/
                  }  /*if (biPF=ACTIVO)*/
                  continue;
                case DISC:
                  KLiberaMemoria( (UWORD *)trama);
                  if (bitPF == ACTIVO)
                  {
                    EnviarTramaCtrl( DM, ACTIVO, ACTIVO);
                    vs = vr = va = 0;
                    KSignal( idProcesoN3, ED_LIBERACION_IND);
                    return(ENLACE_LIBERADO);
                  }  /*if (biPF=ACTIVO)*/
                  else continue;
                case DM:
                  KLiberaMemoria( (UWORD *)trama);
                  if (bitPF == ACTIVO)
                  {
                    KSignal( idProcesoN3, ED_LIBERACION_IND);
                    return(ENLACE_LIBERADO);
                  }
                  else continue;
                default:
                 /* Si recibimos cualquier otro tipo de trama, liberamos la
                    memoria correspondiente: */
                  KLiberaMemoria( (UWORD *)trama);
                  continue;
              } /*switch( tipoTrama)*/
            } /*else if (error != 0)*/
          }  /*if (FRecibeDatos(...) == TRUE)*/
          else continue;
        case FI_DATOS_ENVIADOS:
          TratarFIdatosEnviados();
          /* Permanecemos en el estado actual: */
          continue;
        case ED_UNIDAD_DATOS_REQ:
          TratarEDunidadDatosReq();
          /* En cualquier caso permanecemos en el estado actual: */
          continue;
        case GED_UNIDAD_DATOS_REQ:
          TratarGEDunidadDatosReq();
          /* Permanecemos en el estado actual: */
          continue;
        case ED_LIBERACION_MEMO_REQ:
          KLeePipe( pipeLecN2N3, (VOID *)&mensaje23, LONG_MENSAJE23, KWAIT);
          KLiberaMemoria( (UWORD *)mensaje23.datos);
          /* Permanecemos en el estado actual: */
          continue;
        case GED_LIBERACION_MEMO_REQ:
          KLeePipe( pipeLecN2GED, (VOID *)&mensaje23, LONG_MENSAJE23, KWAIT) ;
          KLiberaMemoria( (UWORD *)mensaje23.datos);
          /* Permanecemos en el estado actual: */
          continue;
        case GED_SUPRESION_IET_REQ:
          KSignal( idProcesoN3, ED_LIBERACION_IND);
          LiberaBtramasUI();
          return(IET_NO_ASIGNADO);
        default:
          continue;
      }  /*switch(signal)*/
    }  /*if (resp == KINTERRUP)*/
  }  /*while(n200 < N200)*/
  /*-------------------------------------------------------------------------
     Si no se consiguió establecer el enlace después de las N200
     retransmisiones, se pasa al estado ENLACE_LIBERADO y se indica al nivel 3
     con el signal ED_LIBERACION_IND:
  -------------------------------------------------------------------------*/
  KSignal( idProcesoN3, ED_LIBERACION_IND);
  return(ENLACE_LIBERADO);
} /*EstablecimientoPendiente*/
```

```
UWORD EnlaceEstablecido()
{
  /* Inicialización variables globales empleadas en los subestados internos al
     estado EnlaceEstablecido: */
  n200 = 0;
  numTxTramaI = 0;

  /*-------------------------------------------------------------------------
    Al establecerse el enlace, no existe ninguna trama I esperando acuse de
      recibo (equivale a "NoWaitingAck") y el remoto está inicialmente
      preparado para recibir tramas I (equivale a "RR").
    Por esto el estado inicial en el proceso de transmisión/recepción
      (estadoEnTx) durante la conexión de enlace establecida será
      NO_WAITING_ACK_RR y por tanto la 1ª función que se ejecutará será
      NoWaitingAck_RR :
  -------------------------------------------------------------------------*/
  estadoEnTx = NO_WAITING_ACK_RR;
  while(TRUE)
  {
    switch(estadoEnTx)
    {
      /*-----------------------------------------------------------------------
        Existe cuatro subestados dentro del estado "ENLACE_ESTABLECIDO"
        que vienen definidos según dos parámetros:
          1.- El estado de espera de confirmación de la trama I (ventana 1):
              (esperando acuse de recibo o no).
          2.- El estado de la entidad remota:
              Receptor Preparado o No Preparado para recibir más tramas I
      -----------------------------------------------------------------------*/
      case NO_WAITING_ACK_RR:
        estadoEnTx = NoWaitingAck_RR();
      case NO_WAITING_ACK_RNR:
        estadoEnTx = NoWaitingAck_RNR();
      case WAITING_ACK_RR:
        estadoEnTx = WaitingAck_RR();
      case WAITING_ACK_RNR:
        estadoEnTx = WaitingAck_RNR();
      default:
        /*-----------------------------------------------------------------------
          Si el estado al que se evoluciona no es ninguno de los cuatro
            subestados, se pasa al discriminador de estados principal,
            es decir, se pasa al switch(estado) de la función Nivel2Main.
          Cuando estadoEnTx tome los siguientes valores:
            LIBERACION_PENDIENTE, ESTABLECMIENTO_PENDIENTE o ENLACE_LIBERADO,
            se pasará al switch (estado) de la función Nivel2Main.
        -----------------------------------------------------------------------*/
        return(estadoEnTx);
    }
  }
} /*EnlaceEstablecido*/


UWORD NoWaitingAck_RR( void)
{
  /*--- Definiciones variables locales a este "estado" (a esta funcion): ---*/
  UWORD signal;
  UWORD tipoTrama;
  UBYTE bitIR, bitPF;
  UBYTE numIETrecibido, numIPASrecibido;
  UWORD resp;
  UWORD error;          /* Error devuelto por el nivel físico */
  resp = KEsperaContador( timer);
  if (resp == KNO_ERROR)
  {
    if (n200 < N200)
    {
      nr = vr;
      EnviarTramaCtrl( RR, ACTIVO, ACTIVO);
```

```
      KActivaContador( timer, T203);
      n200++;
      esperandoRespuesta = TRUE;
      return(NO_WAITING_ACK_RR);
    }
    else /*if (n200 < N200), es decir cuando (n200 == N200)*/
    {
      EnviarTramaCtrl( DM, ACTIVO, ACTIVO);
      KSignal( idProcesoN3, ED_LIBERACION_IND);
      return(ENLACE_LIBERADO);
    }
  } /*if (resp == KNO_ERROR)*/
  else /*(resp == KINTERRUP)*/
  {
    /*Se ha recibido un signal, por tanto se procesa:*/
    KLeeSignal( &signal);
    switch( signal)
    {
      case FI_ERROR_IND:
      case FI_DATOS_RECIBIDOS:
        if (FRecibeDatos( idSCC, (void **)&trama, &longTrama, &error)
          == TRUE)
        {
          if (error != 0)
          {
            if (trama != NULL)
              KLiberaMemoria( (UWORD *)trama);
            if (error & ERROR_DESACT)
            {
              /*liberar buffers de memoria de mensajes y tramas*/
              LiberaBtx();
              LiberaBtramasI();
              LiberaBtramasCtrl();
              LiberaBtramasUI();
              KSignal( idProcesoN3, ED_LIBERACION_IND);
              return(ENLACE_LIBERADO);
            }
          }
          else  /*if (error != 0)*/
          {
            /* Si no hubo error al recibir la trama, se procesa: */
            tipoTrama = IdentificaTrama( trama, longTrama, &bitIR, &bitPF,
              &numIETrecibido, &numIPASrecibido);
            switch(tipoTrama)
            {
              case I:
                if ((ns == vr) && (NR_VALIDO(nr, va, ns)))
                {
                  if (nr == vs)
                  {
                    /* Si se cumple esta condición, está confirmando: */
                    va = nr;
                  }
                  vr++;
                  n200 = 0;
                  FormarMensaje23( &mensaje23, ED_DATOS_IND,
                    longTrama, trama);
                  KEscribePipe( pipeEscN2N3, (void *)&mensaje23,
                    LONG_MENSAJE23, KWAIT);
                  KSignal( idProcesoN3, ED_DATOS_IND);
                  nr = vr;  /* Para formar el cabecero de la trama RR */
                  /* Si bitPF = 0 ó 1, se enviará una respuesta RR (bitIR = 1)
                     y bitPF = 1: */
                  EnviarTramaCtrl( RR, ACTIVO, ACTIVO);
                  KActivaContador( timer, T203);
                  return(NO_WAITING_ACK_RR);
                }
                else /*if ((ns == vr) && (NR_VALIDO(nr, va, ns)))*/
                {
                  /* Se descarta las tramas I cuyo ns no sea igual a vr
```

```
                o cuando su nr no sea válido: */
              KLiberaMemoria( (UWORD *)trama);
              return(NO_WAITING_ACK_RR);
            }
          case RR:
            KLiberaMemoria( (UWORD *)trama);
            if (bitPF == ACTIVO)
            {
              if (esperandoRespuesta == TRUE)
                esperandoRespuesta = FALSE;
              nr = vr;  /* Para formar el cabecero de la trama RR */
              EnviarTramaCtrl( RR, ACTIVO, ACTIVO);
              KActivaContador( timer, T203);
              n200 = 0;
            }
            return(NO_WAITING_ACK_RR);
          case RNR:
            KLiberaMemoria( (UWORD *)trama);
            if ((bitPF == ACTIVO) && (NR_VALIDO(nr, va, ns)))
            {

              if (bitIR == INACTIVO)
              {
                /* Si la trama RNR recibida es una instrucción, se enviará
                   una trama RR respuesta: */
                nr = vr;  /* Para formar el cabecero de la trama RR */
                EnviarTramaCtrl( RR, ACTIVO, ACTIVO);
                n200 = 0;
              }
              else /*(bitIR == ACTIVO); trama recibida es una respuesta*/
                esperandoRespuesta = FALSE;
              KActivaContador( timer, T203);
              return(NO_WAITING_ACK_RNR);
            }
            else
              return(NO_WAITING_ACK_RR);
          case UI:
          case TIPO_GED_UI:
            TratarTramaUIrecibida( numIETrecibido, numIPASrecibido);
            return(NO_WAITING_ACK_RR);
          case DISC:
            KLiberaMemoria( (UWORD*)trama);
            if (bitPF == ACTIVO)
            {
              EnviarTramaCtrl( UA, ACTIVO, ACTIVO);
              vs = vr = va = 0;
              KSignal( idProcesoN3, ED_LIBERACION_IND);
              return(ENLACE_LIBERADO);
            }  /*if (biPF=ACTIVO)*/
            return(NO_WAITING_ACK_RR);
          case SABME:
            KLiberaMemoria( (UWORD *)trama);
            if (bitPF == ACTIVO)
            {
              EnviarTramaCtrl( UA, ACTIVO, ACTIVO);
              KActivaContador( timer, T203);
              vs = vr = va = 0;
              KSignal( idProcesoN3, ED_ESTABLEC_IND);
              return( ENLACE_ESTABLECIDO);
            }  /*if (biPF=ACTIVO)*/
            else
              return(NO_WAITING_ACK_RR);
          default:
            /* Si recibimos cualquier otro tipo de trama, liberamos la
               memoria correspondiente: */
            KLiberaMemoria( (UWORD *)trama);
            return(NO_WAITING_ACK_RR);
        } /*switch(tipoTrama)*/
      } /*else if (error != 0)*/
    } /*if (FRecibeDatos(...) == TRUE)*/
```

```
      case ED_DATOS_REQ:
        if (KLeePipe( pipeLecN2N3, (VOID *)&mensaje23, LONG_MENSAJE23, KWAIT)
          == KNO_ERROR)
          if (vs < (va + K))
          {
            /* Para formar el cabecero de la trama I: */
            ns = vs;
            nr = vr;
            FormarCabeceroTrama( &trama, &longTrama, I, INACTIVO, ACTIVO);
            if (FEnviaDatos( idSCC, (void *)(trama), longTrama,
              (void *)(mensaje23.datos), mensaje23.longDatos ) == TRUE)
              numTxTramaI = 1;
            else
              ventanaEsperandoTx = TRUE;
            KActivaContador( timer, T200);
            vs++;
            ventana.header     = trama;
            ventana.lengHeader = longTrama;
            ventana.info       = mensaje23.datos;
            ventana.longInfo   = mensaje23.longDatos;
            /* Paso al estado esperando acuse de recibo de la trama I
               enviada: */
            KActivaContador( timer, T203);
            n200 = 0;
            return(WAITING_ACK_RR);
          } /*if (vs <= va + k))*/
          else
        /* Por aquí no pasa nunca el programa: */
        return(NO_WAITING_ACK_RR);
      case ED_LIBERACION_REQ:
        EnviarTramaCtrl( DISC, INACTIVO, ACTIVO);
        KActivaContador( timer, T200);
        vs = vr = va = 0;
        return(LIBERACION_PENDIENTE);
      case FI_DATOS_ENVIADOS:
        TratarFIdatosEnviados();
        /* Permanecemos en el estado actual: */
        return(NO_WAITING_ACK_RR);
      case ED_UNIDAD_DATOS_REQ:
        TratarEDunidadDatosReq();
        /* Permanecemos en el estado actual: */
        return(NO_WAITING_ACK_RR);
      case GED_UNIDAD_DATOS_REQ:
        TratarGEDunidadDatosReq();
        /* Permanecemos en el estado actual: */
        return(NO_WAITING_ACK_RR);
      case ED_LIBERACION_MEMO_REQ:
        KLeePipe( pipeLecN2N3, (VOID *)&mensaje23, LONG_MENSAJE23, KWAIT);
        KLiberaMemoria( (UWORD *)mensaje23.datos);
        /* Permanecemos en el estado actual: */
        return(NO_WAITING_ACK_RR);
      case GED_LIBERACION_MEMO_REQ:
        KLeePipe( pipeLecN2GED, (VOID *)&mensaje23, LONG_MENSAJE23, KWAIT) ;
        KLiberaMemoria( (UWORD *)mensaje23.datos);
        /* Permanecemos en el estado actual: */
        return(NO_WAITING_ACK_RR);
      case GED_SUPRESION_IET_REQ:
        KSignal( idProcesoN3, ED_LIBERACION_IND);
        LiberaBtramasUI();
        LiberaBtramasI();
        return(IET_NO_ASIGNADO);
      default:
        return(NO_WAITING_ACK_RR);
    } /*switch(signal)*/
  } /*else if (resp == KINTERRUP)*/
} /*NoWaitingAck_RR*/


UWORD NoWaitingAck_RNR( void)
```

```
{
  /*--- Definiciones variables locales a este "estado" (a esta funcion): ---*/
  UWORD signal;
  UWORD tipoTrama;
  UBYTE bitIR, bitPF;
  UBYTE numIETrecibido, numIPASrecibido;
  UWORD resp;
  UWORD error;
  resp = KEsperaContador( timer);
  if (resp == KNO_ERROR)
  {
    if (n200 < N200)
    {
      nr = vr;  /* Para formar el cabecero de la trama RR */
      EnviarTramaCtrl( RR, ACTIVO, ACTIVO);
      KActivaContador( timer, T203);
      n200++;
      esperandoRespuesta = TRUE;
      return(NO_WAITING_ACK_RR);
    }
    else /*if (n200 < N200), es decir cuando (n200 == N200)*/
    {
      LiberaBtramasI();
      EnviarTramaCtrl( DM, ACTIVO, ACTIVO);
      KSignal( idProcesoN3, ED_LIBERACION_IND);
      return(ENLACE_LIBERADO);
    }
  } /*if (resp == KNO_ERROR)*/
  else /*(resp == KINTERRUP)*/
  {
    /* Se ha recibido un signal, por tanto se procesa: */
    KLeeSignal( &signal);
    switch( signal)
    {
      case FI_ERROR_IND:
      case FI_DATOS_RECIBIDOS:
        if (FRecibeDatos( idSCC, (void **)&trama, &longTrama, &error) == TRUE)
        {
          if (error != 0)
          {
            if (trama != NULL)
              KLiberaMemoria( (UWORD *)trama);
            if (error & ERROR_DESACT)
            {
              /*liberar buffers de memoria de mensajes y tramas*/
              LiberaBtx();
              LiberaBtramasI();
              LiberaBtramasCtrl();
              LiberaBtramasUI();
              KSignal( idProcesoN3, ED_LIBERACION_IND);
              return(ENLACE_LIBERADO);
            }
            return(ENLACE_LIBERADO);
          }
          else  /*if (error != 0)*/
          {
            /* Si no hubo error al recibir la trama, se procesa: */
            tipoTrama = IdentificaTrama( trama, longTrama, &bitIR, &bitPF,
              &numIETrecibido, &numIPASrecibido);
            switch(tipoTrama)
            {
              case I:
                if ((ns == vr) && (NR_VALIDO(nr, va, ns)))
                {
                  if (nr == vs)
                  {
                    /* Si se cumple esta condición, la variable
                       nr es correcta: */
                    va = nr;
                  }
```

```
        vr++;
        FormarMensaje23( &mensaje23, ED_DATOS_IND,
          longTrama, trama);
        KEscribePipe( pipeEscN2N3, (void *)&mensaje23,
          LONG_MENSAJE23, KWAIT);
        KSignal( idProcesoN3, ED_DATOS_IND);
        nr = vr;  /* Para formar el cabecero de la trama RR */
        /* Si bitPF = 0 ó 1, se enviará una respuesta RR (bitIR = 1)
           y bitPF = 1: */
        EnviarTramaCtrl( RR, INACTIVO, INACTIVO);
        KActivaContador( timer, T203);
        n200 = 0;
      }
      else /*if ((ns == vr) && (NR_VALIDO(nr, va, ns)))*/
      {
        /* Se descarta las tramas I cuyo ns no sea igual a vr
           o cuando su nr no sea válido: */
        KLiberaMemoria( (UWORD *)trama);
      }
      /* En cualquier caso, permanecemos en el estado actual: */
      return(NO_WAITING_ACK_RNR);
    case RNR:
      KLiberaMemoria( (UWORD *)trama);
      if ((bitPF == ACTIVO) && (NR_VALIDO(nr, va, ns)))
      {
        if (nr == vs)
          /* Significa que la variable nr recibida es correcta: */
          va = nr;
        if (bitIR == INACTIVO)
        {
          /* Si la trama RNR recibida es una instrucción, se enviará
             una trama RR respuesta: */
          nr = vr; /* Para formar el cabecero de la trama RR */
          EnviarTramaCtrl( RR, ACTIVO, ACTIVO);
          n200 = 0;
        }
        else
          esperandoRespuesta = FALSE;
        KActivaContador( timer, T203);
        return(NO_WAITING_ACK_RNR);
      }
      else
        return(NO_WAITING_ACK_RNR);
    case RR:
      KLiberaMemoria( (UWORD *)trama);
      if (NR_VALIDO(nr, va, ns) && (nr == vs))
      {
        /*-------------------------------------------------------------
          Significa que la variable nr es correcta, pués en este
            estado no se espera confirmación, sino que la variable
            nr de la trama recibida sea correcta:
              - Si el bufferTramasI no está vacío, envío una trama I
                  y se añade al bufferTx y a la estructura ventana
                  y pasamos al estado WaitingAck_RR
              - Si el buffer está vacío pasamos al estado
                  NoWaitingAck_RR
          Si la variable nr (caso imposible según nuestro algoritmo)
            no es correcta, permanecemos en el estado actual
        -------------------------------------------------------------*/
        va = nr;
        if ((frenteBtramasI != finalBtramasI) && (vs < (va + K) ))
        {
          /* Para formar el cabecero de la trama I: */
          nr = vr;
          ns = vs;
          FormarCabeceroTrama( &trama, &longTrama, I,
            INACTIVO, ACTIVO);
          if (FEnviaDatos( idSCC, (void *)(trama), longTrama,
            (void *)(bufferTramasI[frenteBtramasI].info),
            bufferTramasI[frenteBtramasI].longInfo) == TRUE)
```

```
                {
                  numTxTramaI = 1;
                  GuardaBtx( TIPO_I, trama,
                    bufferTramasI[frenteBtramasI].info);
                }
                else
                  numTxTramaI = 0;
                vs++;
                esperandoRespuesta = TRUE;
                ventana.header     = trama;
                ventana.lengHeader = longTrama;
                ventana.info       = bufferTramasI[frenteBtramasI].info;
                ventana.longInfo = bufferTramasI[frenteBtramasI].longInfo;
                if (++frenteBtramasI == LONG_BUF_TRAMAS_I)
                  frenteBtramasI = 0;
                ventanaEsperandoTx = TRUE;
                KActivaContador( timer, T200);
                n200 = 0;
                return(WAITING_ACK_RR);
              }/*if((frenteBtramasI != finalBtramasI) && (vs<=(va + K)))*/
              else
                return(NO_WAITING_ACK_RR);
            } /*if (NR_VALIDO(nr, va, ns) && (nr == vs))*/
            return(NO_WAITING_ACK_RNR);
          case UI:
          case TIPO_GED_UI:
            TratarTramaUIrecibida( numIETrecibido, numIPASrecibido);
            return(NO_WAITING_ACK_RNR);
          case DISC:
            KLiberaMemoria( (UWORD*)trama);
            if (bitPF == ACTIVO)
            {
              EnviarTramaCtrl( UA, ACTIVO, ACTIVO);
              vs = vr = va = 0;
              KSignal( idProcesoN3, ED_LIBERACION_IND);
              return(ENLACE_LIBERADO);
            }  /*if (biPF=ACTIVO)*/
            return(NO_WAITING_ACK_RNR);
          case SABME:
            KLiberaMemoria( (UWORD *)trama);
            if (bitPF == ACTIVO)
            {
              EnviarTramaCtrl( UA, ACTIVO, ACTIVO);
              KActivaContador( timer, T203);
              vs = vr = va = 0;
              LiberaBtramasI();
              LiberaVentana();
              KSignal( idProcesoN3, ED_ESTABLEC_IND);
              return( ENLACE_ESTABLECIDO);
            }  /*if (biPF=ACTIVO)*/
            else
              return(NO_WAITING_ACK_RNR);
          default:
            /* Si recibimos cualquier otro tipo de trama, liberamos la
               memoria correspondiente: */
            KLiberaMemoria( (UWORD *)trama);
            return(NO_WAITING_ACK_RNR);
        } /*switch(tipoTrama)*/
      } /*else if (error != 0)*/
    } /*if (FRecibeDatos(...) == TRUE)*/
  case FI_DATOS_ENVIADOS:
    TratarFIdatosEnviados();
    /* Permanecemos en el estado actual: */
    return(NO_WAITING_ACK_RNR);
  case ED_UNIDAD_DATOS_REQ:
    TratarEDunidadDatosReq();
    /* Permanecemos en el estado actual: */
    return(NO_WAITING_ACK_RNR);
  case GED_UNIDAD_DATOS_REQ:
    TratarGEDunidadDatosReq();
```

```
{
  /* Se ha recibido un signal, por tanto se procesa: */
  KLeeSignal( &signal);
  switch( signal)
  {
    case FI_ERROR_IND:
    case FI_DATOS_RECIBIDOS:
      if (FRecibeDatos( idSCC, (void **)&trama, &longTrama, &error) == TRUE)
      {
        if (error != 0)
        {
          if (trama != NULL)
            KLiberaMemoria( (UWORD *)trama);
          if (error & ERROR_DESACT)
          {
            /*liberar buffers de memoria de mensajes y tramas*/
            LiberaBtx();
            LiberaBtramasI();
            LiberaVentana();
            LiberaBtramasCtrl();
            LiberaBtramasUI();
            KSignal( idProcesoN3, ED_LIBERACION_IND);
            return(ENLACE_LIBERADO);
          }
        }
        else  /*if (error != 0)*/
        {
          /* Si no hubo error al recibir la trama, se procesa: */
          tipoTrama = IdentificaTrama( trama, longTrama, &bitIR, &bitPF,
            &numIETrecibido, &numIPASrecibido);
          switch(tipoTrama)
          {
            case I:
              if ((ns == vr) && (NR_VALIDO(nr, va, ns)))
              {
                if (nr == vs)
                {
                  /* Si se cumple esta condición, está confirmando
                     la última trama I enviada: */
                  va = nr;
                  confirmando = TRUE;
                  LiberaVentana();
                }
                vr++;
                FormarMensaje23( &mensaje23, ED_DATOS_IND,
                  longTrama, trama);
                KEscribePipe( pipeEscN2N3, (void *)&mensaje23,
                  LONG_MENSAJE23, KWAIT);
                KSignal( idProcesoN3, ED_DATOS_IND);
                nr = vr;  /* Para formar el cabecero de la trama RR */
                /* Si bitPF = 0 ó 1, se enviará una respuesta RR (bitIR = 1)
                   y bitPF = 1: */
                EnviarTramaCtrl( RR, ACTIVO, ACTIVO);
                KActivaContador( timer, T203);
                /* Si la trama I recibida está confirmando la última trama
                   I enviada, pasamos al estado NoWaitingAck_RNR: */
                if (confirmando == TRUE)
                  return(NO_WAITING_ACK_RNR);
              }
              else /*if ((ns == vr) && (NR_VALIDO(nr, va, ns)))*/
              {
                /* Se descarta las tramas I cuyo ns no sea igual a vr
                   o cuando su nr no sea válido: */
                KLiberaMemoria( (UWORD *)trama);
              }
              /* En otro caso, permanecemos en el estado actual: */
              return(WAITING_ACK_RNR);
            case REJ:
              KLiberaMemoria( (UWORD *)trama);
              if (NR_VALIDO(nr, va, ns))
```

```
    {
      /* Si la trama REJ recibida es una intrucción, se enviará
         una trama RR respuesta: */
      if ((bitIR == INACTIVO) && (bitPF == ACTIVO))
        EnviarTramaCtrl( RR, ACTIVO, ACTIVO);
      /* En cualquier caso se reenviará la última trama I
         (trama I de la ventana): */
      if (FEnviaDatos( idSCC, (void *)(ventana.header),
        ventana.lengHeader, (void *)(ventana.info),
        ventana.longInfo) == TRUE)
      {
        numTxTramaI = 1;
        GuardaBtx( TIPO_I, ventana.header, ventana.info);
      }
      else
        numTxTramaI = 0;
      ventanaEsperandoTx = TRUE;
      esperandoRespuesta = TRUE;
      n200 = 0;
      KActivaContador( timer, T200);
      return(WAITING_ACK_RR);
    }
    else /*if (NR_VALIDO(nr, va, ns))*/
      return(WAITING_ACK_RNR);
  case RR:
    if (NR_VALIDO(nr, va, ns) && (nr == vs))
    {
      /*-------------------------------------------------------------
        Significa que está confirmando última trama enviada:
          - Si el bufferTramasI no está vacío, envío una trama I
              y se añade al bufferTx y a la estructura ventana
              y pasamos al estado WaitingAck_RR
          - Si el buffer está vacío pasamos al estado
              NoWaitingAck_RR
        Si la variable nr no está confirmando, pasamos al estado
          WaitingAck_RR
      -------------------------------------------------------------*/
      va = nr;
      esperandoRespuesta = FALSE;
      if ((frenteBtramasI != finalBtramasI) && (vs < (va + K) ))
      {
        /* Para formar el cabecero de la trama I: */
        nr = vr;
        ns = vs;
        FormarCabeceroTrama( &trama, &longTrama, I,
          INACTIVO, ACTIVO);
        if (FEnviaDatos( idSCC, (void *)(trama), longTrama,
          (void *)(bufferTramasI[frenteBtramasI].info),
          bufferTramasI[frenteBtramasI].longInfo) == TRUE)
        {
          numTxTramaI = 1;
          GuardaBtx( TIPO_I, trama,
            bufferTramasI[frenteBtramasI].info);
        }
        else
          numTxTramaI = 0;
        vs++;
        ventana.header     = trama;
        ventana.lengHeader = longTrama;
        ventana.info       = bufferTramasI[frenteBtramasI].info;
        ventana.longInfo = bufferTramasI[frenteBtramasI].longInfo;
        if (++frenteBtramasI == LONG_BUF_TRAMAS_I)
          frenteBtramasI = 0;
        ventanaEsperandoTx = TRUE;
        KActivaContador( timer, T200);
        n200 = 0;
        return(WAITING_ACK_RR);
      }/*if((frenteBtramasI != finalBtramasI) && (vs<=(va + K)))*/
      else
        return(NO_WAITING_ACK_RR);
```

```
    } /*if (NR_VALIDO(nr, va, ns) && (nr == vs))*/
    return(WAITING_ACK_RNR);
  case RNR:
    KLiberaMemoria( (UWORD *)trama);
    if (NR_VALIDO(nr, va, ns))
      if (nr == vs)
      {
        /* Si (nr=vs) está confirmando la última trama I enviada
           y pasamos al estado NoWaitingAck_RNR: */
        va = nr;
        esperandoRespuesta = FALSE;
        n200 = 0;
        KActivaContador( timer, T203);
        if (bitPF == ACTIVO)
        {
          EnviarTramaCtrl( RR, ACTIVO, ACTIVO);
          KActivaContador( timer, T200);
        }
        else
          KActivaContador( timer, T203);
        return(NO_WAITING_ACK_RNR);
      }
      else
      {
        /*Si no está confirmando, permanecemos en el estado actual*/
        n200 = 0;
        if (bitPF == ACTIVO)
        {
          EnviarTramaCtrl( RR, ACTIVO, ACTIVO);
          KActivaContador( timer, T200);
        }
        else
          KActivaContador( timer, T203);
        return( NO_WAITING_ACK_RNR);
      }
    else /* if(NR_VALIDO(nr, va, ns)) */
      return(WAITING_ACK_RNR);
  case UI:
  case TIPO_GED_UI:
    TratarTramaUIrecibida( numIETrecibido, numIPASrecibido);
    return(WAITING_ACK_RNR);
  case DISC:
    KLiberaMemoria( (UWORD *)trama);
    if (bitPF == ACTIVO)
    {
      EnviarTramaCtrl( UA, ACTIVO, ACTIVO);
      vs = vr = va = 0;
      LiberaBtramasI();
      LiberaVentana();
      KSignal( idProcesoN3, ED_LIBERACION_IND);
      return(ENLACE_LIBERADO);
    } /*if (biPF=ACTIVO)*/
    return(WAITING_ACK_RNR);
  case SABME:
    KLiberaMemoria( (UWORD *)trama);
    if (bitPF == ACTIVO)
    {
      EnviarTramaCtrl( UA, ACTIVO, ACTIVO);
      KActivaContador( timer, T203);
      vs = vr = va = 0;
      LiberaBtramasI();
      LiberaVentana();
      KSignal( idProcesoN3, ED_ESTABLEC_IND);
      return( ENLACE_ESTABLECIDO);
    }  /*if (biPF=ACTIVO)*/
    else
      return(WAITING_ACK_RNR);
  default:
    /* Si recibimos cualquier otro tipo de trama, liberamos la
       memoria correspondiente: */
```

```
                KLiberaMemoria( (UWORD *)trama);
                return(WAITING_ACK_RNR);
              } /*switch(tipoTrama)*/
            } /*else if (error != 0)*/
          } /*if (FRecibeDatos(...) == TRUE)*/
        case FI_DATOS_ENVIADOS:
          /* En cualquier caso permanecemos en el estado actual: */
          return(WAITING_ACK_RNR);
        case ED_UNIDAD_DATOS_REQ:
          TratarEDunidadDatosReq();
          /* Permanecemos en el estado actual: */
          return(WAITING_ACK_RNR);
        case GED_UNIDAD_DATOS_REQ:
          TratarGEDunidadDatosReq();
          /* Permanecemos en el estado actual: */
          return(WAITING_ACK_RNR);
        case ED_DATOS_REQ:
          TratarEDdatosReq();
          /* Permanecemos en el estado actual: */
          return(WAITING_ACK_RNR);
        case ED_LIBERACION_REQ:
          LiberaBtramasI();
          LiberaVentana();
          /*-------------------------------------------------------------------
             Libero la trama de la ventana, sólo en los casos que estoy
             esperando acuse de recibo y por tanto, existe una trama I en la
             ventana de transmisión
          -------------------------------------------------------------------*/
          EnviarTramaCtrl( DISC, INACTIVO, ACTIVO);
          KActivaContador( timer, T200);
          vs = vr = va = 0;
          return(LIBERACION_PENDIENTE);
        case ED_LIBERACION_MEMO_REQ:
          KLeePipe( pipeLecN2N3, (VOID *)&mensaje23, LONG_MENSAJE23, KWAIT);
          KLiberaMemoria( (UWORD *)mensaje23.datos);
          /* Permanecemos en el estado actual: */
          return(WAITING_ACK_RNR);
        case GED_LIBERACION_MEMO_REQ:
          KLeePipe( pipeLecN2GED, (VOID *)&mensaje23, LONG_MENSAJE23, KWAIT) ;
          KLiberaMemoria( (UWORD *)mensaje23.datos);
          /* Permanecemos en el estado actual: */
          return(WAITING_ACK_RNR);
        case GED_SUPRESION_IET_REQ:
          KSignal( idProcesoN3, ED_LIBERACION_IND);
          LiberaBtramasUI();
          LiberaBtramasI();
          LiberaVentana();
          return(IET_NO_ASIGNADO);
        default:
          return(WAITING_ACK_RNR);
      } /*switch(signal)*/
    } /*else if (resp == KINTERRUP)*/
} /*WaitingAck_RNR*/


UWORD WaitingAck_RR( void)
{
  /*--- Definiciones variables locales a este "estado" (a esta funcion): ---*/
  UWORD signal;
  UWORD tipoTrama;
  UBYTE bitIR, bitPF;
  UBYTE numIETrecibido, numIPASrecibido;
  UWORD resp;
  UWORD error;          /* Error devuelto por el nivel físico */
  resp = KEsperaContador( timer);
  if (resp == KNO_ERROR)
  {
    if (ventanaEsperandoTx == TRUE)
    {
```

```
      if (numTxTramaI < N202)
      {
        if (EnviarTramaVentana() == TRUE)
          numTxTramaI++;
        KActivaContador( timer, T200);
        return(WAITING_ACK_RR);
      }
      else
      {
        LiberaBtramasCtrl();
        LiberaBtramasI();
        LiberaVentana();
        KSignal( idProcesoN3, ED_LIBERACION_IND);
        return( ENLACE_LIBERADO);
      }
    }
    else /*if (ventanaEsperandoTx == TRUE)*/
    {
      if (n200 < N200)
      {
        nr = vr;   /* Para formar el cabecero de la trama RR */
        EnviarTramaCtrl(  RR, ACTIVO, ACTIVO);
        n200++;
        KActivaContador( timer, T200);
        esperandoRespuesta = TRUE;
        return(WAITING_ACK_RR);
      }
      else /*if (n200 < N200)*/
      {
        LiberaBtramasI();
        LiberaVentana();
        EnviarTramaCtrl( DM, ACTIVO, ACTIVO);
        KSignal( idProcesoN3, ED_LIBERACION_IND);
        return(ENLACE_LIBERADO);
      }
    }
  }
  else /*if (resp == KNO_ERROR)*/
  {
    /*Se ha recibido un signal, por tanto se procesa:*/
    KLeeSignal( &signal);
    switch( signal)
    {
      case FI_ERROR_IND:
      case FI_DATOS_RECIBIDOS:
        if (FRecibeDatos( idSCC, (void **)&trama, &longTrama, &error) == TRUE)
        {
          if (error != 0)
          {
            if (trama != NULL)
              KLiberaMemoria( (UWORD *)trama);
            if (error & ERROR_DESACT)
            {
              /*liberar buffers de memoria de mensajes y tramas*/
              LiberaBtx();
              LiberaBtramasI();
              LiberaVentana();
              LiberaBtramasCtrl();
              LiberaBtramasUI();
              KSignal( idProcesoN3, ED_LIBERACION_IND);
              return(ENLACE_LIBERADO);
            }
          }
          else  /*if (error != 0)*/
          {
            /* Si no hubo error al recibir la trama, se procesa: */
            tipoTrama = IdentificaTrama( trama, longTrama, &bitIR, &bitPF,
              &numIETrecibido, &numIPASrecibido);
            switch(tipoTrama)
            {
```

```
case I:
  if ((ns == vr) && (NR_VALIDO(nr, va, ns)))
  {
    FormarMensaje23( &mensaje23, ED_DATOS_IND,
      longTrama, trama);
    KEscribePipe( pipeEscN2N3, (void *)&mensaje23,
      LONG_MENSAJE23, KWAIT);
    KSignal( idProcesoN3, ED_DATOS_IND);
    vr++;
    if (nr == vs)
    {
      /* Si se cumple esta condición, está confirmando
         la última trama I enviada: */
      va = nr;
      LiberaVentana();
      if ((frenteBtramasI != finalBtramasI) && (vs < (va + K)))
      {
        /*-------------------------------------------------------
          - Si el bufferTramasI no está vacío, envío una trama I
              y se añade al bufferTx y a la estructura ventana
              y permanecemos en el estado WaitingAck_RR
          - Si el buffer está vacío, es decir, no hay datos en
              espera, pasamos al estado NoWaitingAck_RR
        -------------------------------------------------------*/
        /* Para formar el cabecero de la trama I: */
        nr = vr;
        ns = vs;
        esperandoRespuesta = TRUE;
        FormarCabeceroTrama( &trama, &longTrama, I,
          INACTIVO, ACTIVO);
        if (FEnviaDatos( idSCC, (void *)(trama), longTrama,
          (void *)(bufferTramasI[frenteBtramasI].info),
          bufferTramasI[frenteBtramasI].longInfo) == TRUE)
        {
          numTxTramaI = 1;
          GuardaBtx( TIPO_I, trama,
            bufferTramasI[frenteBtramasI].info);
        }
        else
          numTxTramaI = 0;
        vs++;
        ventana.header     = trama;
        ventana.lengHeader = longTrama;
        ventana.info       = bufferTramasI[frenteBtramasI].info;
        ventana.longInfo   =
          bufferTramasI[frenteBtramasI].longInfo;
        if (++frenteBtramasI == LONG_BUF_TRAMAS_I)
          frenteBtramasI = 0;
        ventanaEsperandoTx = TRUE;
        n200 = 0;
        KActivaContador( timer, T200);
        return(WAITING_ACK_RR);
      }
      else /*if((frenteBtramasI!=finalBtramasI) && (vs<= ...)))*/
      {
        /* No hay datos que transmitir; por tanto pasamos al
           estado NoWaitingAck_RR: */
        nr = vr;  /* Para formar el cabecero de la trama RR */
        /* Si bitPF = 0 ó 1, se enviará una respuesta RR
           (bitIR = 1) y bitPF = 1: */
        EnviarTramaCtrl( RR, ACTIVO, ACTIVO);
        KActivaContador( timer, T203);
        return(NO_WAITING_ACK_RR);
      }
    }
    else /*if (nr == vs)*/
    {
      /* Si no confirma la última trama I enviada, permanecemos
         en el estado WaitingAck_RR: */
      nr = vr;  /* Para formar el cabecero de la trama RR */
```

```
            /* Si bitPF = 0 ó 1, se enviará una respuesta RR
               (bitIR = 1) y bitPF = 1: */
            EnviarTramaCtrl( RR, ACTIVO, ACTIVO);
            KActivaContador( timer, T203);
            return(WAITING_ACK_RR);
          }
        }
        else /*if ((ns == vr) && (NR_VALIDO(nr, va, ns)))*/
        {
          /* Si variables ns y nr de la trama I recibida no son
             correctas, descartamos la trama: */
          KLiberaMemoria( (UWORD *)trama);
          return(WAITING_ACK_RR);
        }
      case RR:
        KLiberaMemoria( (UWORD *)trama);
        if (NR_VALIDO(nr, va, ns) && (nr == vs))
        {
          /*-------------------------------------------------------------
            Significa que está confirmando última trama enviada:
              - Si el bufferTramasI no está vacío, envío una trama I
                  y se añade al bufferTx y a la estructura ventana
                  y permanecemos en el estado WaitingAck_RR
              - Si el buffer está vacío pasamos al estado
                  NoWaitingAck_RR
            Si la variable nr no está confirmando, permanecemos
              también en el estado WaitingAck_RR
          -------------------------------------------------------------*/
          va = nr;
          esperandoRespuesta = FALSE;
          if ((frenteBtramasI != finalBtramasI) && (vs < (va + K) ))
          {
            /* Para formar el cabecero de la trama I: */
            nr = vr;
            ns = vs;
            esperandoRespuesta = TRUE;
            FormarCabeceroTrama( &trama, &longTrama, I,
              INACTIVO, ACTIVO);
            if (FEnviaDatos( idSCC, (void *)(trama), longTrama,
              (void *)(bufferTramasI[frenteBtramasI].info),
              bufferTramasI[frenteBtramasI].longInfo) == TRUE)
            {
              numTxTramaI = 1;
              GuardaBtx( TIPO_I, trama,
              bufferTramasI[frenteBtramasI].info);
            }
            else
              numTxTramaI = 0;
            vs++;
            ventana.header     = trama;
            ventana.lengHeader = longTrama;
            ventana.info       = bufferTramasI[frenteBtramasI].info;
            ventana.longInfo   = bufferTramasI[frenteBtramasI].longInfo;
            if (++frenteBtramasI == LONG_BUF_TRAMAS_I)
              frenteBtramasI = 0;
            ventanaEsperandoTx = TRUE;
            KActivaContador( timer, T200);
            n200 = 0;
            return(WAITING_ACK_RR);
          }/*if((frenteBtramasI!=finalBtramasI) && (vs <= (va+K)))*/
          else
            return(NO_WAITING_ACK_RR);
        } /*if (NR_VALIDO(nr, va, ns) && (nr == vs))*/
        else
          return(WAITING_ACK_RR);
      case RNR:
        KLiberaMemoria( (UWORD *)trama);
        if (NR_VALIDO(nr, va, ns))
        {
          if (nr == vs)
```

```
        {
          /* Si (nr=vs) está confirmando la última trama I enviada y
             pasamos al estado NoWaitingAck_RNR: */
          va = nr;
          esperandoRespuesta = FALSE;
          n200 = 0;
          KActivaContador( timer, T203);
          if (bitPF == ACTIVO)
          {
            EnviarTramaCtrl( RR, ACTIVO, ACTIVO);
            KActivaContador( timer, T200);
          }
          else
            KActivaContador( timer, T203);
          return(NO_WAITING_ACK_RNR);
        }
        else
        {
          /* Si no está confirmando, pasamos al estado
             WaitingAck_RNR*/
          n200 = 0;
          KActivaContador( timer, T203);
          if (bitPF == ACTIVO)
          {
            EnviarTramaCtrl( RR, ACTIVO, ACTIVO);
            KActivaContador( timer, T200);
          }
          else
            KActivaContador( timer, T203);
          return(WAITING_ACK_RNR);
        }
      }
      else
      /* Si la trama RNR recibida no posee un nr válido, se ignora
         y permanecemos en el estado actual: */
         return(WAITING_ACK_RR);
    case REJ:
      if (NR_VALIDO(nr, va, ns))
      {
        /* Si la trama REJ recibida es una intrucción, se enviará
           una trama RR respuesta: */
        if ((bitIR == INACTIVO) && (bitPF == ACTIVO))
          EnviarTramaCtrl( RR, ACTIVO, ACTIVO);
        /* En cualquier caso se reenviará la última trama I
           (trama I de la ventana): */
        if (FEnviaDatos( idSCC, (void *)(ventana.header),
          ventana.lengHeader, (void *)(ventana.info),
          ventana.longInfo) == TRUE)
        {
          numTxTramaI = 1;
          GuardaBtx( TIPO_I, ventana.header, ventana.info);
        }
        else
          numTxTramaI = 0;
        ventanaEsperandoTx = TRUE;
        esperandoRespuesta = TRUE;
        n200 = 0;
        KActivaContador( timer, T200);
        return(WAITING_ACK_RR);
      }
      else /*if (NR_VALIDO(nr, va, ns))*/
        return(WAITING_ACK_RR);
    case UI:
    case TIPO_GED_UI:
      TratarTramaUIrecibida( numIETrecibido, numIPASrecibido);
      return(WAITING_ACK_RR);
    case DISC:
      KLiberaMemoria( (UWORD *)trama);
      if (bitPF == ACTIVO)
      {
```

```
            EnviarTramaCtrl( UA, ACTIVO, ACTIVO);
            vs = vr = va = 0;
            LiberaBtramasI();
            LiberaVentana();
            KSignal( idProcesoN3, ED_LIBERACION_IND);
            return(ENLACE_LIBERADO);
          } /*if (biPF=ACTIVO)*/
          return(WAITING_ACK_RR);
        case SABME:
          KLiberaMemoria( (UWORD *)trama);
          if (bitPF == ACTIVO)
          {
            EnviarTramaCtrl( UA, ACTIVO, ACTIVO);
            KActivaContador( timer, T203);
            vs = vr = va = 0;
            LiberaBtramasI();
            LiberaVentana();
            KSignal( idProcesoN3, ED_ESTABLEC_IND);
            return( ENLACE_ESTABLECIDO);
          }  /*if (biPF=ACTIVO)*/
          else
            return(WAITING_ACK_RR);
        default:
          /* Si recibimos cualquier otro tipo de trama, liberamos la
             memoria correspondiente: */
          KLiberaMemoria( (UWORD *)trama);
          return(WAITING_ACK_RR);
      } /*switch(tipoTrama)*/
    } /*else if (error != 0)*/
  } /*if (FRecibeDatos(...) == TRUE)*/
case ED_LIBERACION_REQ:
  LiberaBtramasI();
  LiberaVentana();
  /*-------------------------------------------------------------------
     Libero la trama de la ventana, sólo en los casos que estoy
     esperando acuse de recibo y por tanto, existe una trama I en la
     ventana de transmisión
  -------------------------------------------------------------------*/
  EnviarTramaCtrl( DISC, INACTIVO, ACTIVO);
  KActivaContador( timer, T200);
  vs = vr = va = 0;
  return(LIBERACION_PENDIENTE);
case ED_DATOS_REQ:
  TratarEDdatosReq();
  /* Permanecemos en el estado actual: */
  return(WAITING_ACK_RR);
case FI_DATOS_ENVIADOS:
  TratarFIdatosEnviados();
  /* Permanecemos en el estado actual: */
  return(WAITING_ACK_RR);
case ED_UNIDAD_DATOS_REQ:
  TratarEDunidadDatosReq();
  /* Permanecemos en el estado actual: */
  return(WAITING_ACK_RR);
case GED_UNIDAD_DATOS_REQ:
  TratarGEDunidadDatosReq();
  /* Permanecemos en el estado actual: */
  return(WAITING_ACK_RR);
case ED_LIBERACION_MEMO_REQ:
  KLeePipe( pipeLecN2N3, (VOID *)&mensaje23, LONG_MENSAJE23, KWAIT);
  KLiberaMemoria( (UWORD *)mensaje23.datos);
  /* Permanecemos en el estado actual: */
  return(WAITING_ACK_RR);
case GED_LIBERACION_MEMO_REQ:
  KLeePipe( pipeLecN2GED, (VOID *)&mensaje23, LONG_MENSAJE23, KWAIT) ;
  KLiberaMemoria( (UWORD *)mensaje23.datos);
  /* Permanecemos en el estado actual: */
  return(WAITING_ACK_RR);
case GED_SUPRESION_IET_REQ:
  KSignal( idProcesoN3, ED_LIBERACION_IND);
```

```
        LiberaBtramasUI();
        LiberaBtramasI();
        LiberaVentana();
        return(IET_NO_ASIGNADO);
      default:
        /* Permanecemos en el estado actual: */
        return(WAITING_ACK_RR);
    } /*switch(signal)*/
  }  /*else if (resp == KINTERRUP)*/
} /*WaitingAck_RR*/


UWORD LiberacionPendiente( void)
{
  /*--- Definiciones variables locales a este "estado" (a esta funcion): ---*/
  UWORD tipoTrama;
  UBYTE bitIR, bitPF;
  UBYTE numIETrecibido, numIPASrecibido;
  UWORD error;
  UWORD resp;
  UWORD n200 = 0;  /*Inicializo número de retransmisiones de la trama DISC*/
  while( n200 < N200)  /*Hasta N200 = 3 retransmisiones de la trama DISC*/
  {
    resp = KEsperaContador(timer);
    if (resp == KNO_ERROR)  /*terminó el temporizador*/
    {
      EnviarTramaCtrl( DISC, INACTIVO, ACTIVO);
      KActivaContador( timer, T200);
      vs = vr = va = 0;
      n200++;
      /*-------------------------------------------------------------------------
         Permanecemos en el estado actual (LiberacionPendiente)
         hasta completar las N200 retransmisiones, o al llegar la trama
         adecuada, por evolución a otro estado
      -------------------------------------------------------------------------*/
      continue;  /*Salta a evaluar denuevo la condición del while*/
    } /*if (resp == KNO_ERROR)*/
    if (resp == KINTERRUP)
    {
      KLeeSignal(&signal);
      switch(signal)
      {
        case FI_ERROR_IND:
        case FI_DATOS_RECIBIDOS:
          if (FRecibeDatos( idSCC, (void **)&trama, &longTrama, &error)
            == TRUE)
          {
            if (error != 0)
            {
              if (trama != NULL)
                KLiberaMemoria( (UWORD *)trama);
              if (error & ERROR_DESACT)
              {
                /*liberar memoria dinámica asociada a los buffers*/
                LiberaBtx();
                LiberaBtramasUI();
                LiberaBtramasCtrl();
                KSignal( idProcesoN3, ED_LIBERACION_CONF);
                return(ENLACE_LIBERADO);
              }
            }
            else  /*if (error != 0)*/
            {
              /* Si no hubo error al recibir la trama, se procesa: */
              tipoTrama = IdentificaTrama( trama, longTrama, &bitIR, &bitPF,
                &numIETrecibido, &numIPASrecibido);
              switch(tipoTrama)
              {
                case UI:
```

```
            case TIPO_GED_UI:
              TratarTramaUIrecibida( numIETrecibido, numIPASrecibido);
              continue;
            case UA:
            case DM:
              KLiberaMemoria( (UWORD*)trama);
              if (bitPF == ACTIVO)
              {
                KSignal( idProcesoN3, ED_LIBERACION_CONF);
                return(ENLACE_LIBERADO);
              }
              else continue;
            case DISC:
              KLiberaMemoria( (UWORD*)trama);
              if (bitPF == ACTIVO)
              {
                /*-------------------------------------------------------
                  Respondemos enviando una trama UA, pero no pasaremos al
                  estado EnlaceLiberado hasta que no recibamos una
                  trama UA
                -------------------------------------------------------*/
                EnviarTramaCtrl( UA, ACTIVO, ACTIVO);
                KSignal( idProcesoN3, ED_LIBERACION_CONF);
              }
              continue;
            case SABME:
              KLiberaMemoria( (UWORD*)trama);
              if (bitPF == ACTIVO)
              {
                EnviarTramaCtrl( DM, INACTIVO, INACTIVO);
                KSignal( idProcesoN3, ED_LIBERACION_CONF);
                return(ENLACE_LIBERADO);
              }
              else continue;
            default:
            /* Si recibimos cualquier otro tipo de trama, liberamos la
               memoria correspondiente: */
              KLiberaMemoria( (UWORD *)trama);
              continue;
          } /*switch(tipoTrama)*/
        } /*else (error!=0)*/
      }
      else /*if (FRecibeDatos(...)==TRUE)*/
        /* Permanecemos en el estado actual: */
        continue;
    case FI_DATOS_ENVIADOS:
      TratarFIdatosEnviados();
      /* Permanecemos en el estado actual: */
      continue;
    case ED_UNIDAD_DATOS_REQ:
      TratarEDunidadDatosReq();
      /* Permanecemos en el estado actual: */
      continue;
    case GED_UNIDAD_DATOS_REQ:
      TratarGEDunidadDatosReq();
      /* Permanecemos en el estado actual: */
      continue;
    case ED_LIBERACION_MEMO_REQ:
      KLeePipe( pipeLecN2N3, (VOID *)&mensaje23, LONG_MENSAJE23, KWAIT);
      KLiberaMemoria( (UWORD *)mensaje23.datos);
      /* Permanecemos en el estado actual: */
      continue;
    case GED_LIBERACION_MEMO_REQ:
      KLeePipe( pipeLecN2GED, (VOID *)&mensaje23, LONG_MENSAJE23, KWAIT) ;
      KLiberaMemoria( (UWORD *)(mensaje23.datos));
      /* Permanecemos en el estado actual: */
      continue;
    case GED_SUPRESION_IET_REQ:
      KSignal( idProcesoN3, ED_LIBERACION_CONF);
      LiberaBtramasUI();
```

```
          return(IET_NO_ASIGNADO);
        default:
          /* Permanecemos en el estado actual: */
          continue;
      } /*switch(signal)*/
    } /*if (resp == KINTERRUP)*/
  } /*while(n200 < N200)*/
  /*-------------------------------------------------------------------------
     Si no se recibió una respuesta correcta después de enviar N200
     retransmisiones de la trama DISC, se pasa al estado ENLACE_LIBERADO
     y se indica al nivel 3 con el signal ED_LIBERACION_CONF.
  -------------------------------------------------------------------------*/
  KSignal( idProcesoN3, ED_LIBERACION_CONF);
  return(ENLACE_LIBERADO);
} /*LiberacionPendiente*/

/*------------------- Fin definiciones de funciones ------------------------*/
```

```
/*****************************************************************************/
/*                              GED_ET.C                                     */
/*          TRABAJO FIN DE CARRERA  SANTIAGO MARTIN ROMANI                   */
/*      IMPLEMENTACION GESTOR ENLACE DE DATOS PARA EQUIPO TERMINAL           */
/*****************************************************************************/

#define UNIX

/*------------ FICHEROS INCLUIDOS: ----------------------------------------*/
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <alloc.h>
#include "basico.h"    /* ..\kernel\basico.h */
#include "kernel.h"    /* ..\kernel\kernel.h */
#include "inicio.h"
#include "nivel2.hp"
#include "n2n3.h"
#include "ged.hp"



/*------------ DECLARACION DE VARIABLES GLOBALES: --------------------------*/
UWORD signal;
UWORD timerGED_ET;
/* Identificador de la entidad de nivel 2 de terminal de red
   con la que nos comunicaremos: */
UBYTE idProcesoN2;
UBYTE idProcesoGED;
UBYTE *trama;
UWORD longTrama;

UBYTE tipoMensaje;
UBYTE indicadorAccion;
UWORD numRef;
UWORD seed;



/*-------- Valor del Identificador de Equipo Terminal asignado: -------------
   Cuando tome valor 127 (IET_DE_GRUPO) significará que la entidad de nivel 2
   del lado del Equipo Terminal debe estar en el estado IETnoAsignado
-----------------------------------------------------------------------------*/
UBYTE numIETasignado;

MENSAJE23 mensaje23;


/*------- Pipes de comunicación con la entidad de nivel 2 del lado ET: -------
  Sobre el pipe de escritura se mandan los mensajes a la entidad de nivel 2
  del lado del Equipo Terminal, a través del pipe de lectura se reciben los
  mensajes procedentes de dicha entidad de nivel 2:
-----------------------------------------------------------------------------*/
UWORD pipeEscGED, pipeLecGED;

UWORD vecNumRef[LONG_VEC_NUM_REF];
UWORD indiceVecNumRef;

UWORD tramaIdentificada;



/*------------ DEFINICIONES DE PROTOTIPOS DE FUNCIONES: --------------------*/
void  n2_GED_ETmain( UWORD pipeEscrituraGED, UWORD pipeLecturaGED,
  UBYTE identificadorN2);
UWORD IdentificaTramaGED_UI( UBYTE *frame, UWORD frameLength, UWORD *numRef,
  UBYTE *tipoMensaje, UBYTE *indicadorAccion);
void  FormarTramaGED_UI( UBYTE **trama, UWORD *longTrama, UWORD *numRef,
  UBYTE tipoMensaje, UBYTE indicadorAccion);
```

```
void  FormarMensaje23( MENSAJE23 *message23, UWORD tipoMens, UWORD longDatos,
  UBYTE *datos);

void  EnviarGEDliberacionMemoReq( void);
UWORD TratarGEDasignacionIETind( void);

UWORD CheckNumRef( UWORD numRefRecibido);



/*------------ DECLARACION DE FUNCIONES: ----------------------------------*/

void  n2_GED_ETmain(pipeEscrituraGED, pipeLecturaGED, identificadorN2)
UWORD pipeEscrituraGED, pipeLecturaGED;
UWORD identificadorN2;
{
  /*-------------------------------------------------------------------------
    Esta función es la implementación de la entidad de gestión de capa de
    Enlace de Datos del lado del Equipo Terminal. A partir de esta función
    se creará el proceso correspondiente a esta entidad
  -------------------------------------------------------------------------*/
  /*-------------------------------------------------------------------------
    Parámetros de Entrada al Proceso que representa a la entidad del Gestor
    de Enlace de Datos del lado del Equipo Terminal:
  -------------------------------------------------------------------------*/
  pipeEscGED  = pipeEscrituraGED;
  pipeLecGED  = pipeLecturaGED;
  idProcesoN2 = identificadorN2;

  /*-------------------------------------------------------------------------
    Solicitamos al KERNEL el identificador de nuestro proceso de Gestor de
    Enlace de Datos que posteriormente indicamos al proceso-entidad de Capa
    de Enlace de Datos con el que nos comunicaremos:
  -------------------------------------------------------------------------*/
  idProcesoGED = KIDProceso();

  /*---------- Envío del primer mensaje al proceso de nivel 2 (lado ET): -----
    Se utiliza para indicar al proceso que representa la entidad de nivel 2
    del lado del Equipo Terminal el pipe de donde se leerán los mensajes
    procedentes de la misma (pipe de escritura para el nivel 2);
    también se indica el identificador de este proceso de Gestión de Enlace
    de Datos para la comunicación entre ambos procesos:
  -------------------------------------------------------------------------*/
  FormarMensaje23( &mensaje23, idProcesoGED, pipeLecGED, NULL);
  KEscribePipe( pipeEscGED, (void*)&mensaje23, LONG_MENSAJE23, KWAIT);

  /*-------------------------------------------------------------------------
    Se solicita el temporizador necesario para el proceso de la comunicación
     del proceso de Gestor de Enlace de Datos del lado del Equipo Terminal:
  -------------------------------------------------------------------------*/
  KSolicitaContador( &timerGED_ET);

  while(TRUE)   /*El proceso vive para siempre*/
  {
    KWait();
    KLeeSignal(&signal);
    switch(signal)
    {
      case GED_ASIGNACION_IET_IND:
        TratarGEDasignacionIETind();
        break;
      case GED_LIBERACION_MEMO_IND:
        KLeePipe( pipeLecGED, &mensaje23, LONG_MENSAJE23, KWAIT);
        KLiberaMemoria( (UWORD *)mensaje23.datos);
        break;
      case GED_UNIDAD_DATOS_IND:
        KLeePipe( pipeLecGED, &mensaje23, LONG_MENSAJE23, KWAIT);
        tramaIdentificada = IdentificaTramaGED_UI( mensaje23.datos,
          mensaje23.longDatos, &numRef, &tipoMensaje, &indicadorAccion);
```

```
        EnviarGEDliberacionMemoReq();
        if (tramaIdentificada == TRUE)
          switch(tipoMensaje)
          {
            case PETICION_PRUEBA_IDENTIDAD:
              if ((indicadorAccion == IET_DE_GRUPO) ||
                (indicadorAccion == numIETasignado))
              {
                FormarTramaGED_UI( (UBYTE **)trama, &longTrama, &numRef,
                  RESPUESTA_PRUEBA_IDENTIDAD, numIETasignado); /*con Ri = 0*/
                FormarMensaje23( &mensaje23, GED_UNIDAD_DATOS_REQ, longTrama,
                  trama);
                KEscribePipe( pipeEscGED, (void *)(&mensaje23),
                  LONG_MENSAJE23, KWAIT);
                KSignal( idProcesoN2, GED_UNIDAD_DATOS_REQ);
              }
              break;
            case SUPRESION_IDENTIDAD:
              if ((indicadorAccion == IET_DE_GRUPO) ||
                (indicadorAccion == numIETasignado))
              {
                KSignal( idProcesoN2, GED_SUPRESION_IET_REQ);
                numIETasignado = IET_DE_GRUPO;
              }
              break;
          } /*switch(tipoMensaje)*/
        break;
    } /*switch(signal)*/
  } /*while(TRUE)*/
} /*n2_GED_ETmain*/


UWORD TratarGEDasignacionIETind( void)
{
  /*-------------------------------------------------------------------------
    Se ejecutará esta función cada vez que llegue la señal
    GED_ASIGNACION_IET_IND. Se enviará a la entidad de gestión par un mensaje
    de petición de identidad
  -------------------------------------------------------------------------*/
  UBYTE n202;
  UWORD tramaIdentificada;
  for( n202=0, indiceVecNumRef=0; n202<N202; n202++)
  {
    FormarTramaGED_UI( (UBYTE **)trama, &longTrama, &numRef,
      PETICION_IDENTIDAD, IET_DE_GRUPO);
    /*-----------------------------------------------------------------------
       Se guarda el número de referencia de cada mensaje de PETICION_IDENTIDAD
       enviado para posterior comparación cuando llega un mensaje
       IDENTIDAD_ASIGNADA:
    -----------------------------------------------------------------------*/
    vecNumRef[indiceVecNumRef++] = numRef;
    FormarMensaje23( &mensaje23, GED_UNIDAD_DATOS_REQ, longTrama,
      trama);
    KEscribePipe( pipeEscGED, (void *)&mensaje23, LONG_MENSAJE23,
      KWAIT);
    KSignal( idProcesoN2, GED_UNIDAD_DATOS_REQ);
    KActivaContador( timerGED_ET, T202);
    while( KEsperaContador( timerGED_ET) != KNO_ERROR)
    {
      KLeeSignal(&signal);
      switch(signal)
      {
        case GED_UNIDAD_DATOS_IND:
          KLeePipe( pipeLecGED, &mensaje23, LONG_MENSAJE23, KWAIT);
          tramaIdentificada = IdentificaTramaGED_UI( mensaje23.datos,
            mensaje23.longDatos, &numRef, &tipoMensaje, &indicadorAccion);
          EnviarGEDliberacionMemoReq();
          if (tramaIdentificada == TRUE)
          {
            switch(tipoMensaje)
```

```
            case IDENTIDAD_ASIGNADA:
              /* Comparamos número de referencia recibido con los numRef
                 almacenados en el vector "vecNumRef": */
              if (CheckNumRef( numRef) == TRUE)
              {
                numIETasignado = indicadorAccion;
                FormarMensaje23( &mensaje23, numIETasignado,
                  LONG_CERO, NULL);
                KEscribePipe( pipeEscGED, (void *)&mensaje23,
                  LONG_MENSAJE23, KWAIT);
                KSignal( idProcesoN2, GED_ASIGNACION_IET_REQ);
                return(TRUE);
              }
              else
                break;
            default:
              break;
          } /*switch(tipoMensaje)*/
        break;
      case GED_LIBERACION_MEMO_IND:
        KLeePipe( pipeLecGED, &mensaje23, LONG_MENSAJE23, KWAIT);
        KLiberaMemoria( (UWORD *)mensaje23.datos);
        break;
    } /*switch(signal)*/
  } /*while( KEsperaContador() != KNO_ERROR)*/
 } /*for( n202=0; n202<N202; n202++)*/
 KSignal( idProcesoN2, GED_NO_DISPONIBLE_IET_REQ);
 return(FALSE);
} /*TratarGEDasignacionIETind*/


UWORD IdentificaTramaGED_UI( frame, frameLength, numRef, tipoMensaje,
  indicadorAccion)
UBYTE *frame, *tipoMensaje, *indicadorAccion;
UWORD frameLength, *numRef;
{
  if (frame[2] != IDENTIFICADOR_GED)
    return(FALSE);
  *numRef = GET_NUM_REF( frame[3], frame[4]);
  *indicadorAccion = GET_INDICADOR_ACCION(frame[6]);;
  switch(frame[5])
  {
    case PETICION_IDENTIDAD:
      if (*indicadorAccion != IET_DE_GRUPO)
        return(FALSE);
      *tipoMensaje = PETICION_IDENTIDAD;
      break;
    case IDENTIDAD_ASIGNADA:
      *tipoMensaje = IDENTIDAD_ASIGNADA;
      break;
    case IDENTIDAD_RECHAZADA:
      *tipoMensaje = IDENTIDAD_RECHAZADA;
      break;
    case PETICION_PRUEBA_IDENTIDAD:
      if ((frame[3] != OCTETO_NULO) || (frame[4] != OCTETO_NULO))
        return(FALSE);
      *tipoMensaje = PETICION_PRUEBA_IDENTIDAD;
      break;
    case RESPUESTA_PRUEBA_IDENTIDAD:
      *tipoMensaje = RESPUESTA_PRUEBA_IDENTIDAD;
      break;
    case SUPRESION_IDENTIDAD:
      if ((frame[3] != OCTETO_NULO) || (frame[4] != OCTETO_NULO))
        return(FALSE);
      *tipoMensaje = SUPRESION_IDENTIDAD;
      break;
    case VERIFICACION_IDENTIDAD:
      if ((frame[3] != OCTETO_NULO) || (frame[4] != OCTETO_NULO))
        return(FALSE);
      *tipoMensaje = VERIFICACION_IDENTIDAD;
```

```
      break;
    default:
      return(FALSE);
  }
  if ((frameLength > (ADR_LENGTH + LONG_GED_UI_INFO)) &&
    (*tipoMensaje != RESPUESTA_PRUEBA_IDENTIDAD))
    return(FALSE);
  return(TRUE);
} /*IdentificaTramaGED_UI*/


void  FormarTramaGED_UI( trama, longTrama, numRef, tipoMensaje,
  indicadorAccion)
UBYTE **trama, indicadorAccion, tipoMensaje;
UWORD *longTrama, *numRef;
{
  *longTrama = LONG_GED_UI_INFO;
  KPideMemoria( *longTrama, (UWORD**)(&(*trama)));
  (*trama)[0] = IDENTIFICADOR_GED;
  switch(tipoMensaje)
  {
    case PETICION_PRUEBA_IDENTIDAD:
    case SUPRESION_IDENTIDAD:
    case VERIFICACION_IDENTIDAD:
      /*-------------------------------------------------------------------------
        En estos casos los dos octetos correspondientes al número de
        referencia van codificados a cero:
      -------------------------------------------------------------------------*/
      (*trama)[1] = OCTETO_NULO;
      (*trama)[2] = OCTETO_NULO;
      break;
    case IDENTIDAD_ASIGNADA:
    case IDENTIDAD_RECHAZADA:
      /*-------------------------------------------------------------------------
         En estos casos, se responde a la entidad de gestión remota con el
         número de referencia enviado previamente en el mensaje al que estamos
         respondiendo:
      -------------------------------------------------------------------------*/
      (*trama)[1] = (UBYTE)(MAKE_MSB_NUM_REF( *numRef));
      (*trama)[2] = (UBYTE)(MAKE_LSB_NUM_REF( *numRef));
      break;
    case PETICION_IDENTIDAD:
    case RESPUESTA_PRUEBA_IDENTIDAD:
      /*-------------------------------------------------------------------------
        En el caso de querer enviar un mensaje de PETICION DE IDENTIDAD o de
          RESPUESTA_PRUEBA_IDENTIDAD, estos dos octetos contendrán un valor
          aleatorio de 16bits (de 0 a 65535)
        Este valor llamado número de referencia, se utiliza para establecer
          una diferencia entre una serie de equipos de usuario que pueden
          solicitar simultámenamente la asignación de un valor IET.
      -------------------------------------------------------------------------*/
      (*trama)[1] = (UBYTE)(MAKE_MSB_NUM_REF( *numRef));
      (*trama)[2] = (UBYTE)(MAKE_LSB_NUM_REF( *numRef));
      break;
  } /*switch(tipoMensaje)*/
  (*trama)[3] = tipoMensaje;
  (*trama)[4] = MAKE_BYTE_INDICADOR_ACCION( indicadorAccion);
} /*FormarTramaGED_UI*/


void FormarMensaje23( message23, tipoMens, longDatos, datos)
MENSAJE23 *message23;
UWORD tipoMens, longDatos;
UBYTE *datos;
{
  message23->tipo      = tipoMens;
  message23->longDatos = longDatos;
  message23->datos     = datos;
}
```

```
void EnviarGEDliberacionMemoReq( void)
{
  FormarMensaje23( &mensaje23, GED_LIBERACION_MEMO_REQ, mensaje23.longDatos,
    mensaje23.datos);
  KEscribePipe( pipeEscGED, (void *)&mensaje23, LONG_MENSAJE23, KWAIT);
  KSignal( idProcesoN2, GED_LIBERACION_MEMO_REQ);
  KLiberaMemoria( (UWORD *)mensaje23.datos);
}


UWORD CheckNumRef( numRefRecibido)
UWORD numRefRecibido;
{
  /*-------------------------------------------------------------------------
     Compara el Número de Referencia Recibido con los almacenados en el
     vector "vecNumRef". Desde que encuentre uno igual, devuelve TRUE;
     sí no encuentra ninguno igual devuelve FALSE:
  -------------------------------------------------------------------------*/
  UWORD indiceAux;
  for (indiceAux=0; indiceAux<=indiceVecNumRef; indiceAux++)
    if (numRefRecibido == vecNumRef[indiceAux])
    {
      indiceVecNumRef = 0;
      return(TRUE);
    }
  return(FALSE);
}
```

```
/*****************************************************************************/
/*                              N2_TR.C                                      */
/*          TRABAJO FIN DE CARRERA  SANTIAGO MARTIN ROMANI:                  */
/*              IMPLEMENTACION PROTOCOLO LAP-D PARA EL                       */
/*            MICROCONTROLADOR DE COMUNICACIONES MC68302                     */
/*                                                                           */
/*      IMPLEMENTACION PROCESO NIVEL 2 PARA EL LADO DEL TERMINAL DE RED      */
/*****************************************************************************/

#define UNIX
/*#define $CODESEG S105*/
/*#define $INITSEG S205*/
/*#define $DATASEG S305*/


/*------------ FICHEROS INCLUIDOS: ----------------------------------------*/
#include "basico.h"    /* ..\kernel\basico.h */
#include "kernel.h"    /* ..\kernel\kernel.h */
#include "inicio.h"
#include "n2n3.h"
#include "ged.hp"
#include "nivel2.hp"
#include "nivel1.h"

#include "mux_tr.h"
#include "mux.hp"


/*------------ DECLARACION DE VARIABLES GLOBALES: -------------------------*/
UWORD signal;
UWORD estado; /* Estado ptos extremos de conexion de enlace datos pto a pto */

UWORD estadoEnTx;
/*---------------------------------------------------------------------------
  Estado en el proceso de transmisión (subestado del estado EnlaceEstablecido
  en el que se encuentra la conexión de Enlace de Datos en el proceso de tx/rx
---------------------------------------------------------------------------*/

UBYTE estadoNivel1; /* Estado capa o nivel físico */
UWORD timer;  /* temporizador: funcionará como contador hasta T200 ó T203 */

/*---------- Variable puntero trama; se utiliza de dos formas: ---------------
  1.- Como puntero al cabecero de una trama en el proceso de formación y
      envío de una trama
  2.- Como puntero a la zona de memoria que contiene una trama recién
      recibida por el nivel físico
-----------------------------------------------------------------------------*/
UBYTE *trama;

UWORD longTrama;  /* Longitud de la trama en bytes */
UWORD n200;      /* número de retransmisiones */
UWORD n200bis;   /* número de transmisiones (incluyendo retransmisiones) */
UWORD n201;      /* número de octetos en campo de información de trama*/


/*---------------------------------------------------------------------------
  Estructura para la comunicación con la entidad de nivel 3 y con la entidad
    de gestión de capa de Capa de Enlace de Datos
  La información intercambiada através de los pipes utilizará esta estructura
    es decir, en el pipe de escritura se escribirá un mensaje de información
    cuyos campos coincidirán con los de esta estructura
-----------------------------------------------------------------------------*/
MENSAJE23 mensaje23;

/*---------------------------------------------------------------------------
  Estructura de los mensajes intercambiados con el proceso multiplexor
-----------------------------------------------------------------------------*/
MENSAJE_MUX_N2 mensajeN2Mux;


UBYTE numIETasignado, numIPASasignado;
UBYTE numIETrecibido, numIPASrecibido;
```

```
/*-------------------------------------------------------------------------
  Indica si hay una trama de información (trama I) esperando en la ventana
   para ser enviada
-------------------------------------------------------------------------*/
UWORD ventanaEsperandoTx;

/*-------------------------------------------------------------------------
  Esta variable tendrá valor TRUE cuando hayamos enviado una trama de
  supervisión y estemos esperando recibir una trama respuesta a la enviada
-------------------------------------------------------------------------*/
UWORD esperandoRespuesta;

UWORD numTxTramaI;

/*----------- Identificadores utilizados en el proceso de comunicacion: ----*/
UBYTE idProcesoMuxN3;
/* Identificador proceso multiplexor de nivel 3 */
UBYTE idProcesoN2;
/* Identificador proceso nivel 2 para la comunicación con el multiplexor
   de nivel 2 */
UBYTE idProcesoMuxN2;
/* Identificador proceso multiplexor de nivel 2 */
UBYTE idProcesoGED;
/* Identificador proceso Entidad de Gestión de Capa
   de Capa de Enlace de Datos */

/*----------- Pipes de comunicación con el Nivel 3: ----------------------
  A través del pipe de escritura se envían los mensajes al nivel 3;
  a través del pipe de lectura se reciben los mensajes procedentes del
  nivel 3:
-------------------------------------------------------------------------*/
UWORD pipeLecN2N3;
UWORD pipeEscN2N3;

/*----------- Pipes de comunicación con la Entidad de Gestión de Capa: -------
  Análogamente a los pipes de comunicación entre las entidades de nivel 2 y
  nivel 3, a través del pipe de escritura se envían los mensajes del nivel 2
  a la Entidad de Gestión de Capa de Enlace de Datos; y a través del pipe
  de lectura se reciben los mensajes procedentes de la Entidad de Gestión de
  Capa:
-------------------------------------------------------------------------*/
UWORD pipeLecN2GED;
UWORD pipeEscN2GED;

/*----------- Pipes de comunicación con el proceso multiplexor: ---------------
  A través del pipe de escritura se envían los mensajes al proceso multiplexor
  a través del pipe de lectura se reciben los mensajes procedentes del
  proceso multiplexor:
  NOTA: La estructura de información o mensaje intercambiada con el proceso
        multiplexor através de estos pipes es diferente a la intercambiada
        con la entidad-proceso de gestión y con la entidad-proceso de nivel 3
-------------------------------------------------------------------------*/
UWORD pipeLecN2Mux;
UWORD pipeEscN2Mux;

/*------- Conceptos relacionados con los distintos buffers utilizados: -------
  Buffer de Tramas Transmitidas: bufferTx
  Buffer de información confirmada pendiente de envío: bufferTramasI
  Buffer de información sin confirmación pendiente de envío: bufferTramasUI
  ---------------------------------------------------------------------------
  Se utiliza estructuras tipo COLA o FIFO (First In, First Out) para
  implementar dichos buffers. Utilizamos dos índices característicos:
  frente y final.
  FRENTE: Indica la posición del elemento más antiguo de la cola.
          Los elementos de la cola se extraen por el frente.
  FINAL:  Indica la posición en la que se insertará el siguiente elemento que
          se agrege a la cola.
          Los elementos se insertan a la cola por el final de la misma.
  ---------------------------------------------------------------------------
  Al enviar cualquier tipo de trama se guarda en el buffer de tx. (bufferTx)
  Al recibir un signal MUX_TR_DATOS_ENVIADOS_IND del nivel físico,
```

```
  frenteBtramasI    = finalBtramasI    = 0;
  frenteBtramasCtrl = finalBtramasCtrl = 0;

  /*-------------------------------------------------------------------------
    Inicialización variable "número IPAS asignado" de la conexión de Enlace
     de Datos a cero (0)
  -------------------------------------------------------------------------*/
  numIPASasignado = IPAS_CERO;

  /*-------------------------------------------------------------------------
    Inicialización variable esperando trama de supervisión respuesta
    Se activa (tomará el valor TRUE), cuando hayamos enviado una trama de
    supervisión y pasamos a esperar una trama de supervisión respuesta
  -------------------------------------------------------------------------*/
  esperandoRespuesta = FALSE;
}


UWORD GuardaBtx( tipo, header, info)
UWORD tipo;
UBYTE *header, *info;
{
  /*-------------------------------------------------------------------------
    Inserta un elemento (tipo trama, cabecero e información, en su caso)
    en la cola de tramas de transmisión (bufferTx) y devuelve TRUE.
    Si la cola está llena, devuelve FALSE

      ACLARACION IMPORTANTE: El bufferTx junto con las funciones RecuperaBtx
    y LiberaBtx son necesarias para tener un control secuencial de las tramas
    enviadas al nivel físico que esperan recibir un signal
    MUX_TR_DATOS_ENVIADOS_IND para liberar la memoria dinámica utilizada para
    formar el cabecero.
  -------------------------------------------------------------------------*/
  /* Comprobamos si cola llena: */
  if ((frenteBtx == finalBtx+1) ||
    ((frenteBtx == 0) && (finalBtx == LONG_BUF_TX-1)))
    return(FALSE);
  else
  {
    bufferTx[finalBtx].tipo   = tipo;
    bufferTx[finalBtx].header = header;
    bufferTx[finalBtx].info   = info;
    if ((++finalBtx) == LONG_BUF_TX)
      finalBtx = 0;
    return(TRUE);
  }
} /*GuardaBtx*/


UWORD RecuperaBtx(void)
{
  /*-------------------------------------------------------------------------
    Por cada signal MUX_TR_DATOS_ENVIADOS_IND, se ejecutará esta función:
    1.- Se comprueba tipo de la trama que corresponde: Si es distinto de
        TIPO_I, se libera la memoria dinámica asociada al cabecero de la trama
    2.- Si la trama es TIPO_UI, se indicará al nivel 3 que libere la memoria
        dinámica asociada a la información recibida con el signal
        ED_UNIDAD_DATOS_REQ correspondiente.
    3.- Si la trama es TIPO_GED_UI, se indicará a la entidad de gestión de
        enlace de datos del lado del Equipo Terminal que libere la memoria
        asociada con la trama enviada (sin campo de dirección) con el signal
        GED_UNIDAD_DATOS_REQ correspondiente.
    4.- Para cualquier tipo de tramas (TIPO_UI, TIPO_I ó TIPO_NOT_I_NOR_UI)
        se incrementa el índice frenteBtx de forma apropiada.
    NOTA:          La función devuelve FALSE si el buffer está vacío.
    ADVERTENCIA:   Sólo existirá una trama TIPO_I en espera de confirmación y
                   se guarda en su propia estructura: estructura "ventana"
                   (RECORDAR: En el Acceso Básico la ventana es uno)
        La función RecuperaBtx es necesaria para mantener un control del
        orden en el que se libera la memoria correspondiente a las tramas
        entregadas al nivel físico y confirmadas por este con cada signal
```

```
        MUX_TR_DATOS_ENVIADOS_IND
  ------------------------------------------------------------------------*/
  /* Comprobación cola vacía: */
  if (frenteBtx == finalBtx)
    return(FALSE);
  else
  {
    if (bufferTx[frenteBtx].tipo != TIPO_I)
      KLiberaMemoria( (UWORD *)(bufferTx[frenteBtx].header));
    if (bufferTx[frenteBtx].tipo == TIPO_UI)  /*IPAS = 0 e IET = 127*/
    {
      FormarMensaje23( &mensaje23, ED_LIBERACION_MEMO_IND, LONG_CERO,
        bufferTx[frenteBtx].info);
      KEscribePipe( pipeEscN2N3, (void *)&mensaje23, LONG_MENSAJE23, KWAIT);
      KSignal( idProcesoMuxN3, ED_LIBERACION_MEMO_IND);
    }
    else
      if (bufferTx[frenteBtx].tipo == TIPO_GED_UI)
      {
        /* Significa bufferTx[frenteBtx].tipo == TIPO_GED_UI, es decir,
         IPAS = IPAS_GED = 63  e  IET = IET_DE_GRUPO = 127*/
        FormarMensaje23( &mensaje23, GED_LIBERACION_MEMO_IND, LONG_CERO,
          bufferTx[frenteBtx].info);
        KEscribePipe( pipeEscN2GED, (void *)&mensaje23, LONG_MENSAJE23, KWAIT);
        KSignal( idProcesoGED, GED_LIBERACION_MEMO_IND);
      }
      /* El resto de tramas sólo tiene cabecero y, por tanto, no hay que
         liberar un campo de información */
    if ((++frenteBtx) == LONG_BUF_TX)
      frenteBtx = 0;
    return(TRUE);
  } /*else if (frenteBtx == finalBtx)*/
} /*RecuperaBtx*/


void LiberaBtx(void)
{
  /*-------------------------------------------------------------------------
    Realiza los mismos pasos que la función RecuperaBtx, pero para cada una de
      las tramas (ver comentario secuencia de pasos en la propia función
      RecuperaBtx)
    Se ejecutará está función, es decir, se liberará el bufferTx o buffer de
      transmisión sólo cuando el nivel físico se desactive
  -------------------------------------------------------------------------*/
  while (frenteBtx != finalBtx)
  {
    if (bufferTx[frenteBtx].tipo != TIPO_I)
      KLiberaMemoria( (UWORD *)(bufferTx[frenteBtx].header));

    if (bufferTx[frenteBtx].tipo == TIPO_UI)  /*IPAS = 0 e IET = 127*/
    {
      FormarMensaje23( &mensaje23, ED_LIBERACION_MEMO_IND, LONG_CERO,
        bufferTx[frenteBtx].info);
      KEscribePipe( pipeEscN2N3, (void *)&mensaje23, LONG_MENSAJE23, KWAIT);
      KSignal( idProcesoMuxN3, ED_LIBERACION_MEMO_IND);
    }
    else
      if (bufferTx[frenteBtx].tipo == TIPO_GED_UI)
      {
        /* Significa bufferTx[frenteBtx].tipo == TIPO_GED_UI, es decir,
           IPAS = IPAS_GED = 63  e  IET = IET_DE_GRUPO = 127*/
        FormarMensaje23( &mensaje23, GED_LIBERACION_MEMO_IND, LONG_CERO,
          bufferTx[frenteBtx].info);
        KEscribePipe( pipeEscN2GED, (void *)&mensaje23, LONG_MENSAJE23,
          KWAIT);
        KSignal( idProcesoGED, GED_LIBERACION_MEMO_IND);
      }
      /* El resto de tramas sólo tiene cabecero y, por tanto, no hay que
         liberar un campo de información */
    if ((++frenteBtx) == LONG_BUF_TX)
      frenteBtx = 0;
  } /*while (frenteBtx != finalBtx)*/
```

```
} /*LiberaBtx*/


UWORD GuardaBtramasI( info, longInfo)
UBYTE *info;
UWORD longInfo;
{
  /*--- Por cada signal ED_DATOS_REQ recibido, se ejecutará esta función: ---
    1.- Se añade al bufferTramasI la información del mensaje proveniente
        del nivel 3.
    2.- Se incrementa apropiadamente el índice finalBtramasI

    NOTA 1: La función devuelve FALSE si el buffer está lleno.
    NOTA 2: Cuando la ventana esté libre y el remoto esté dispuesto a recibir,
            formaremos una trama I (añadiendóle el correspondiente cabecero) y
            la enviaremos al nivel físico.
  ------------------------------------------------------------------------*/
  /* Comprobamos si cola llena:*/
  if ((frenteBtramasI == finalBtramasI+1) ||
    ((frenteBtramasI == 0) && (finalBtramasI == LONG_BUF_TRAMAS_I-1)))
    return(FALSE);
  else
  {
    bufferTramasI[finalBtramasI].info       = info;
    bufferTramasI[finalBtramasI].longInfo   = longInfo;
    if ((++finalBtramasI) == LONG_BUF_TRAMAS_I)
      finalBtramasI = 0;
    return(TRUE);
  }
}


void LiberaBtramasI(void)
{
  /*------------------------------------------------------------------------
     Libera la información enviada en los mensajes asociados con los signals
       ED_DATOS_REQ pendiente de envío
     Será necesario liberar el buffer de tramas I cuando se desactive el nivel
       físico o cuando se libere el enlace de datos
  ------------------------------------------------------------------------*/
  while(frenteBtramasI != finalBtramasI)
  {
    FormarMensaje23( &mensaje23, ED_LIBERACION_MEMO_IND, LONG_CERO,
      bufferTramasI[frenteBtramasI].info);
    KEscribePipe( pipeEscN2N3, (void *)&mensaje23, LONG_MENSAJE23, KWAIT);
    KSignal( idProcesoMuxN3, ED_LIBERACION_MEMO_IND);
    if (++frenteBtramasI == LONG_BUF_TX)
      frenteBtramasI = 0;
  }
}


UWORD GuardaBtramasUI( info, longInfo, numIPAS)
UBYTE *info, numIPAS;
UWORD longInfo;
{
  /*------------------------------------------------------------------------
    Por cada signal ED_UNIDAD_DATOS_REQ o GED_UNIDAD_DATOS_REQ recibidos,
    se ejecutará esta función:
      1.- Se añade al bufferTramasUI la información del correspondiente
          mensaje: información, su longitud y el IPAS que me identifica
          de dónde proviene la trama UI:
            Si numIPAS = IPAS_CERO = 0, proviene del nivel 3
            Si numIPAS = IPAS_GED = 63, proviene del Gestor de Enlace de Datos
      2.- Se incrementa apropiadamente el índice finalBtramasUI
    NOTA: La función devuelve FALSE si el buffer está lleno.
  ------------------------------------------------------------------------*/
  /* Comprobamos si cola llena:*/
  if ((frenteBtramasUI == finalBtramasUI+1) ||
    ((frenteBtramasUI == 0) && (finalBtramasUI == LONG_BUF_TRAMAS_I-1)))
    return(FALSE);
  else
```

```
  {
    bufferTramasUI[finalBtramasUI].info       = info;
    bufferTramasUI[finalBtramasUI].longInfo   = longInfo;
    bufferTramasUI[finalBtramasUI].numIPAS    = numIPAS;
    if ((++finalBtramasUI) == LONG_BUF_TRAMAS_I)
      finalBtramasUI = 0;
    return(TRUE);
  }
}


void LiberaBtramasUI(void)
{
  /*-------------------------------------------------------------------------
     Libera la información enviada en los mensajes asociados con los signals
       ED_UNIDAD_DATOS_REQ o GED_UNIDAD_DATOS_REQ pendientes de envío
     Será necesario liberar el buffer de tramas UI sólo si se desactiva el
       nivel físico
  -------------------------------------------------------------------------*/
  while(frenteBtramasUI != finalBtramasUI)
  {
    if (bufferTramasUI[finalBtramasUI].numIPAS == IPAS_CERO)
    {
      FormarMensaje23( &mensaje23, ED_LIBERACION_MEMO_IND, LONG_CERO,
        bufferTramasUI[frenteBtramasUI].info);
      KEscribePipe( pipeEscN2N3, (void *)&mensaje23, LONG_MENSAJE23, KWAIT);
      KSignal( idProcesoMuxN3, ED_LIBERACION_MEMO_IND);
    }
    else /*bufferTramasUI[finalBtramasUI].numIPAS == IPAS_GED*/
    {
      FormarMensaje23( &mensaje23, GED_LIBERACION_MEMO_IND, LONG_CERO,
        bufferTramasUI[frenteBtramasUI].info);
      KEscribePipe( pipeEscN2GED, (void *)&mensaje23, LONG_MENSAJE23, KWAIT);
      KSignal( idProcesoGED, GED_LIBERACION_MEMO_IND);
    }
    if (++frenteBtramasUI == LONG_BUF_TX)
      frenteBtramasUI = 0;
  } /*while(frenteBtramasUI != finalBtramasUI)*/
}


UWORD GuardaBtramasCtrl( header, lengHeader)
UBYTE *header;
UWORD lengHeader;
{
  /*-------------------------------------------------------------------------
    Será necesario guardar una trama de control que no pueda ser enviada en un
    momento determinado, para hacerlo posteriormente, cuando el nivel físico
    asi lo permita.
  -------------------------------------------------------------------------*/
  /* Comprobamos si cola llena:*/
  if ((frenteBtramasCtrl == finalBtramasCtrl+1) ||
    ((frenteBtramasCtrl == 0) &&
    (finalBtramasCtrl == LONG_BUF_TRAMAS_CTRL-1)))
    return(FALSE);
  else
  {
    bufferTramasCtrl[finalBtramasCtrl].header       = header;
    bufferTramasCtrl[finalBtramasCtrl].lengHeader   = lengHeader;
    if ((++finalBtramasCtrl) == LONG_BUF_TRAMAS_CTRL)
      finalBtramasCtrl = 0;
    return(TRUE);
  }
}


void LiberaBtramasCtrl(void)
{
  /* Será necesario liberar el buffer de tramas de control sólo si se
     desactiva el nivel físico */
  while(frenteBtramasCtrl != finalBtramasCtrl)
  {
```

```
    KLiberaMemoria( (UWORD *)(bufferTramasCtrl[frenteBtramasCtrl].header));
    if (++frenteBtramasCtrl == LONG_BUF_TRAMAS_CTRL)
      frenteBtramasCtrl = 0;
  }
}


void LiberaVentana(void)
{
  /*-------------------------------------------------------------------------
    Libera la memoria dinámica correspondiente al cabecero y a la información
    de la trama I enviada en espera de confirmación que forma la ventana.
    NOTA:  Se ejecutará esta función cuando se confirma la trama I que forma
           la ventana, o cuando se reinicializa o se pierde el enlace de
           nivel 2 o la conexión física y existe una trama I por confirmar
           en la ventana.
  -------------------------------------------------------------------------*/
  KLiberaMemoria( (UWORD *)ventana.header);
  FormarMensaje23( &mensaje23, ED_LIBERACION_MEMO_IND, LONG_CERO,
    ventana.info);
  KEscribePipe( pipeEscN2N3, (void *)&mensaje23, LONG_MENSAJE23, KWAIT);
  KSignal( idProcesoMuxN3, ED_LIBERACION_MEMO_IND);
}


void FormarMensaje23( mensaje23, tipoMens, longDatos, datos)
MENSAJE23 *mensaje23;
UWORD tipoMens, longDatos;
UBYTE *datos;
{
  /*-------------------------------------------------------------------------
    Forma un mensaje, es decir, actualiza la estructura mensaje con los campos
    apropiados que se quieren transmitir. (Posteriormente el mensaje será
    enviado a la entidad de nivel 3 o al gestor de enlace de datos
  -------------------------------------------------------------------------*/
  mensaje23->tipo      = tipoMens;
  mensaje23->longDatos = longDatos;
  mensaje23->datos     = datos;
}


void  FormarMensajeMuxN2( messageMuxN2, tipoMessage, idProcesoTR,
  header, lengHeader, info, longInfo)
MENSAJE_MUX_N2 *messageMuxN2;
UWORD tipoMessage;
UBYTE idProcesoTR;
UBYTE *header, *info;
UWORD lengHeader, longInfo;
{
  /*-------------------------------------------------------------------------
    Esta función sirve para formar un mensaje (mensajeN2Mux) que luego será
    enviado al proceso multiplexor de nivel 2
  -------------------------------------------------------------------------*/
  messageMuxN2->tipoMensaje = tipoMessage;
  messageMuxN2->idProcesoTR = idProcesoTR;
  messageMuxN2->header      = header;
  messageMuxN2->lengHeader  = lengHeader;
  messageMuxN2->info        = info;
  messageMuxN2->longInfo    = longInfo;
}


UWORD FormarCabeceroTrama( header, headerLength, tipoTrama, bitIR, bitPF)
UBYTE **header, bitIR, bitPF;
UWORD *headerLength, tipoTrama;
{
  /*-------------------------------------------------------------------------
     Esta función forma el cabecero de la trama que se solicita con los
     parámetros adecuados a cada tipo de trama
  -------------------------------------------------------------------------*/
```

```
switch(tipoTrama)
{
  case SABME:
    *headerLength = SHORT_HEADER;
    if (KPideMemoria( *headerLength, (UWORD **)(header)) == KNO_ERROR)
    {
      (*header)[0] = MSB_DIR( numIPASasignado, ACTIVO);
      (*header)[1] = LSB_DIR( numIETasignado);
      (*header)[2] = CTRL_TRAMAS_U( SABME_MASK, ACTIVO);
      return(TRUE);
    }
    else return(FALSE);
  case DM:
    *headerLength = SHORT_HEADER;
    if (KPideMemoria( *headerLength, (UWORD **)(header)) == KNO_ERROR)
    {
      (*header)[0] = MSB_DIR( numIPASasignado, INACTIVO);
      (*header)[1] = LSB_DIR( numIETasignado);
      (*header)[2] = CTRL_TRAMAS_U( DM_MASK, ACTIVO);
      return(TRUE);
    }
    else return(FALSE);
  case DISC:
    *headerLength = SHORT_HEADER;
    if (KPideMemoria( *headerLength, (UWORD **)(header)) == KNO_ERROR)
    {
      (*header)[0] = MSB_DIR( numIPASasignado, ACTIVO);
      (*header)[1] = LSB_DIR( numIETasignado);
      (*header)[2] = CTRL_TRAMAS_U( DISC_MASK, ACTIVO);
      return(TRUE);
    }
    else return(FALSE);
  case UA:
    *headerLength = SHORT_HEADER;
    if (KPideMemoria( *headerLength, (UWORD **)(header)) == KNO_ERROR)
    {
      (*header)[0] = MSB_DIR( numIPASasignado, INACTIVO);
      (*header)[1] = LSB_DIR( numIETasignado);
      (*header)[2] = CTRL_TRAMAS_U( UA_MASK, ACTIVO);
      return(TRUE);
    }
    else return(FALSE);
  case RR:
    *headerLength = LONG_HEADER;
    if (KPideMemoria( *headerLength, (UWORD **)(header)) == KNO_ERROR)
    {
      (*header)[0] = MSB_DIR( numIPASasignado, bitIR);
      (*header)[1] = LSB_DIR( numIETasignado);
      (*header)[2] = MSB_CTRL_RR;
      (*header)[3] = LSB_CTRL_TRAMAS_SI( nr, bitPF);
      return(TRUE);
    }
    else return(FALSE);
  case RNR:
    *headerLength = LONG_HEADER;
    if (KPideMemoria( *headerLength, (UWORD **)(header)) == KNO_ERROR)
    {
      (*header)[0] = MSB_DIR( numIPASasignado, bitIR);
      (*header)[1] = LSB_DIR( numIETasignado);
      (*header)[2] = MSB_CTRL_RNR;
      (*header)[3] = LSB_CTRL_TRAMAS_SI( nr, bitPF);
      return(TRUE);
    }
    else return(FALSE);
  case REJ:
    *headerLength = LONG_HEADER;
    if (KPideMemoria( *headerLength, (UWORD **)(header)) == KNO_ERROR)
    {
      (*header)[0] = MSB_DIR( numIPASasignado, bitIR);
      (*header)[1] = LSB_DIR( numIETasignado);
```

```
        (*header)[2] = MSB_CTRL_REJ;
        (*header)[3] = LSB_CTRL_TRAMAS_SI( nr, bitPF);
        return(TRUE);
      }
      else return(FALSE);
    case I:
      *headerLength = LONG_HEADER;
      if (KPideMemoria( *headerLength, (UWORD **)(header)) == KNO_ERROR)
      {
        (*header)[0] = MSB_DIR( numIPASasignado, ACTIVO);
        (*header)[1] = LSB_DIR( numIETasignado);
        (*header)[2] = MSB_CTRL_TRAMAS_I( ns);
        (*header)[3] = LSB_CTRL_TRAMAS_SI( nr, ACTIVO);
        return(TRUE);
      }
      else return( FALSE);
    case UI:
      *headerLength = SHORT_HEADER;
      if (KPideMemoria( *headerLength, (UWORD **)(header)) == KNO_ERROR)
      {
        (*header)[0] = MSB_DIR( numIPASasignado, ACTIVO);
        (*header)[1] = LSB_DIR( numIETasignado);
        (*header)[2] = CTRL_TRAMAS_U( UI_MASK, INACTIVO);
        return(TRUE);
      }
      else return(FALSE);
    case TIPO_GED_UI:
      /*-------------------------------------------------------------------
         En el caso de una trama proveniente del GED (Gestor de Enlace de
         Datos del lado del ET (Equipo Terminal, será la entidad de nivel 2
         la que forme la cabecera con IPAS_GED (63) e IET_DE_GRUPO (127):
      -------------------------------------------------------------------*/
      *headerLength = (UWORD)ADR_LENGTH;
      if (KPideMemoria( *headerLength, (UWORD **)(header)) == KNO_ERROR)
      {
        (*header)[0] = MSB_DIR( IPAS_GED, ACTIVO);
        (*header)[1] = LSB_DIR( IET_DE_GRUPO);
      }
      else return(FALSE);
    default:
      return(FALSE);
  } /*switch(tipoTrama)*/
} /*FormarCabeceroTrama*/


UWORD IdentificaTrama( frame, frameLength, bitioIR, bitioPF,
  numIETrecibido, numIPASrecibido)
UBYTE *frame, *bitioIR, *bitioPF, *numIETrecibido, *numIPASrecibido;
UWORD frameLength;
{
  /*-------------------------------------------------------------------
    Esta función identifica el tipo de trama recibida a partir de la trama
    recibida por el nivel físico y de su longitud, teniendo encuenta los
    distintos campos que caracterizan a las tramas (campo de dirección y campo
    de control) y los distintos valores que pueden tomar
  -------------------------------------------------------------------*/
  if ((GET_BIT_ED(frame[0]) != 0) || (GET_BIT_ED(frame[1]) != 1))
    return(DESCONOCIDO);
  *numIPASrecibido = GET_IPAS(frame[0]);
  *bitioIR         = GET_BIT_IR(frame[0]);
  *numIETrecibido  = GET_IET(frame[1]);
  if (((*numIPASrecibido == numIPASasignado) &&
    (*numIETrecibido == numIETasignado))
    || ((ES_TRAMA_UI(frame[2])) && (*numIETrecibido == IET_DE_GRUPO)))
    /*-------------------------------------------------------------------
    Si los valores de IET y IPAS de la trama recibida no coinciden con los
    valores locales;o dicha trama no es tipo UI y el IET no es el de difusión,
    dicha trama se descarta:
    -------------------------------------------------------------------*/
    switch(frameLength)
```

```
{
  case SHORT_HEADER:  /*cabecero de 3 bytes*/
    /* Se trata de tramas tipo U (no numeradas) */
    *bitioPF = GET_BIT_PF_TRAMAS_U(frame[2]);
    switch(GET_CTRL_TRAMAS_U(frame[2]))
    {
      case SABME_MASK:
        if (*bitioIR == ACTIVO)
          return(SABME);
        else
          return(DESCONOCIDO);
      case DM_MASK:
        if (*bitioIR == INACTIVO)
          return(DM);
        else
          return(DESCONOCIDO);
      case DISC_MASK:
        if (*bitioIR == ACTIVO)
          return(DISC);
        else
          return(DESCONOCIDO);
      case UA_MASK:
        if (*bitioIR == INACTIVO)
          return(UA);
        else
          return(DESCONOCIDO);
      default:
        return(DESCONOCIDO);
    }
  case LONG_HEADER: /*cabecero de 4 bytes*/
    /*Se trata de tramas tipo S (Supervision)*/
    *bitioPF = GET_BIT_PF_TRAMAS_SI(frame[3]);
    nr = GET_NX(frame[3]);
    switch(frame[2])
    {
      case MSB_CTRL_RR:
        return(RR);
      case MSB_CTRL_RNR:
        return(RNR);
      case MSB_CTRL_REJ:
        return(REJ);
      default:
        return(DESCONOCIDO);
    }
  default:
    if (LSBIT(frame[2]) == 0)
    {
      /* Se trata de tramas I  (Información): */
      ns = GET_NX(frame[2]);
      nr = GET_NX(frame[3]);
      *bitioPF = GET_BIT_PF_TRAMAS_SI(frame[3]);
      return(I);
    }
    else
      if (GET_CTRL_TRAMAS_U(frame[2]) == UI_MASK)
      {
        /* Se trata de tramas UI  (Información no numerada): */
        *bitioPF = GET_BIT_PF_TRAMAS_U(frame[2]);
        return(UI);
      }
      else
      {
        if ((*numIPASrecibido == IPAS_GED) &&
          (*numIETrecibido == IET_DE_GRUPO))
          /* Se trata de tramas GED_UI: */
          return(TIPO_GED_UI);
        else
          /* En otro caso, la trama recibida es de un tipo desconocido: */
          return(DESCONOCIDO);
      }
```

```
    } /*switch(frameLength)*/
  else /* if (((*numIPASrecibido == numIPASasignado) && ...)*/
    if ((*numIPASrecibido == IPAS_GED) && (*numIETrecibido == IET_DE_GRUPO))
      return( TIPO_GED_UI);
    else
      return(DESCONOCIDO);
} /*IdentificaTrama*/


UWORD EnviarTramaDelBufferUI( void)
{
  /*-------------------------------------------------------------------------
    Se ejecuta cada vez que se quiere enviar la 1ª trama en espera del
    bufferTramasUI: Se guarda en el bufferTx (buffer de transmisión) y se
    incrementa frenteBtramasUI apropiadamente
  -------------------------------------------------------------------------*/
  UWORD tipoTrama;
  if (bufferTramasUI[frenteBtramasUI].numIPAS == IPAS_CERO)
    tipoTrama = TIPO_UI;
  else /*(bufferTramasUI[frenteBtramasUI].numIPAS == IPAS_GED)*/
    tipoTrama = TIPO_GED_UI;
  FormarCabeceroTrama( &trama, &longTrama, tipoTrama,
    ACTIVO, INACTIVO);
  /* Se enviará la trama al proceso multiplexor de capa 2: */
  FormarMensajeMuxN2( &mensajeN2Mux, MUX_TR_DATOS_REQ, idProcesoN2,
    trama, longTrama, bufferTramasUI[frenteBtramasUI].info,
    bufferTramasUI[frenteBtramasUI].longInfo);
  KEscribePipe( pipeEscN2Mux, (void *)&mensajeN2Mux, LONG_MENSAJE_MUX_N2,
    KWAIT);
  KSignal( idProcesoMuxN2, MUX_TR_DATOS_REQ);
  return(TRUE);
} /*EnviarTramaDelBufferUI()*/


UWORD EnviarTramaDelBufferCtrl( void)
{
  /*-------------------------------------------------------------------------
    Se ejecuta cada vez que se quiere enviar la 1ª trama en espera del
    bufferTramasCtrl:  (función análoga a "EnviarTramaDelBufferUI")
    Se guarda en el bufferTx y se incrementa frenteBtramasCtrl apropiadamente
  -------------------------------------------------------------------------*/
  /* Se enviará la trama al proceso multiplexor de capa 2: */
  FormarMensajeMuxN2( &mensajeN2Mux, MUX_TR_DATOS_REQ, idProcesoN2,
    bufferTramasCtrl[frenteBtramasCtrl].header,
    bufferTramasCtrl[frenteBtramasCtrl].lengHeader, NULL, LONG_CERO);
  KEscribePipe( pipeEscN2Mux, (void *)&mensajeN2Mux, LONG_MENSAJE_MUX_N2,
    KWAIT);
  KSignal( idProcesoMuxN2, MUX_TR_DATOS_REQ);
  GuardaBtx( TIPO_NOT_I_NOR_UI, bufferTramasCtrl[frenteBtramasCtrl].header,
    NULL);
  if (++frenteBtramasCtrl == LONG_BUF_TRAMAS_CTRL)
    frenteBtramasCtrl = 0;
  return(TRUE);
} /*EnviarTramaDelBufferCtrl()*/


UWORD EnviarTramaVentana( void)
{
  /*-------------------------------------------------------------------------
    Se ejecuta cuando se quiera enviar la trama I contenida en la ventana de
    transmisión. Se guardará en el bufferTx
  -------------------------------------------------------------------------*/
  /* Se enviará la trama al proceso multiplexor de capa 2: */
  FormarMensajeMuxN2( &mensajeN2Mux, MUX_TR_DATOS_REQ, idProcesoN2,
    ventana.header, ventana.lengHeader, ventana.info, ventana.longInfo);
  KEscribePipe( pipeEscN2Mux, (void *)&mensajeN2Mux, LONG_MENSAJE_MUX_N2,
    KWAIT);
  KSignal( idProcesoMuxN2, MUX_TR_DATOS_REQ);
  GuardaBtx( TIPO_I, ventana.header, ventana.info);
  return(TRUE);
```

```
} /*EnviarTramaVentana*/


void  TratarTramaUIrecibida( numIETrecibido, numIPASrecibido)
UBYTE numIETrecibido, numIPASrecibido;
{
  /*-------------------------------------------------------------------------
     Esta función se ejecuta cada vez que se recibe una trama UI
     (UI o TIPO_GED_UI):
       - Si se trata de una trama UI, se enviará al nivel3
       - Si se trata de una trama TIPO_GED_UI, se enviará a GED
  -------------------------------------------------------------------------*/
  if ((numIPASrecibido == IPAS_GED) && (numIETrecibido == IET_DE_GRUPO))
  {
    FormarMensaje23( &mensaje23, GED_UNIDAD_DATOS_IND, longTrama, trama);
    KEscribePipe( pipeEscN2GED, (void *)&mensaje23, LONG_MENSAJE23, KWAIT);
    KSignal( idProcesoGED, GED_UNIDAD_DATOS_IND);
  }
  else
    if (numIPASrecibido == IPAS_CERO)
    {
      FormarMensaje23( &mensaje23, ED_UNIDAD_DATOS_IND, longTrama, trama);
      KEscribePipe( pipeEscN2N3, (void *)&mensaje23, LONG_MENSAJE23, KWAIT);
      KSignal( idProcesoMuxN3, ED_UNIDAD_DATOS_IND);
    }
    else
      KLiberaMemoria( (UWORD*)trama);
} /*TratarTramaUIrecibida*/


UWORD TratarEDdatosReq( void)
{
  /*-------------------------------------------------------------------------
    Guarda el mensaje de nivel 3 recibido para su posterior envío formando
    una trama I. Se ejecuta cuando estamos en uno de los estados que no
    permiten enviar en ese preciso momento una trama I:
      NoWaitingAck_RNR
      WaitingAck_RR
      WaitingAck_RNR
  -------------------------------------------------------------------------*/
  if (KLeePipe( pipeLecN2N3, (VOID *)&mensaje23, LONG_MENSAJE23, KWAIT)
    == KNO_ERROR)
    /* La función KLeePipe siempre devolverá KNO_ERROR */
  {
    GuardaBtramasI( mensaje23.datos, mensaje23.longDatos);
    return(TRUE);
  }
  return(FALSE);
}


UWORD TratarEDunidadDatosReq( void)
{
  /*-------------------------------------------------------------------------
    Esta función se ejecuta cada vez que recibo un signal ED_UNIDAD_DATOS_REQ:
      - Si bufferTramasUI está vacío:
          - Se puede transmitirá la información recién llegada formando una
            trama UI, y se añadirá al bufferTx
      - Si el bufferTramasUI no está vacío:
          - Se añade la información recién llegada al propio bufferTramasUI
            con numIPAS = IPAS_CERO = 0.

    NOTA: Es necesario contemplar el caso que no se haya transmitido ninguna
          trama y llegue un ED_UNIDAD_DATOS_REQ, por eso si el bufferTramasUI
          está vacío se intenta transmitir la información recién llegada.
          Si no se contempla, no llegaría un MUX_TR_DATOS_ENVIADOS_IND
          que permitiese transmitir esa 1ª trama UI al comenzar el programa.
  -------------------------------------------------------------------------*/
  KLeePipe( pipeLecN2N3, (VOID *)&mensaje23, LONG_MENSAJE23, KWAIT);
  FormarCabeceroTrama( &trama, &longTrama, UI, ACTIVO, INACTIVO);
```

```
  if (frenteBtramasUI == finalBtramasUI)
  {
    /* Se enviará la trama al proceso multiplexor de capa 2: */
    FormarMensajeMuxN2( &mensajeN2Mux, MUX_TR_DATOS_REQ, idProcesoN2,
      trama, longTrama, mensaje23.datos, mensaje23.longDatos);
    KEscribePipe( pipeEscN2Mux, (void *)&mensajeN2Mux, LONG_MENSAJE_MUX_N2,
      KWAIT);
    KSignal( idProcesoMuxN2, MUX_TR_DATOS_REQ);
    GuardaBtx( TIPO_UI, trama, mensaje23.datos);
    return(TRUE);
  }
  GuardaBtramasUI( mensaje23.datos, mensaje23.longDatos, IPAS_CERO);
  return(TRUE);
} /*TratarEDunidadDatosReq*/


UWORD TratarGEDunidadDatosReq( void)
{
  /*-------------------------------------------------------------------------
    Esta función se ejecuta cada vez que recibo un signal GED_UNIDAD_DATOS_REQ:
      - Si bufferTramasUI está vacío:
          - Se transmitirá la información recién llegada formando una
            trama TIPO_GED_UI y se añadirá al bufferTx
      - Si el bufferTramasUI no está vacío:
          - Se añade la información recién llegada al propio bufferTramasUI
            teniendo en cuenta que el número de IPAS destino será el IPAS_GED,
            es decir, el destino es el Gestor de Enlace del Terminal de Red

    NOTA: Es necesario contemplar el caso que no se haya transmitido ninguna
          trama y llegue un GED_UNIDAD_DATOS_REQ, por eso si el bufferTramasUI
          está vacío se transmite la información recién llegada.
          Si no se contempla, no llegaría un MUX_TR_DATOS_ENVIADOS_IND que
          permitiese transmitir esa 1ª trama TIPO_GED_UI al comenzar el
          programa.
  -------------------------------------------------------------------------*/
  KLeePipe( pipeLecN2GED, (VOID *)&mensaje23, LONG_MENSAJE23, KWAIT);
  FormarCabeceroTrama( &trama, &longTrama, TIPO_GED_UI, ACTIVO, INACTIVO);
  if (frenteBtramasUI == finalBtramasUI)
  {
    /* Se enviará la trama al proceso multiplexor de capa 2: */
    FormarMensajeMuxN2( &mensajeN2Mux, MUX_TR_DATOS_REQ, idProcesoN2,
      trama, longTrama, mensaje23.datos, mensaje23.longDatos);
    KEscribePipe( pipeEscN2Mux, (void *)&mensajeN2Mux, LONG_MENSAJE_MUX_N2,
      KWAIT);
    KSignal( idProcesoMuxN2, MUX_TR_DATOS_REQ);
    GuardaBtx( TIPO_GED_UI, trama, mensaje23.datos);
    return(TRUE);
  }
  GuardaBtramasUI( mensaje23.datos, mensaje23.longDatos, IPAS_GED);
  return(TRUE);
} /*TratarGEDunidadDatosReq*/


UWORD TratarMuxTRdatosEnviadosInd( void)
{
  /*-------------------------------------------------------------------------
    Se ejecutará esta función por cada signal MUX_TR_DATOS_ENVIADOS_IND:
    - Se ejecuta la función RecuperaBtx y después:
    - Si bufferTramasCtrl no está vacío:
        - Enviar la 1ª trama de control en espera
    - Si bufferTramasCtrl está vacío:
        - Si (ventanaEsperandoTx = TRUE)
            - Enviar trama I de la ventana y (ventanaEsperandoTx = FALSE)
        - Si no (ventanaEsperandoTx = TRUE)
            - Enviar la 1ª trama del bufferTramasUI
  -------------------------------------------------------------------------*/
  RecuperaBtx();
  /*-------------------------------------------------------------------------
    Se ejecutará la función RecuperaBtx cada vez que se reciba un signal
    MUX_TR_DATOS_ENVIADOS_IND: Esta función es necesaria para mantener un
```

```
    control del orden en el que se libera la memoria correspondiente a las
    tramas entregadas al nivel físico y que son confirmadas por éste con cada
    signal MUX_TR_DATOS_ENVIADOS_IND:
  ------------------------------------------------------------------------*/
  if (frenteBtramasCtrl != finalBtramasCtrl)
    EnviarTramaDelBufferCtrl();
  else
    if (ventanaEsperandoTx == TRUE)
    {
      if( EnviarTramaVentana() == TRUE)
      {
        numTxTramaI++;
        //ventanaEsperandoTx = FALSE;
      }
    }
    else
      if (frenteBtramasUI != finalBtramasUI)
        EnviarTramaDelBufferUI();
  return(TRUE);
} /*TratarMuxTRdatosEnviadosInd*/


UWORD EnviarTramaCtrl( frameType, bitioIR, bitioPF)
UWORD frameType;
UBYTE bitioIR, bitioPF;
{
  /*-------------------------------------------------------------------------
    - Si el bufferTramasCtrl está vacío:
        - Si se puede transmitir, se guarda en el bufferTx
    - Si no se pudo transmitir o si el bufferTramasCtrl no está vacío, se
      guarda en el bufferTramasCtrl para su posterior transmisión
  -------------------------------------------------------------------------*/
  FormarCabeceroTrama( &trama, &longTrama, frameType, bitioIR, bitioPF);
  if (frenteBtramasCtrl == finalBtramasCtrl)
  {
    /* Se enviará la trama al proceso multiplexor de capa 2: */
    FormarMensajeMuxN2( &mensajeN2Mux, MUX_TR_DATOS_REQ, idProcesoN2,
      trama, longTrama, NULL, LONG_CERO);
    KEscribePipe( pipeEscN2Mux, (void *)&mensajeN2Mux, LONG_MENSAJE_MUX_N2,
      KWAIT);
    KSignal( idProcesoMuxN2, MUX_TR_DATOS_REQ);
    GuardaBtx( TIPO_NOT_I_NOR_UI, trama, NULL);
    return(TRUE);
  }
  GuardaBtramasCtrl( trama, longTrama);
  return(TRUE);
}


UWORD EnlaceLiberado( void)
{
  /*--- Definiciones variables locales a este "estado" (a esta funcion): ---
    Estas definiciones son locales a cada una de las siguientes funciones que
    representan a cada uno de los posibles estados de la conexión de enlace
    de datos:
       EnlaceLiberado
       EstablecimientoPendiente
       LiberaciónPendiente
       WaitingAck_RR          |
       WaitingAck_RNR         |  Estos últimos cuatro son subestados
       NoWaitingAck_RR        |  del estado o función EnlaceEstablecido
       NoWaitingAck_RNR       |
  ------------------------------------------------------------------------*/
  UWORD tipoTrama;
  /* Tipo de trama del protocolo LAP-D que se ha recibido o que se enviará */
  UBYTE bitIR, bitPF;
  /*---------- Bit de Instrucción/Respuesta o Comando/Respuesta: ------------
       Es el 2º bit de menor peso del 1er. octeto del campo de dirección
       de todas las tramas del protocolo LAP-D.
     Bit de Polling/Final:
```

```
     Es el bit de menor peso del 2º octeto del campo de control de las
     tramas I y S del protocolo LAP-D.
   Los valores de ambos bits (ACTIVO = 1 ó INACTIVO = 0) se utilizarán en el
     proceso de formación y envío, y en el proceso de identificación de las
     tramas intercambiadas através de la conexión de nivel de enlace.
-------------------------------------------------------------------------*/
UBYTE numIETrecibido, numIPASrecibido;
UWORD error;      /* Error devuelto por el nivel físico */

/*-------------------------------------------------------------------------
   En los casos que se haya evolucionado a este estado porque se estaba
   esperando una respuesta y no se recibió después de N200 retransmisión de
   la instrucción, la variable debe ser puesta a FALSE:
-------------------------------------------------------------------------*/
if (esperandoRespuesta == TRUE)
  esperandoRespuesta = FALSE;

KWait();
KLeeSignal(&signal);
switch(signal)
{
  case MUX_TR_DATOS_IND:
    KLeePipe( pipeLecN2Mux, (VOID *)(&mensajeN2Mux), LONG_MENSAJE_MUX_N2,
      KWAIT);
    tipoTrama = IdentificaTrama( mensajeN2Mux.header,
      mensajeN2Mux.lengHeader, &bitIR, &bitPF,
      &numIETrecibido, &numIPASrecibido);
    switch(tipoTrama)
    {
      case UI:
      case TIPO_GED_UI:
        TratarTramaUIrecibida( numIETrecibido, numIPASrecibido);
        return( ENLACE_LIBERADO);
      case SABME:
        KLiberaMemoria( (UWORD*)trama);
        if (bitPF == ACTIVO)
        {
          numIETasignado  = numIETrecibido;
          numIPASasignado = numIPASasignado; /*será cero siempre*/
          EnviarTramaCtrl( UA, INACTIVO, ACTIVO);
          KActivaContador( timer, T203);
          vs = vr = va = 0;
          KSignal( idProcesoMuxN3, ED_ESTABLEC_IND);
          return(ENLACE_ESTABLECIDO);
        } /*if (bitPF == ACTIVO)*/
        else return(ENLACE_LIBERADO);
      case DISC:
        KLiberaMemoria( (UWORD*)trama);
        if (bitPF == ACTIVO)
          EnviarTramaCtrl( DM, INACTIVO, ACTIVO);
        /* En cualquier caso, permanecemos en el estado actual: */
        return(ENLACE_LIBERADO);
      default:
        /* Si recibimos cualquier otro tipo de trama, liberamos la
           memoria correspondiente: */
        KLiberaMemoria( (UWORD *)trama);
        return(ENLACE_LIBERADO);
    } /*switch(tipoTrama)*/
  case MUX_TR_DATOS_ENVIADOS_IND:
    TratarMuxTRdatosEnviadosInd();
    /* Permanecemos en el estado actual: */
    return(ENLACE_LIBERADO);
  case ED_ESTABLEC_REQ:
    EnviarTramaCtrl( SABME, ACTIVO, ACTIVO);
    KActivaContador( timer, T200);
    vs = vr = va = 0;
    return(ESTABLECIMIENTO_PENDIENTE);
  case ED_UNIDAD_DATOS_REQ:
    TratarEDunidadDatosReq();
    /* En cualquier caso permanecemos en el estado actual: */
```

```
      return(ENLACE_LIBERADO);
    case GED_UNIDAD_DATOS_REQ:
      TratarGEDunidadDatosReq();
      /* Permanecemos en el estado actual: */
      return(ENLACE_LIBERADO);
    case ED_LIBERACION_MEMO_REQ:
      KLeePipe( pipeLecN2N3, (VOID *)&mensaje23, LONG_MENSAJE23, KWAIT);
      KLiberaMemoria( (UWORD *)mensaje23.datos);
      /* Permanecemos en el estado actual: */
      return(ENLACE_LIBERADO);
    case GED_LIBERACION_MEMO_REQ:
      KLeePipe( pipeLecN2GED, (VOID *)&mensaje23, LONG_MENSAJE23, KWAIT) ;
      KLiberaMemoria( (UWORD *)(mensaje23.datos));
      /* Permanecemos en el estado actual: */
      return(ENLACE_LIBERADO);
    case GED_SUPRESION_IET_REQ:
    case MUX_TR_ENLACE_LIBERADO_IND:
      /*En ambos casos la conexión de Enlace de Datos se libera*/
      LiberaBtramasUI();
      return(ENLACE_LIBERADO);
    default:
      return(ENLACE_LIBERADO);
  } /*switch(signal)*/
} /*EnlaceLiberado*/


UWORD EstablecimientoPendiente( void)
{
  /*--- Definiciones variables locales a este "estado" (a esta funcion): ---*/
  UWORD signal;
  UWORD tipoTrama;
  UWORD resp;
  UBYTE bitIR, bitPF;
  UBYTE numIETrecibido, numIPASrecibido;
  UWORD error;
  UWORD n200 = 0; /* Inicializo número de retransmisiones de la trama SABME */
  while( n200 < N200)  /*Hasta N200 = 3 retransmisiones de la trama SABME*/
  {
    resp = KEsperaContador(timer);
    if (resp == KNO_ERROR)  /*terminó el temporizador*/
    {
      /*-------------------------------------------------------------------
         Permanecemos en el estado actual (EstablecimientoPendiente)
         hasta completar las N200 (3) retransmisiones, o al llegar la trama
         adecuada, por evolución a otro estado
      -------------------------------------------------------------------*/
      EnviarTramaCtrl( SABME, ACTIVO, ACTIVO);
      KActivaContador( timer, T200);
      vs = vr = va = 0;
      n200++;
      continue;  /*Salta a evaluar denuevo la condición del while*/
    } /*if (resp == KNO_ERROR)*/
    if (resp == KINTERRUP)
    {
      KLeeSignal(&signal);
      switch(signal)
      {
        case MUX_TR_DATOS_IND:
          KLeePipe( pipeLecN2Mux, (VOID *)(&mensajeN2Mux),
            LONG_MENSAJE_MUX_N2, KWAIT);
          tipoTrama = IdentificaTrama( mensajeN2Mux.header,
            mensajeN2Mux.lengHeader, &bitIR, &bitPF,
            &numIETrecibido, &numIPASrecibido);
          switch(tipoTrama)
          {
            case UI:
            case TIPO_GED_UI:
              TratarTramaUIrecibida( numIETrecibido, numIPASrecibido);
              continue;
```

```
      case UA:
        KLiberaMemoria( (UWORD *)trama);
        if (bitPF == ACTIVO)
        {
          vs = vr = va = 0;
          KActivaContador( timer, T203);
          KSignal( idProcesoMuxN3, ED_ESTABLEC_CONF);
          return( ENLACE_ESTABLECIDO);
        }
        else continue;
      case SABME:
        KLiberaMemoria( (UWORD *)trama);
        if (bitPF == ACTIVO)
        {
          EnviarTramaCtrl( UA, ACTIVO, ACTIVO);
          KActivaContador( timer, T203);
          vs = vr = va = 0;
          /*-----------------------------------------------------------
            Respondemos enviando una trama UA, pero no pasaremos al
            estado EnlaceEstablecido hasta que no recibamos una
            trama UA
          -----------------------------------------------------------*/
        }  /*if (biPF=ACTIVO)*/
        continue;
      case DISC:
        KLiberaMemoria( (UWORD *)trama);
        if (bitPF == ACTIVO)
        {
          EnviarTramaCtrl( DM, INACTIVO, ACTIVO);
          vs = vr = va = 0;
          KSignal( idProcesoMuxN3, ED_LIBERACION_IND);
          KSignal( idProcesoMuxN2, MUX_TR_ENLACE_LIBERADO_REQ);
          return(ENLACE_LIBERADO);
        }  /*if (biPF=ACTIVO)*/
        else continue;
      case DM:
        KLiberaMemoria( (UWORD *)trama);
        if (bitPF == ACTIVO)
        {
          KSignal( idProcesoMuxN3, ED_LIBERACION_IND);
          KSignal( idProcesoMuxN2, MUX_TR_ENLACE_LIBERADO_REQ);
          return(ENLACE_LIBERADO);
        }
        else continue;
      default:
        /* Si recibimos cualquier otro tipo de trama, liberamos la
           memoria correspondiente: */
        KLiberaMemoria( (UWORD *)trama);
        continue;
    } /*switch( tipoTrama)*/
  case MUX_TR_DATOS_ENVIADOS_IND:
    TratarMuxTRdatosEnviadosInd();
    /* Permanecemos en el estado actual: */
    continue;
  case ED_UNIDAD_DATOS_REQ:
    TratarEDunidadDatosReq();
    /* En cualquier caso permanecemos en el estado actual: */
    continue;
  case GED_UNIDAD_DATOS_REQ:
    TratarGEDunidadDatosReq();
    /* Permanecemos en el estado actual: */
    continue;
  case ED_LIBERACION_MEMO_REQ:
    KLeePipe( pipeLecN2N3, (VOID *)&mensaje23, LONG_MENSAJE23, KWAIT);
    KLiberaMemoria( (UWORD *)mensaje23.datos);
    /* Permanecemos en el estado actual: */
    continue;
  case GED_LIBERACION_MEMO_REQ:
    KLeePipe( pipeLecN2GED, (VOID *)&mensaje23, LONG_MENSAJE23, KWAIT) ;
    KLiberaMemoria( (UWORD *)mensaje23.datos);
```

```
          /* Permanecemos en el estado actual: */
          continue;
        case GED_SUPRESION_IET_REQ:
        case MUX_TR_ENLACE_LIBERADO_IND:
          /*En ambos casos se libera la conexión de Enlace de Datos:*/
          KSignal( idProcesoMuxN3, ED_LIBERACION_IND);
          LiberaBtramasUI();
          return(ENLACE_LIBERADO);
        default:
          continue;
      }  /*switch(signal)*/
    }  /*if (resp == KINTERRUP)*/
  }  /*while(n200 < N200)*/
  /*-------------------------------------------------------------------------
     Si no se consiguió establecer el enlace después de las N200
     retransmisiones, se pasa al estado ENLACE_LIBERADO y se indica al nivel 3
     con el signal ED_LIBERACION_IND:
  -------------------------------------------------------------------------*/
  KSignal( idProcesoMuxN3, ED_LIBERACION_IND);
  KSignal( idProcesoMuxN2, MUX_TR_ENLACE_LIBERADO_REQ);
  return(ENLACE_LIBERADO);
} /*EstablecimientoPendiente*/


UWORD EnlaceEstablecido()
{
  /* Inicialización variables globales empleadas en los subestados internos al
     estado EnlaceEstablecido: */
  n200 = 0;
  numTxTramaI = 0;

  /*-------------------------------------------------------------------------
    Al establecerse el enlace, no existe ninguna trama I esperando acuse de
      recibo (equivale a "NoWaitingAck") y el remoto está inicialmente
      preparado para recibir tramas I (equivale a "RR").
    Por esto el estado inicial en el proceso de transmisión/recepción
      (estadoEnTx) durante la conexión de enlace establecida será
      NO_WAITING_ACK_RR y por tanto la 1ª función que se ejecutará será
      NoWaitingAck_RR :
  -------------------------------------------------------------------------*/
  estadoEnTx = NO_WAITING_ACK_RR;
  while(TRUE)
  {
    switch(estadoEnTx)
    {
      /*-----------------------------------------------------------------------
        Existe cuatro subestados dentro del estado "ENLACE_ESTABLECIDO"
        que vienen definidos según dos parámetros:
          1.- El estado de espera de confirmación de la trama I (ventana 1):
              (esperando acuse de recibo o no).
          2.- El estado de la entidad remota:
              Receptor Preparado o No Preparado para recibir más tramas I
      -----------------------------------------------------------------------*/
      case NO_WAITING_ACK_RR:
        estadoEnTx = NoWaitingAck_RR();
      case NO_WAITING_ACK_RNR:
        estadoEnTx = NoWaitingAck_RNR();
      case WAITING_ACK_RR:
        estadoEnTx = WaitingAck_RR();
      case WAITING_ACK_RNR:
        estadoEnTx = WaitingAck_RNR();
      default:
        /*---------------------------------------------------------------------
          Si el estado al que se evoluciona no es ninguno de los cuatro
            subestados, se pasa al discriminador de estados principal,
            es decir, se pasa al switch(estado) de la función Nivel2Main.
          Cuando estadoEnTx tome los siguientes valores:
            LIBERACION_PENDIENTE, ESTABLECMIENTO_PENDIENTE o ENLACE_LIBERADO,
            se pasará al switch (estado) de la función Nivel2Main.
```

```
        ------------------------------------------------------------------*/
        return(estadoEnTx);
    }
  }
} /*EnlaceEstablecido*/


UWORD NoWaitingAck_RR( void)
{
  /*--- Definiciones variables locales a este "estado" (a esta funcion): ---*/
  UWORD signal;
  UWORD tipoTrama;
  UBYTE bitIR, bitPF;
  UBYTE numIETrecibido, numIPASrecibido;
  UWORD resp;
  UWORD error;          /* Error devuelto por el nivel físico */
  resp = KEsperaContador( timer);
  if (resp == KNO_ERROR)
  {
    if (n200 < N200)
    {
      nr = vr;
      EnviarTramaCtrl( RR, INACTIVO, ACTIVO);
      KActivaContador( timer, T203);
      n200++;
      esperandoRespuesta = TRUE;
      return(NO_WAITING_ACK_RR);
    }
    else /*if (n200 < N200), es decir cuando (n200 == N200)*/
    {
      EnviarTramaCtrl( DM, INACTIVO, ACTIVO);
      KSignal( idProcesoMuxN3, ED_LIBERACION_IND);
      KSignal( idProcesoMuxN2, MUX_TR_ENLACE_LIBERADO_REQ);
      return(ENLACE_LIBERADO);
    }
  } /*if (resp == KNO_ERROR)*/
  else /*(resp == KINTERRUP)*/
  {
    /*Se ha recibido un signal, por tanto se procesa:*/
    KLeeSignal( &signal);
    switch( signal)
    {
      case MUX_TR_DATOS_IND:
        KLeePipe( pipeLecN2Mux, (VOID *)(&mensajeN2Mux), LONG_MENSAJE_MUX_N2,
          KWAIT);
        tipoTrama = IdentificaTrama( mensajeN2Mux.header,
          mensajeN2Mux.lengHeader, &bitIR, &bitPF,
          &numIETrecibido, &numIPASrecibido);
        switch(tipoTrama)
        {
          case I:
            if ((ns == vr) && (NR_VALIDO(nr, va, ns)))
            {
              if (nr == vs)
              {
                /* Si se cumple esta condición, está confirmando: */
                va = nr;
              }
              vr++;
              n200 = 0;
              FormarMensaje23( &mensaje23, ED_DATOS_IND, longTrama, trama);
              KEscribePipe( pipeEscN2N3, (void *)&mensaje23,
                LONG_MENSAJE23, KWAIT);
              KSignal( idProcesoMuxN3, ED_DATOS_IND);
              nr = vr;  /* Para formar el cabecero de la trama RR */
              /* Si bitPF = 0 ó 1, se enviará una respuesta RR (bitIR = 1)
                 y bitPF = 1: */
              EnviarTramaCtrl( RR, INACTIVO, ACTIVO);
              KActivaContador( timer, T203);
              return(NO_WAITING_ACK_RR);
```

```
    }
    else /*if ((ns == vr) && (NR_VALIDO(nr, va, ns)))*/
    {
      /* Se descarta las tramas I cuyo ns no sea igual a vr
         o cuando su nr no sea válido: */
      KLiberaMemoria( (UWORD *)trama);
      return(NO_WAITING_ACK_RR);
    }
  case RR:
    KLiberaMemoria( (UWORD *)trama);
    if (bitPF == ACTIVO)
    {
      if (esperandoRespuesta == TRUE)
        esperandoRespuesta = FALSE;
      nr = vr;  /* Para formar el cabecero de la trama RR */
      EnviarTramaCtrl( RR, INACTIVO, ACTIVO);
      KActivaContador( timer, T203);
      n200 = 0;
    }
    return(NO_WAITING_ACK_RR);
  case RNR:
    KLiberaMemoria( (UWORD *)trama);
    if ((bitPF == ACTIVO) && (NR_VALIDO(nr, va, ns)))
    {
      if (bitIR == ACTIVO)
      {
        /* Si la trama RNR recibida es una instrucción, se enviará
           una trama RR respuesta: */
        nr = vr;  /* Para formar el cabecero de la trama RR */
        EnviarTramaCtrl( RR, INACTIVO, ACTIVO);
        n200 = 0;
      }
      else /*(bitIR == INACTIVO); trama recibida es una respuesta*/
        esperandoRespuesta = FALSE;
      KActivaContador( timer, T203);
      return(NO_WAITING_ACK_RNR);
    }
    else
      return(NO_WAITING_ACK_RR);
  case UI:
  case TIPO_GED_UI:
    TratarTramaUIrecibida( numIETrecibido, numIPASrecibido);
    return(NO_WAITING_ACK_RR);
  case DISC:
    KLiberaMemoria( (UWORD*)trama);
    if (bitPF == ACTIVO)
    {
      EnviarTramaCtrl( UA, INACTIVO, ACTIVO);
      vs = vr = va = 0;
      KSignal( idProcesoMuxN3, ED_LIBERACION_IND);
      KSignal( idProcesoMuxN2, MUX_TR_ENLACE_LIBERADO_REQ);
      return(ENLACE_LIBERADO);
    }  /*if (biPF=ACTIVO)*/
    return(NO_WAITING_ACK_RR);
  case SABME:
    KLiberaMemoria( (UWORD *)trama);
    if (bitPF == ACTIVO)
    {
      EnviarTramaCtrl( UA, INACTIVO, ACTIVO);
      KActivaContador( timer, T203);
      vs = vr = va = 0;
      KSignal( idProcesoMuxN3, ED_ESTABLEC_IND);
      return( ENLACE_ESTABLECIDO);
    }  /*if (biPF=ACTIVO)*/
    else
      return(NO_WAITING_ACK_RR);
  default:
    /* Si recibimos cualquier otro tipo de trama, liberamos la
       memoria correspondiente: */
    KLiberaMemoria( (UWORD *)trama);
```

```
        return(NO_WAITING_ACK_RR);
    } /*switch(tipoTrama)*/
  case ED_DATOS_REQ:
    if (KLeePipe( pipeLecN2N3, (VOID *)&mensaje23, LONG_MENSAJE23, KWAIT)
      == KNO_ERROR)
      if (vs < (va + K))
      {
        /* Para formar el cabecero de la trama I: */
        ns = vs;
        nr = vr;
        FormarCabeceroTrama( &trama, &longTrama, I, ACTIVO, ACTIVO);
        /* Enviaremos un mensaje conteniendo la trama I al proceso
           multiplexor: */
        FormarMensajeMuxN2( &mensajeN2Mux, MUX_TR_DATOS_REQ,
          idProcesoN2, trama, longTrama, mensaje23.datos,
          mensaje23.longDatos);
        KEscribePipe( pipeEscN2Mux, (void *)&mensajeN2Mux,
          LONG_MENSAJE_MUX_N2, KWAIT);
        KSignal( idProcesoMuxN2, MUX_TR_DATOS_REQ);
        numTxTramaI = 1;
        KActivaContador( timer, T200);
        vs++;
        ventana.header     = trama;
        ventana.lengHeader = longTrama;
        ventana.info       = mensaje23.datos;
        ventana.longInfo   = mensaje23.longDatos;
        /* Paso al estado esperando acuse de recibo de la trama I
           enviada: */
        KActivaContador( timer, T203);
        n200 = 0;
        return(WAITING_ACK_RR);
      } /*if (vs <= va + k))*/
      else
    /* Por aquí no pasa nunca el programa: */
    return(NO_WAITING_ACK_RR);
  case ED_LIBERACION_REQ:
    EnviarTramaCtrl( DISC, ACTIVO, ACTIVO);
    KActivaContador( timer, T200);
    vs = vr = va = 0;
    return(LIBERACION_PENDIENTE);
  case MUX_TR_DATOS_ENVIADOS_IND:
    TratarMuxTRdatosEnviadosInd();
    /* Permanecemos en el estado actual: */
    return(NO_WAITING_ACK_RR);
  case ED_UNIDAD_DATOS_REQ:
    TratarEDunidadDatosReq();
    /* Permanecemos en el estado actual: */
    return(NO_WAITING_ACK_RR);
  case GED_UNIDAD_DATOS_REQ:
    TratarGEDunidadDatosReq();
    /* Permanecemos en el estado actual: */
    return(NO_WAITING_ACK_RR);
  case ED_LIBERACION_MEMO_REQ:
    KLeePipe( pipeLecN2N3, (VOID *)&mensaje23, LONG_MENSAJE23, KWAIT);
    KLiberaMemoria( (UWORD *)mensaje23.datos);
    /* Permanecemos en el estado actual: */
    return(NO_WAITING_ACK_RR);
  case GED_LIBERACION_MEMO_REQ:
    KLeePipe( pipeLecN2GED, (VOID *)&mensaje23, LONG_MENSAJE23, KWAIT) ;
    KLiberaMemoria( (UWORD *)mensaje23.datos);
    /* Permanecemos en el estado actual: */
    return(NO_WAITING_ACK_RR);
  case GED_SUPRESION_IET_REQ:
  case MUX_TR_ENLACE_LIBERADO_IND:
    KSignal( idProcesoMuxN3, ED_LIBERACION_IND);
    LiberaBtramasUI();
    LiberaBtramasI();
    return(ENLACE_LIBERADO);
  default:
    return(NO_WAITING_ACK_RR);
```

```
    } /*switch(signal)*/
  } /*else if (resp == KINTERRUP)*/
} /*NoWaitingAck_RR*/


UWORD NoWaitingAck_RNR( void)
{
  /*--- Definiciones variables locales a este "estado" (a esta funcion): ---*/
  UWORD signal;
  UWORD tipoTrama;
  UBYTE bitIR, bitPF;
  UBYTE numIETrecibido, numIPASrecibido;
  UWORD resp;
  UWORD error;
  resp = KEsperaContador( timer);
  if (resp == KNO_ERROR)
  {
    if (n200 < N200)
    {
      nr = vr;  /* Para formar el cabecero de la trama RR */
      EnviarTramaCtrl( RR, INACTIVO, ACTIVO);
      KActivaContador( timer, T203);
      n200++;
      esperandoRespuesta = TRUE;
      return(NO_WAITING_ACK_RR);
    }
    else /*if (n200 < N200), es decir cuando (n200 == N200)*/
    {
      LiberaBtramasI();
      EnviarTramaCtrl( DM, INACTIVO, ACTIVO);
      KSignal( idProcesoMuxN3, ED_LIBERACION_IND);
      KSignal( idProcesoMuxN2, MUX_TR_ENLACE_LIBERADO_REQ);
      return(ENLACE_LIBERADO);
    }
  } /*if (resp == KNO_ERROR)*/
  else /*(resp == KINTERRUP)*/
  {
    /* Se ha recibido un signal, por tanto se procesa: */
    KLeeSignal( &signal);
    switch( signal)
    {
      case MUX_TR_DATOS_IND:
        KLeePipe( pipeLecN2Mux, (VOID *)(&mensajeN2Mux), LONG_MENSAJE_MUX_N2,
          KWAIT);
        tipoTrama = IdentificaTrama( mensajeN2Mux.header,
          mensajeN2Mux.lengHeader, &bitIR, &bitPF,
          &numIETrecibido, &numIPASrecibido);
        switch(tipoTrama)
        {
          case I:
            if ((ns == vr) && (NR_VALIDO(nr, va, ns)))
            {
              if (nr == vs)
              {
                /* Si se cumple esta condición, la variable
                   nr es correcta: */
                va = nr;
              }
              vr++;
              FormarMensaje23( &mensaje23, ED_DATOS_IND, longTrama, trama);
              KEscribePipe( pipeEscN2N3, (void *)&mensaje23,
                LONG_MENSAJE23, KWAIT);
              KSignal( idProcesoMuxN3, ED_DATOS_IND);
              nr = vr;  /* Para formar el cabecero de la trama RR */
                /* Si bitPF = 0 ó 1, se enviará una respuesta RR (bitIR = 1)
                   y bitPF = 1: */
              EnviarTramaCtrl( RR, ACTIVO, INACTIVO);
              KActivaContador( timer, T203);
              n200 = 0;
```

```
    }
    else /*if ((ns == vr) && (NR_VALIDO(nr, va, ns)))*/
    {
      /* Se descarta las tramas I cuyo ns no sea igual a vr
         o cuando su nr no sea válido: */
      KLiberaMemoria( (UWORD *)trama);
    }
    /* En cualquier caso, permanecemos en el estado actual: */
    return(NO_WAITING_ACK_RNR);
  case RNR:
    KLiberaMemoria( (UWORD *)trama);
    if ((bitPF == ACTIVO) && (NR_VALIDO(nr, va, ns)))
    {
      if (nr == vs)
        /* Significa que la variable nr recibida es correcta: */
        va = nr;
      if (bitIR == ACTIVO)
      {
        /* Si la trama RNR recibida es una instrucción, se enviará
           una trama RR respuesta: */
        nr = vr; /* Para formar el cabecero de la trama RR */
        EnviarTramaCtrl( RR, INACTIVO, ACTIVO);
        n200 = 0;
      }
      else
        esperandoRespuesta = FALSE;
      KActivaContador( timer, T203);
      return(NO_WAITING_ACK_RNR);
    }
    else
      return(NO_WAITING_ACK_RNR);
  case RR:
    KLiberaMemoria( (UWORD *)trama);
    if (NR_VALIDO(nr, va, ns) && (nr == vs))
    {
      /*-----------------------------------------------------------------
        Significa que la variable nr es correcta, pués en este
          estado no se espera confirmación, sino que la variable
          nr de la trama recibida sea correcta:
            - Si el bufferTramasI no está vacío, envío una trama I
                y se añade al bufferTx y a la estructura ventana
                y pasamos al estado WaitingAck_RR
            - Si el buffer está vacío pasamos al estado
                NoWaitingAck_RR
        Si la variable nr (caso imposible según nuestro algoritmo)
          no es correcta, permanecemos en el estado actual
      -----------------------------------------------------------------*/
      va = nr;
      if ((frenteBtramasI != finalBtramasI) && (vs < (va + K) ))
      {
        /* Para formar el cabecero de la trama I: */
        nr = vr;
        ns = vs;
        FormarCabeceroTrama( &trama, &longTrama, I, ACTIVO, ACTIVO);
        /* Enviaremos un mensaje conteniendo la trama I al proceso
           multiplexor: */
        FormarMensajeMuxN2( &mensajeN2Mux, MUX_TR_DATOS_REQ,
          idProcesoN2, trama, longTrama, mensaje23.datos,
          mensaje23.longDatos);
        KEscribePipe( pipeEscN2Mux, (void *)&mensajeN2Mux,
          LONG_MENSAJE_MUX_N2, KWAIT);
        KSignal( idProcesoMuxN2, MUX_TR_DATOS_REQ);
        numTxTramaI = 1;
        GuardaBtx( TIPO_I, trama, bufferTramasI[frenteBtramasI].info);
        vs++;
        esperandoRespuesta = TRUE;
        ventana.header     = trama;
        ventana.lengHeader = longTrama;
        ventana.info       = bufferTramasI[frenteBtramasI].info;
        ventana.longInfo   = bufferTramasI[frenteBtramasI].longInfo;
```

```
            if (++frenteBtramasI == LONG_BUF_TRAMAS_I)
              frenteBtramasI = 0;
            ventanaEsperandoTx = TRUE;
            KActivaContador( timer, T200);
            n200 = 0;
            return(WAITING_ACK_RR);
          }/*if((frenteBtramasI != finalBtramasI) && (vs<=(va + K)))*/
          else
            return(NO_WAITING_ACK_RR);
        } /*if (NR_VALIDO(nr, va, ns) && (nr == vs))*/
          return(NO_WAITING_ACK_RNR);
      case UI:
      case TIPO_GED_UI:
        TratarTramaUIrecibida( numIETrecibido, numIPASrecibido);
        return(NO_WAITING_ACK_RNR);
      case DISC:
        KLiberaMemoria( (UWORD*)trama);
        if (bitPF == ACTIVO)
        {
          EnviarTramaCtrl( UA, INACTIVO, ACTIVO);
          vs = vr = va = 0;
          KSignal( idProcesoMuxN3, ED_LIBERACION_IND);
          KSignal( idProcesoMuxN2, MUX_TR_ENLACE_LIBERADO_REQ);
          return(ENLACE_LIBERADO);
        }  /*if (biPF=ACTIVO)*/
        return(NO_WAITING_ACK_RNR);
      case SABME:
        KLiberaMemoria( (UWORD *)trama);
        if (bitPF == ACTIVO)
        {
          EnviarTramaCtrl( UA, INACTIVO, ACTIVO);
          KActivaContador( timer, T203);
          vs = vr = va = 0;
          LiberaBtramasI();
          LiberaVentana();
          KSignal( idProcesoMuxN3, ED_ESTABLEC_IND);
          return( ENLACE_ESTABLECIDO);
        }  /*if (biPF=ACTIVO)*/
        else
          return(NO_WAITING_ACK_RNR);
      default:
        /* Si recibimos cualquier otro tipo de trama, liberamos la
           memoria correspondiente: */
        KLiberaMemoria( (UWORD *)trama);
        return(NO_WAITING_ACK_RNR);
    } /*switch(tipoTrama)*/
  case MUX_TR_DATOS_ENVIADOS_IND:
    TratarMuxTRdatosEnviadosInd();
    /* Permanecemos en el estado actual: */
    return(NO_WAITING_ACK_RNR);
  case ED_UNIDAD_DATOS_REQ:
    TratarEDunidadDatosReq();
    /* Permanecemos en el estado actual: */
    return(NO_WAITING_ACK_RNR);
  case GED_UNIDAD_DATOS_REQ:
    TratarGEDunidadDatosReq();
    /* Permanecemos en el estado actual: */
    return(NO_WAITING_ACK_RNR);
  case ED_DATOS_REQ:
    TratarEDdatosReq();
    /* Permanecemos en el estado actual: */
    return(NO_WAITING_ACK_RNR);
  case ED_LIBERACION_REQ:
    LiberaBtramasI();
    EnviarTramaCtrl( DISC, ACTIVO, ACTIVO);
    KActivaContador( timer, T200);
    vs = vr = va = 0;
    return(LIBERACION_PENDIENTE);
  case ED_LIBERACION_MEMO_REQ:
    KLeePipe( pipeLecN2N3, (VOID *)&mensaje23, LONG_MENSAJE23, KWAIT);
```

```
        KLiberaMemoria( (UWORD *)mensaje23.datos);
        /* Permanecemos en el estado actual: */
        return(NO_WAITING_ACK_RNR);
      case GED_LIBERACION_MEMO_REQ:
        KLeePipe( pipeLecN2GED, (VOID *)&mensaje23, LONG_MENSAJE23, KWAIT) ;
        KLiberaMemoria( (UWORD *)mensaje23.datos);
        /* Permanecemos en el estado actual: */
        return(NO_WAITING_ACK_RNR);
      case GED_SUPRESION_IET_REQ:
      case MUX_TR_ENLACE_LIBERADO_IND:
        KSignal( idProcesoMuxN3, ED_LIBERACION_IND);
        LiberaBtramasUI();
        LiberaBtramasI();
        return(ENLACE_LIBERADO);
      default:
        return(NO_WAITING_ACK_RNR);
    } /*switch(signal)*/
  } /*else if (resp == KINTERRUP)*/
} /*NoWaitingAck_RNR*/


UWORD WaitingAck_RNR( void)
{
  /*--- Definiciones variables locales a este "estado" (a esta funcion): ---*/
  UWORD signal;
  UWORD tipoTrama;
  UBYTE bitIR, bitPF;
  UBYTE numIETrecibido, numIPASrecibido;
  UWORD resp;
  UWORD error;        /* Error devuelto por el nivel físico */
  UWORD confirmando;
  /* La variable "confirmando" indica si está confirmando la última trama I
     enviada: */
  confirmando = FALSE;
  resp = KEsperaContador( timer);
  if (resp == KNO_ERROR)
  {
    if (n200 < N200)
    {
      nr = vr;  /* Para formar el cabecero de la trama RR */
      EnviarTramaCtrl( RR, INACTIVO, ACTIVO);
      n200++;
      KActivaContador( timer, T200);
      esperandoRespuesta = TRUE;
      return(WAITING_ACK_RNR);
    }
    else /*if (n200 < N200), es decir cuando (n200 == N200)*/
    {
      LiberaBtramasI();
      LiberaVentana();
      EnviarTramaCtrl( DM, INACTIVO, ACTIVO);
      KSignal( idProcesoMuxN3, ED_LIBERACION_IND);
      return(ENLACE_LIBERADO);
    }
  }
  else /*if (resp == KNO_ERROR)*/
  {
    /* Se ha recibido un signal, por tanto se procesa: */
    KLeeSignal( &signal);
    switch( signal)
    {
      case MUX_TR_DATOS_IND:
        KLeePipe( pipeLecN2Mux, (VOID *)(&mensajeN2Mux), LONG_MENSAJE_MUX_N2,
          KWAIT);
        tipoTrama = IdentificaTrama( mensajeN2Mux.header,
          mensajeN2Mux.lengHeader, &bitIR, &bitPF,
          &numIETrecibido, &numIPASrecibido);
        switch(tipoTrama)
        {
```

```
case I:
  if ((ns == vr) && (NR_VALIDO(nr, va, ns)))
  {
    if (nr == vs)
    {
      /* Si se cumple esta condición, está confirmando
         la última trama I enviada: */
      va = nr;
      confirmando = TRUE;
      LiberaVentana();
    }
    vr++;
    FormarMensaje23( &mensaje23, ED_DATOS_IND,
      longTrama, trama);
    KEscribePipe( pipeEscN2N3, (void *)&mensaje23,
      LONG_MENSAJE23, KWAIT);
    KSignal( idProcesoMuxN3, ED_DATOS_IND);
    nr = vr;  /* Para formar el cabecero de la trama RR */
    /* Si bitPF = 0 ó 1, se enviará una respuesta RR (bitIR = 1)
       y bitPF = 1: */
    EnviarTramaCtrl( RR, INACTIVO, ACTIVO);
    KActivaContador( timer, T203);
    /* Si la trama I recibida está confirmando la última trama
       I enviada, pasamos al estado NoWaitingAck_RNR: */
    if (confirmando == TRUE)
      return(NO_WAITING_ACK_RNR);
  }
  else /*if ((ns == vr) && (NR_VALIDO(nr, va, ns)))*/
  {
    /* Se descarta las tramas I cuyo ns no sea igual a vr
       o cuando su nr no sea válido: */
    KLiberaMemoria( (UWORD *)trama);
  }
  /* En otro caso, permanecemos en el estado actual: */
  return(WAITING_ACK_RNR);
case REJ:
  KLiberaMemoria( (UWORD *)trama);
  if (NR_VALIDO(nr, va, ns))
  {
    /* Si la trama REJ recibida es una intrucción, se enviará
       una trama RR respuesta: */
    if ((bitIR == ACTIVO) && (bitPF == ACTIVO))
      EnviarTramaCtrl( RR, INACTIVO, ACTIVO);
    /* En cualquier caso se reenviará la última trama I
       (trama I de la ventana): */
    /* Enviaremos un mensaje conteniendo la trama I al proceso
       multiplexor: */
    FormarMensajeMuxN2( &mensajeN2Mux, MUX_TR_DATOS_REQ,
      idProcesoN2, ventana.header, ventana.lengHeader,
      ventana.info, ventana.longInfo);
    KEscribePipe( pipeEscN2Mux, (void *)&mensajeN2Mux,
      LONG_MENSAJE_MUX_N2, KWAIT);
    KSignal( idProcesoMuxN2, MUX_TR_DATOS_REQ);
    numTxTramaI = 1;
    GuardaBtx( TIPO_I, ventana.header, ventana.info);
    ventanaEsperandoTx = TRUE;
    esperandoRespuesta = TRUE;
    n200 = 0;
    KActivaContador( timer, T200);
    return(WAITING_ACK_RR);
  }
  else /*if (NR_VALIDO(nr, va, ns))*/
    return(WAITING_ACK_RNR);
case RR:
  if (NR_VALIDO(nr, va, ns) && (nr == vs))
  {
    /*-----------------------------------------------------------------
      Significa que está confirmando última trama enviada:
        - Si el bufferTramasI no está vacío, envío una trama I
            y se añade al bufferTx y a la estructura ventana
```

```
            y pasamos al estado WaitingAck_RR
          - Si el buffer está vacío pasamos al estado NoWaitingAck_RR
            Si la variable nr no está confirmando, pasamos al estado
            WaitingAck_RR
        ----------------------------------------------------------------*/
        va = nr;
        esperandoRespuesta = FALSE;
        if ((frenteBtramasI != finalBtramasI) && (vs < (va + K) ))
        {
          /* Para formar el cabecero de la trama I: */
          nr = vr;
          ns = vs;
          FormarCabeceroTrama( &trama, &longTrama, I, ACTIVO, ACTIVO);
          /* Enviaremos un mensaje conteniendo la trama I al proceso
             multiplexor: */
          FormarMensajeMuxN2( &mensajeN2Mux, MUX_TR_DATOS_REQ,
            idProcesoN2, ventana.header, ventana.lengHeader,
            ventana.info, ventana.longInfo);
          KEscribePipe( pipeEscN2Mux, (void *)&mensajeN2Mux,
            LONG_MENSAJE_MUX_N2, KWAIT);
          KSignal( idProcesoMuxN2, MUX_TR_DATOS_REQ);
          numTxTramaI = 1;
          GuardaBtx( TIPO_I, trama, bufferTramasI[frenteBtramasI].info);
          vs++;
          ventana.header     = trama;
          ventana.lengHeader = longTrama;
          ventana.info       = bufferTramasI[frenteBtramasI].info;
          ventana.longInfo = bufferTramasI[frenteBtramasI].longInfo;
          if (++frenteBtramasI == LONG_BUF_TRAMAS_I)
            frenteBtramasI = 0;
          ventanaEsperandoTx = TRUE;
          KActivaContador( timer, T200);
          n200 = 0;
          return(WAITING_ACK_RR);
        }/*if((frenteBtramasI != finalBtramasI) && (vs<=(va + K)))*/
        else
          return(NO_WAITING_ACK_RR);
      } /*if (NR_VALIDO(nr, va, ns) && (nr == vs))*/
      return(WAITING_ACK_RNR);
    case RNR:
      KLiberaMemoria( (UWORD *)trama);
      if (NR_VALIDO(nr, va, ns))
        if (nr == vs)
        {
          /* Si (nr=vs) está confirmando la última trama I enviada
             y pasamos al estado NoWaitingAck_RNR: */
          va = nr;
          esperandoRespuesta = FALSE;
          n200 = 0;
          KActivaContador( timer, T203);
          if (bitPF == ACTIVO)
          {
            EnviarTramaCtrl( RR, INACTIVO, ACTIVO);
            KActivaContador( timer, T200);
          }
          else
            KActivaContador( timer, T203);
          return(NO_WAITING_ACK_RNR);
        }
        else
        {
          /*Si no está confirmando, permanecemos en el estado actual*/
          n200 = 0;
          if (bitPF == ACTIVO)
          {
            EnviarTramaCtrl( RR, INACTIVO, ACTIVO);
            KActivaContador( timer, T200);
          }
          else
            KActivaContador( timer, T203);
```

```
            return( NO_WAITING_ACK_RNR);
          }
        else /* if(NR_VALIDO(nr, va, ns)) */
          return(WAITING_ACK_RNR);
      case UI:
      case TIPO_GED_UI:
        TratarTramaUIrecibida( numIETrecibido, numIPASrecibido);
        return(WAITING_ACK_RNR);
      case DISC:
        KLiberaMemoria( (UWORD *)trama);
        if (bitPF == ACTIVO)
        {
          EnviarTramaCtrl( UA, INACTIVO, ACTIVO);
          vs = vr = va = 0;
          LiberaBtramasI();
          LiberaVentana();
          KSignal( idProcesoMuxN3, ED_LIBERACION_IND);
          KSignal( idProcesoMuxN2, MUX_TR_ENLACE_LIBERADO_REQ);
          return(ENLACE_LIBERADO);
        } /*if (biPF=ACTIVO)*/
        return(WAITING_ACK_RNR);
      case SABME:
        KLiberaMemoria( (UWORD *)trama);
        if (bitPF == ACTIVO)
        {
          EnviarTramaCtrl( UA, INACTIVO, ACTIVO);
          KActivaContador( timer, T203);
          vs = vr = va = 0;
          LiberaBtramasI();
          LiberaVentana();
          KSignal( idProcesoMuxN3, ED_ESTABLEC_IND);
          return( ENLACE_ESTABLECIDO);
        }  /*if (biPF=ACTIVO)*/
        else
          return(WAITING_ACK_RNR);
      default:
        /* Si recibimos cualquier otro tipo de trama, liberamos la
           memoria correspondiente: */
        KLiberaMemoria( (UWORD *)trama);
        return(WAITING_ACK_RNR);
    } /*switch(tipoTrama)*/
  case MUX_TR_DATOS_ENVIADOS_IND:
    /* En cualquier caso permanecemos en el estado actual: */
    return(WAITING_ACK_RNR);
  case ED_UNIDAD_DATOS_REQ:
    TratarEDunidadDatosReq();
    /* Permanecemos en el estado actual: */
    return(WAITING_ACK_RNR);
  case GED_UNIDAD_DATOS_REQ:
    TratarGEDunidadDatosReq();
    /* Permanecemos en el estado actual: */
    return(WAITING_ACK_RNR);
  case ED_DATOS_REQ:
    TratarEDdatosReq();
    /* Permanecemos en el estado actual: */
    return(WAITING_ACK_RNR);
  case ED_LIBERACION_REQ:
    LiberaBtramasI();
    LiberaVentana();
    /*-------------------------------------------------------------------
       Libero la trama de la ventana, sólo en los casos que estoy
       esperando acuse de recibo y por tanto, existe una trama I en la
       ventana de transmisión
    -------------------------------------------------------------------*/
    EnviarTramaCtrl( DISC, ACTIVO, ACTIVO);
    KActivaContador( timer, T200);
    vs = vr = va = 0;
    return(LIBERACION_PENDIENTE);
  case ED_LIBERACION_MEMO_REQ:
    KLeePipe( pipeLecN2N3, (VOID *)&mensaje23, LONG_MENSAJE23, KWAIT);
```

```
        KLiberaMemoria( (UWORD *)mensaje23.datos);
        /* Permanecemos en el estado actual: */
        return(WAITING_ACK_RNR);
      case GED_LIBERACION_MEMO_REQ:
        KLeePipe( pipeLecN2GED, (VOID *)&mensaje23, LONG_MENSAJE23, KWAIT) ;
        KLiberaMemoria( (UWORD *)mensaje23.datos);
        /* Permanecemos en el estado actual: */
        return(WAITING_ACK_RNR);
      case GED_SUPRESION_IET_REQ:
      case MUX_TR_ENLACE_LIBERADO_IND:
        KSignal( idProcesoMuxN3, ED_LIBERACION_IND);
        LiberaBtramasUI();
        LiberaBtramasI();
        LiberaVentana();
        return(ENLACE_LIBERADO);
      default:
        return(WAITING_ACK_RNR);
    } /*switch(signal)*/
  } /*else if (resp == KINTERRUP)*/
} /*WaitingAck_RNR*/


UWORD WaitingAck_RR( void)
{
  /*--- Definiciones variables locales a este "estado" (a esta funcion): ---*/
  UWORD signal;
  UWORD tipoTrama;
  UBYTE bitIR, bitPF;
  UBYTE numIETrecibido, numIPASrecibido;
  UWORD resp;
  UWORD error;          /* Error devuelto por el nivel físico */
  resp = KEsperaContador( timer);
  if (resp == KNO_ERROR)
  {
    if (ventanaEsperandoTx == TRUE)
    {
      if (numTxTramaI < N202)
      {
        if (EnviarTramaVentana() == TRUE)
          numTxTramaI++;
        KActivaContador( timer, T200);
        return(WAITING_ACK_RR);
      }
      else
      {
        LiberaBtramasCtrl();
        LiberaBtramasI();
        LiberaVentana();
        KSignal( idProcesoMuxN3, ED_LIBERACION_IND);
        KSignal( idProcesoMuxN2, MUX_TR_ENLACE_LIBERADO_REQ);
        return( ENLACE_LIBERADO);
      }
    }
    else /*if (ventanaEsperandoTx == TRUE)*/
    {
      if (n200 < N200)
      {
        nr = vr;   /* Para formar el cabecero de la trama RR */
        EnviarTramaCtrl(  RR, INACTIVO, ACTIVO);
        n200++;
        KActivaContador( timer, T200);
        esperandoRespuesta = TRUE;
        return(WAITING_ACK_RR);
      }
      else /*if (n200 < N200)*/
      {
        LiberaBtramasI();
        LiberaVentana();
        EnviarTramaCtrl( DM, INACTIVO, ACTIVO);
```

```
        KSignal( idProcesoMuxN3, ED_LIBERACION_IND);
        return(ENLACE_LIBERADO);
      }
    }
  }
  else /*if (resp == KNO_ERROR)*/
  {
    /*Se ha recibido un signal, por tanto se procesa:*/
    KLeeSignal( &signal);
    switch( signal)
    {
      case MUX_TR_DATOS_IND:
        KLeePipe( pipeLecN2Mux, (VOID *)(&mensajeN2Mux), LONG_MENSAJE_MUX_N2,
          KWAIT);
        tipoTrama = IdentificaTrama( mensajeN2Mux.header,
          mensajeN2Mux.lengHeader, &bitIR, &bitPF,
          &numIETrecibido, &numIPASrecibido);
        switch(tipoTrama)
        {
          case I:
            if ((ns == vr) && (NR_VALIDO(nr, va, ns)))
            {
              FormarMensaje23( &mensaje23, ED_DATOS_IND, longTrama, trama);
              KEscribePipe( pipeEscN2N3, (void *)&mensaje23,
                LONG_MENSAJE23, KWAIT);
              KSignal( idProcesoMuxN3, ED_DATOS_IND);
              vr++;
              if (nr == vs)
              {
                /* Si se cumple esta condición, está confirmando
                   la última trama I enviada: */
                va = nr;
                LiberaVentana();
                if ((frenteBtramasI != finalBtramasI) && (vs < (va + K)))
                {
                  /*-------------------------------------------------------------
                    - Si el bufferTramasI no está vacío, envío una trama I
                        y se añade al bufferTx y a la estructura ventana
                        y permanecemos en el estado WaitingAck_RR
                    - Si el buffer está vacío, es decir, no hay datos en
                        espera, pasamos al estado NoWaitingAck_RR
                  -------------------------------------------------------------*/
                  /* Para formar el cabecero de la trama I: */
                  nr = vr;
                  ns = vs;
                  esperandoRespuesta = TRUE;
                  FormarCabeceroTrama( &trama, &longTrama, I, ACTIVO, ACTIVO);
                  /* Enviaremos un mensaje conteniendo la trama I al proceso
                     multiplexor: */
                  FormarMensajeMuxN2( &mensajeN2Mux, MUX_TR_DATOS_REQ,
                    idProcesoN2, trama, longTrama,
                    bufferTramasI[frenteBtramasI].info,
                    bufferTramasI[frenteBtramasI].longInfo);
                  KEscribePipe( pipeEscN2Mux, (void *)&mensajeN2Mux,
                    LONG_MENSAJE_MUX_N2, KWAIT);
                  KSignal( idProcesoMuxN2, MUX_TR_DATOS_REQ);
                  numTxTramaI = 1;
                  GuardaBtx( TIPO_I, trama,
                    bufferTramasI[frenteBtramasI].info);
                  vs++;
                  ventana.header     = trama;
                  ventana.lengHeader = longTrama;
                  ventana.info       = bufferTramasI[frenteBtramasI].info;
                  ventana.longInfo   =
                    bufferTramasI[frenteBtramasI].longInfo;
                  if (++frenteBtramasI == LONG_BUF_TRAMAS_I)
                    frenteBtramasI = 0;
                  ventanaEsperandoTx = TRUE;
                  n200 = 0;
                  KActivaContador( timer, T200);
```

```
            return(WAITING_ACK_RR);
          }
          else /*if((frenteBtramasI!=finalBtramasI) && (vs<= ...)))*/
          {
            /* No hay datos que transmitir; por tanto pasamos al
               estado NoWaitingAck_RR: */
            nr = vr;  /* Para formar el cabecero de la trama RR */
            /* Si bitPF = 0 ó 1, se enviará una respuesta RR
               (bitIR = 1) y bitPF = 1: */
            EnviarTramaCtrl( RR, INACTIVO, ACTIVO);
            KActivaContador( timer, T203);
            return(NO_WAITING_ACK_RR);
          }
        }
        else /*if (nr == vs)*/
        {
          /* Si no confirma la última trama I enviada, permanecemos
             en el estado WaitingAck_RR: */
          nr = vr;  /* Para formar el cabecero de la trama RR */
          /* Si bitPF = 0 ó 1, se enviará una respuesta RR
             (bitIR = 1) y bitPF = 1: */
          EnviarTramaCtrl( RR, INACTIVO, ACTIVO);
          KActivaContador( timer, T203);
          return(WAITING_ACK_RR);
        }
      }
      else /*if ((ns == vr) && (NR_VALIDO(nr, va, ns)))*/
      {
        /* Si variables ns y nr de la trama I recibida no son
           correctas, descartamos la trama: */
        KLiberaMemoria( (UWORD *)trama);
        return(WAITING_ACK_RR);
      }
    case RR:
      KLiberaMemoria( (UWORD *)trama);
      if (NR_VALIDO(nr, va, ns) && (nr == vs))
      {
        /*-----------------------------------------------------------------
          Significa que está confirmando última trama enviada:
            - Si el bufferTramasI no está vacío, envío una trama I
                y se añade al bufferTx y a la estructura ventana
                y permanecemos en el estado WaitingAck_RR
            - Si el buffer está vacío pasamos al estado
                NoWaitingAck_RR
          Si la variable nr no está confirmando, permanecemos
            también en el estado WaitingAck_RR
        -----------------------------------------------------------------*/
        va = nr;
        esperandoRespuesta = FALSE;
        if ((frenteBtramasI != finalBtramasI) && (vs < (va + K) ))
        {
          /* Para formar el cabecero de la trama I: */
          nr = vr;
          ns = vs;
          esperandoRespuesta = TRUE;
          FormarCabeceroTrama( &trama, &longTrama, I, ACTIVO, ACTIVO);
          /* Enviaremos un mensaje conteniendo la trama I al proceso
             multiplexor: */
          FormarMensajeMuxN2( &mensajeN2Mux, MUX_TR_DATOS_REQ,
            idProcesoN2, trama, longTrama,
            bufferTramasI[frenteBtramasI].info,
            bufferTramasI[frenteBtramasI].longInfo);
          KEscribePipe( pipeEscN2Mux, (void *)&mensajeN2Mux,
            LONG_MENSAJE_MUX_N2, KWAIT);
          KSignal( idProcesoMuxN2, MUX_TR_DATOS_REQ);
          numTxTramaI = 1;
          GuardaBtx( TIPO_I, trama,
            bufferTramasI[frenteBtramasI].info);
          vs++;
          ventana.header      = trama;
```

```
        ventana.lengHeader = longTrama;
        ventana.info      = bufferTramasI[frenteBtramasI].info;
        ventana.longInfo = bufferTramasI[frenteBtramasI].longInfo;
        if (++frenteBtramasI == LONG_BUF_TRAMAS_I)
          frenteBtramasI = 0;
        ventanaEsperandoTx = TRUE;
        KActivaContador( timer, T200);
        n200 = 0;
        return(WAITING_ACK_RR);
      }/*if((frenteBtramasI!=finalBtramasI) && (vs <= (va+K)))*/
      else
        return(NO_WAITING_ACK_RR);
    } /*if (NR_VALIDO(nr, va, ns) && (nr == vs))*/
    else
      return(WAITING_ACK_RR);
  case RNR:
    KLiberaMemoria( (UWORD *)trama);
    if (NR_VALIDO(nr, va, ns))
    {
      if (nr == vs)
      {
        /* Si (nr=vs) está confirmando la última trama I enviada y
           pasamos al estado NoWaitingAck_RNR: */
        va = nr;
        esperandoRespuesta = FALSE;
        n200 = 0;
        KActivaContador( timer, T203);
        if (bitPF == ACTIVO)
        {
          EnviarTramaCtrl( RR, INACTIVO, ACTIVO);
          KActivaContador( timer, T200);
        }
        else
          KActivaContador( timer, T203);
          return(NO_WAITING_ACK_RNR);
      }
      else /*if (nr == vs)*/
      {
        /* Si no está confirmando, pasamos al estado WaitingAck_RNR*/
        n200 = 0;
        KActivaContador( timer, T203);
        if (bitPF == ACTIVO)
        {
          EnviarTramaCtrl( RR, INACTIVO, ACTIVO);
          KActivaContador( timer, T200);
        }
        else
          KActivaContador( timer, T203);
        return(WAITING_ACK_RNR);
      }
    }
    else /*if (NR_VALIDO(nr, va, ns))*/
      /* Si la trama RNR recibida no posee un nr válido, se ignora
         y permanecemos en el estado actual: */
      return(WAITING_ACK_RR);
  case REJ:
    if (NR_VALIDO(nr, va, ns))
    {
      /* Si la trama REJ recibida es una intrucción, se enviará
         una trama RR respuesta: */
      if ((bitIR == ACTIVO) && (bitPF == ACTIVO))
        EnviarTramaCtrl( RR, INACTIVO, ACTIVO);
        /* En cualquier caso se reenviará la última trama I
           (trama I de la ventana): */
      /* Enviaremos un mensaje conteniendo la trama I al proceso
         multiplexor: */
      FormarMensajeMuxN2( &mensajeN2Mux, MUX_TR_DATOS_REQ,
        idProcesoN2, ventana.header, ventana.lengHeader,
        ventana.info, ventana.longInfo);
      KEscribePipe( pipeEscN2Mux, (void *)&mensajeN2Mux,
```

```
            LONG_MENSAJE_MUX_N2, KWAIT);
          KSignal( idProcesoMuxN2, MUX_TR_DATOS_REQ);
          numTxTramaI = 1;
          GuardaBtx( TIPO_I, ventana.header, ventana.info);
          ventanaEsperandoTx = TRUE;
          esperandoRespuesta = TRUE;
          n200 = 0;
          KActivaContador( timer, T200);
          return(WAITING_ACK_RR);
        }
        else /*if (NR_VALIDO(nr, va, ns))*/
          return(WAITING_ACK_RR);
      case UI:
      case TIPO_GED_UI:
        TratarTramaUIrecibida( numIETrecibido, numIPASrecibido);
        return(WAITING_ACK_RR);
      case DISC:
        KLiberaMemoria( (UWORD *)trama);
        if (bitPF == ACTIVO)
        {
          EnviarTramaCtrl( UA, INACTIVO, ACTIVO);
          vs = vr = va = 0;
          LiberaBtramasI();
          LiberaVentana();
          KSignal( idProcesoMuxN3, ED_LIBERACION_IND);
          return(ENLACE_LIBERADO);
        } /*if (biPF=ACTIVO)*/
        return(WAITING_ACK_RR);
      case SABME:
        KLiberaMemoria( (UWORD *)trama);
        if (bitPF == ACTIVO)
        {
          EnviarTramaCtrl( UA, INACTIVO, ACTIVO);
          KActivaContador( timer, T203);
          vs = vr = va = 0;
          LiberaBtramasI();
          LiberaVentana();
          KSignal( idProcesoMuxN3, ED_ESTABLEC_IND);
          return( ENLACE_ESTABLECIDO);
        }  /*if (biPF=ACTIVO)*/
        else
          return(WAITING_ACK_RR);
      default:
        /* Si recibimos cualquier otro tipo de trama, liberamos la
           memoria correspondiente: */
        KLiberaMemoria( (UWORD *)trama);
        return(WAITING_ACK_RR);
    } /*switch(tipoTrama)*/
  case ED_LIBERACION_REQ:
    LiberaBtramasI();
    LiberaVentana();
    /*-------------------------------------------------------------------
       Libero la trama de la ventana, sólo en los casos que estoy
       esperando acuse de recibo y por tanto, existe una trama I en la
       ventana de transmisión
    -------------------------------------------------------------------*/
    EnviarTramaCtrl( DISC, ACTIVO, ACTIVO);
    KActivaContador( timer, T200);
    vs = vr = va = 0;
    return(LIBERACION_PENDIENTE);
  case ED_DATOS_REQ:
    TratarEDdatosReq();
    /* Permanecemos en el estado actual: */
    return(WAITING_ACK_RR);
  case MUX_TR_DATOS_ENVIADOS_IND:
    TratarMuxTRdatosEnviadosInd();
    /* Permanecemos en el estado actual: */
    return(WAITING_ACK_RR);
  case ED_UNIDAD_DATOS_REQ:
    TratarEDunidadDatosReq();
```

```
        /* Permanecemos en el estado actual: */
        return(WAITING_ACK_RR);
      case GED_UNIDAD_DATOS_REQ:
        TratarGEDunidadDatosReq();
        /* Permanecemos en el estado actual: */
        return(WAITING_ACK_RR);
      case ED_LIBERACION_MEMO_REQ:
        KLeePipe( pipeLecN2N3, (VOID *)&mensaje23, LONG_MENSAJE23, KWAIT);
        KLiberaMemoria( (UWORD *)mensaje23.datos);
        /* Permanecemos en el estado actual: */
        return(WAITING_ACK_RR);
      case GED_LIBERACION_MEMO_REQ:
        KLeePipe( pipeLecN2GED, (VOID *)&mensaje23, LONG_MENSAJE23, KWAIT) ;
        KLiberaMemoria( (UWORD *)mensaje23.datos);
        /* Permanecemos en el estado actual: */
        return(WAITING_ACK_RR);
      case GED_SUPRESION_IET_REQ:
      case MUX_TR_ENLACE_LIBERADO_IND:
        KSignal( idProcesoMuxN3, ED_LIBERACION_IND);
        LiberaBtramasUI();
        LiberaBtramasI();
        LiberaVentana();
        return(ENLACE_LIBERADO);
      default:
        /* Permanecemos en el estado actual: */
        return(WAITING_ACK_RR);
    } /*switch(signal)*/
  }  /*else if (resp == KINTERRUP)*/
} /*WaitingAck_RR*/


UWORD LiberacionPendiente( void)
{
  /*--- Definiciones variables locales a este "estado" (a esta funcion): ---*/
  UWORD tipoTrama;
  UBYTE bitIR, bitPF;
  UBYTE numIETrecibido, numIPASrecibido;
  UWORD error;
  UWORD resp;
  UWORD n200 = 0;  /*Inicializo número de retransmisiones de la trama DISC*/
  while( n200 < N200)  /*Hasta N200 = 3 retransmisiones de la trama DISC*/
  {
    resp = KEsperaContador(timer);
    if (resp == KNO_ERROR)  /*terminó el temporizador*/
    {
      EnviarTramaCtrl( DISC, ACTIVO, ACTIVO);
      KActivaContador( timer, T200);
      vs = vr = va = 0;
      n200++;
      /*-------------------------------------------------------------------------
         Permanecemos en el estado actual (LiberacionPendiente)
         hasta completar las N200 retransmisiones, o al llegar la trama
         adecuada, por evolución a otro estado
      -------------------------------------------------------------------------*/
      continue;  /*Salta a evaluar denuevo la condición del while*/
    } /*if (resp == KNO_ERROR)*/
    if (resp == KINTERRUP)
    {
      KLeeSignal(&signal);
      switch(signal)
      {
        case MUX_TR_DATOS_IND:
          KLeePipe( pipeLecN2Mux, (VOID *)(&mensajeN2Mux), LONG_MENSAJE_MUX_N2,
            KWAIT);
          tipoTrama = IdentificaTrama( mensajeN2Mux.header,
            mensajeN2Mux.lengHeader, &bitIR, &bitPF,
            &numIETrecibido, &numIPASrecibido);
          switch(tipoTrama)
          {
```

```
      case UI:
      case TIPO_GED_UI:
        TratarTramaUIrecibida( numIETrecibido, numIPASrecibido);
        continue;
      case UA:
      case DM:
        KLiberaMemoria( (UWORD*)trama);
        if (bitPF == ACTIVO)
        {
          KSignal( idProcesoMuxN3, ED_LIBERACION_CONF);
          return(ENLACE_LIBERADO);
        }
        else continue;
      case DISC:
        KLiberaMemoria( (UWORD*)trama);
        if (bitPF == ACTIVO)
        {
          /*-------------------------------------------------------------
            Respondemos enviando una trama UA, pero no pasaremos al
            estado EnlaceLiberado hasta que no recibamos una trama UA
          -------------------------------------------------------------*/
          EnviarTramaCtrl( UA, INACTIVO, ACTIVO);
          KSignal( idProcesoMuxN3, ED_LIBERACION_CONF);
        }
        continue;
      case SABME:
        KLiberaMemoria( (UWORD*)trama);
        if (bitPF == ACTIVO)
        {
          EnviarTramaCtrl( DM, ACTIVO, INACTIVO);
          KSignal( idProcesoMuxN3, ED_LIBERACION_CONF);
          return(ENLACE_LIBERADO);
        }
        else continue;
      default:
        /* Si recibimos cualquier otro tipo de trama, liberamos la
           memoria correspondiente: */
        KLiberaMemoria( (UWORD *)trama);
        continue;
    } /*switch(tipoTrama)*/
  case MUX_TR_DATOS_ENVIADOS_IND:
    TratarMuxTRdatosEnviadosInd();
    /* Permanecemos en el estado actual: */
    continue;
  case ED_UNIDAD_DATOS_REQ:
    TratarEDunidadDatosReq();
    /* Permanecemos en el estado actual: */
    continue;
  case GED_UNIDAD_DATOS_REQ:
    TratarGEDunidadDatosReq();
    /* Permanecemos en el estado actual: */
    continue;
  case ED_LIBERACION_MEMO_REQ:
    KLeePipe( pipeLecN2N3, (VOID *)&mensaje23, LONG_MENSAJE23, KWAIT);
    KLiberaMemoria( (UWORD *)mensaje23.datos);
    /* Permanecemos en el estado actual: */
    continue;
  case GED_LIBERACION_MEMO_REQ:
    KLeePipe( pipeLecN2GED, (VOID *)&mensaje23, LONG_MENSAJE23, KWAIT) ;
    KLiberaMemoria( (UWORD *)(mensaje23.datos));
    /* Permanecemos en el estado actual: */
    continue;
  case GED_SUPRESION_IET_REQ:
  case MUX_TR_ENLACE_LIBERADO_IND:
    KSignal( idProcesoMuxN3, ED_LIBERACION_CONF);
    LiberaBtramasUI();
    return(ENLACE_LIBERADO);
  default:
    /* Permanecemos en el estado actual: */
    continue;
```

```
      } /*switch(signal)*/
    } /*if (resp == KINTERRUP)*/
  } /*while(n200 < N200)*/
  /*-------------------------------------------------------------------------
     Si no se recibió una respuesta correcta después de enviar N200
     retransmisiones de la trama DISC, se pasa al estado ENLACE_LIBERADO
     y se indica al nivel 3 con el signal ED_LIBERACION_CONF.
  -------------------------------------------------------------------------*/
  KSignal( idProcesoMuxN3, ED_LIBERACION_CONF);
  return(ENLACE_LIBERADO);
} /*LiberacionPendiente*/

/*------------------- Fin definiciones de funciones ------------------------*/
```

```
/*****************************************************************************/
/*                                GED_TR.C                                   */
/*          TRABAJO FIN DE CARRERA  SANTIAGO MARTIN ROMANI                   */
/*    IMPLEMENTACION DEL GESTOR DE ENLACE DE DATOS DEL TERMINAL DE RED       */
/*****************************************************************************/


#define UNIX

/*------------ FICHEROS INCLUIDOS: ------------------------------------------*/
#include "basico.h"    /* ..\kernel\basico.h */
#include "kernel.h"    /* ..\kernel\kernel.h */
#include "inicio.h"
#include "nivel2.hp"
#include "n2n3.h"
#include "ged.hp"



/*------------ DECLARACION DE VARIABLES GLOBALES: --------------------------*/
UWORD signal;
UWORD estado;
/* Temporizador utilizado en el proceso de Gestor de Enlace de Datos del lado
   del Terminal de Red: */
UWORD timerGED_TR;
UWORD tramaIdentificada;

UBYTE tipoMensaje;
UBYTE indicadorAccion;


/*-----------------------------------------------------------------------------
  Identificadores utilizados en el proceso de comunicación entre la entidad de
  Gestión de Enlace de Datos y la entidad de Enlace de Datos del lado del
  Terminal de Red:
-----------------------------------------------------------------------------*/
UBYTE idProcesoN2;  /*id. entidad de Enlace de Datos*/
UBYTE idProcesoGED; /*id. entidad de Gestión de Enlace de Datos*/

UBYTE *trama;
UWORD longTrama;
UWORD seed;
MENSAJE23 mensaje23;
UWORD numRef;

/* Vector de números de IETs asignados a equipos terminales no automáticos: */
UWORD vecIETsAsignados[MAX_IETS_ASIGNADOS];

/*------ Pipes de comunicacion con la entidad de nivel del lado TR: ----------
  Sobre el pipe de escritura se mandan los mensajes a la entidad de nivel 2
  del lado del Terminal de Red, a través del pipe de lectura se reciben los
  mensajes procedentes de dicha entidad de nivel 2:
-----------------------------------------------------------------------------*/
UWORD pipeEscGED, pipeLecGED;


/*------------ Definiciones de los distintos buffers utilizados: -------------
  Buffers (estructuras de tipo cola) utilizados en el proceso de Gestión de
  Enlace de Datos:  bufferPeticionesId y bufferVerificacionesId
  Representan una cola de peticiones de identidad y verificaciones de
  identidad solicitadas por el Terminal de Usuario que se encuentran en
  espera:
    1.- bufferPeticionesId:
        Almacena número de referencia de cada petición de identidad
    2.- bufferVerificacionesId:
        Almacena número de IET que hay que verificar
-----------------------------------------------------------------------------*/
UWORD bufferPeticionesId[LONG_BUF_PETICIONES_ID];
UWORD frenteBpeticionesId, finalBpeticionesId;
```

```
UBYTE bufferVerificacionesId[LONG_BUF_VERIFICACIONES_ID];
UWORD frenteBverificacionesId, finalBverificacionesId;



/*------------ DEFINICIONES DE PROTOTIPOS DE FUNCIONES: --------------------*/
void  n2_GED_TRmain( UWORD pipeEscrituraGED, UWORD pipeLecturaGED,
  UBYTE identificadorN2);
UWORD Idle( void);
UWORD EsperaRespPruebaIdIETdeGrupo( void);
UWORD EsperaRespPruebaIdIETconcreto( void);

void  FormarTramaGED_UI( UBYTE **trama, UWORD *longTrama, UWORD *numRef,
  UBYTE tipoMessage, UBYTE indicadorAccion);

UWORD IdentificaTramaGED_UI( UBYTE *frame, UWORD frameLength, UWORD *numRef,
  UBYTE *tipoMensaje, UBYTE *indicadorAccion);
UBYTE GetNewIET(void);
void  FormarMensaje23( MENSAJE23 *message23, UWORD tipoMens, UWORD longDatos,
  UBYTE *datos);

void  EnviarGEDliberacionMemoReq( void);
void  EnviarGEDunidadDatosReq( void);

UWORD GuardaBverificacionesId( UBYTE numIET);
UBYTE RecuperaBverificacionesId( void);
UWORD GuardaBpeticionesId( UWORD numRef);
UWORD RecuperaBpeticionesId( void);

void InicializaParametros( void);



/*------------ DECLARACION DE FUNCIONES: -----------------------------------*/

void  n2_GED_TRmain( pipeEscrituraGED, pipeLecturaGED, identificadorN2)
UWORD pipeEscrituraGED, pipeLecturaGED;
UWORD identificadorN2;
{
  /*-------------------------------------------------------------------------
    A partir de esta función se ejecutará el proceso que representa a la
    entidad de gestión de Capa de Enlace de Datos del lado del Terminal de Red
  -------------------------------------------------------------------------*/
  UBYTE tipoMensaje;
  UBYTE indicadorAccion;
  UWORD i;

  /*-------------------------------------------------------------------------
     Parámetros de Entrada al Proceso que representa a la entidad del Gestor
     de Enlace de Datos del lado del Terminal de Red:
  -------------------------------------------------------------------------*/
  pipeEscGED  = pipeEscrituraGED;
  pipeLecGED  = pipeLecturaGED;
  idProcesoN2 = identificadorN2;

  /*-------------------------------------------------------------------------
    Solicitamos al KERNEL el identificador de nuestro proceso de Gestor de
    Enlace de Datos que posteriormente indicamos al proceso asociado del
    nivel 2:
  -------------------------------------------------------------------------*/
  idProcesoGED = KIDProceso();

  /*---------- Envío del primer mensaje al proceso de nivel 2 (lado TR): -----
    Se utiliza para indicar al proceso que representa la entidad de nivel 2
    del lado del Terminal de Red el pipe de donde se leerán los mensajes
    procedentes de la misma (pipe de escritura para el nivel 2);
    también se indica el identificador de este proceso de Gestión de Enlace
    de Datos para la comunicación entre ambos procesos:
  -------------------------------------------------------------------------*/
```

```
  FormarMensaje23( &mensaje23, idProcesoGED, pipeLecGED, NULL);
  KEscribePipe( pipeEscGED, (void*)&mensaje23, LONG_MENSAJE23, KWAIT);

  /*-------------------------------------------------------------------------
    Se solicita el temporizador necesario para el proceso de la comunicación
     del proceso de Gestor de Enlace de Datos del lado del Terminal de Red:
  -------------------------------------------------------------------------*/
  KSolicitaContador( &timerGED_TR);
  InicializaParametros();
  estado = IDLE;
  while(TRUE)   /*El proceso vive para siempre*/
  {
    switch(estado)
    {
      case IDLE:
        estado = Idle();
        break;
      case ESPERA_RESP_PRUEBA_ID_IET_DE_GRUPO:
        estado = EsperaRespPruebaIdIETdeGrupo();
        break;
      case ESPERA_RESP_PRUEBA_ID_IET_CONCRETO:
        estado = EsperaRespPruebaIdIETconcreto();
        break;
    }
  }
} /*n2_GED_TRmain*/

void InicializaParametros( void)
{
  UWORD i;
  /*-------------------------------------------------------------------------
     Inicializamos los elementos del vector de números de IETs asignados a
     NO_ASIGNADO:
  -------------------------------------------------------------------------*/
  for(i=0; i<MAX_IETS_ASIGNADOS; i++)
    vecIETsAsignados[i] = NO_ASIGNADO;

  /*---------- Inicialización índices buffers: ----------------------------*/
  frenteBpeticionesId     = finalBpeticionesId     = 0;
  frenteBverificacionesId = finalBverificacionesId = 0;
}


void  FormarTramaGED_UI( trama, longTrama, numRef, tipoMessage,
  indicadorAccion)
UBYTE **trama, indicadorAccion, tipoMessage;
UWORD *longTrama, *numRef;
{
  *longTrama = LONG_GED_UI_INFO;
  KPideMemoria( *longTrama, (UWORD**)(&(*trama)));
  /*Los cabeceros de las tramas GED_UI los añade la entidad de Capa de Enlace
    de Datos*/
  (*trama)[0] = IDENTIFICADOR_GED;
  switch(tipoMessage)
  {
    case PETICION_PRUEBA_IDENTIDAD:
    case SUPRESION_IDENTIDAD:
    case VERIFICACION_IDENTIDAD:
      /*-----------------------------------------------------------------
        En estos casos los dos octetos correspondientes al número de
        referencia van codificados a cero:
      -----------------------------------------------------------------*/
      (*trama)[1] = OCTETO_NULO;
      (*trama)[2] = OCTETO_NULO;
      break;
    case IDENTIDAD_ASIGNADA:
    case IDENTIDAD_RECHAZADA:
      /*-----------------------------------------------------------------
         En estos casos, se responde a la entidad de gestión remota con el
         número de referencia enviado previamente en el mensaje al que estamos
```

```
        respondiendo:
      ------------------------------------------------------------------------*/
      (*trama)[1] = (UBYTE)(MAKE_MSB_NUM_REF( *numRef));
      (*trama)[2] = (UBYTE)(MAKE_LSB_NUM_REF( *numRef));
      break;
    case PETICION_IDENTIDAD:
    case RESPUESTA_PRUEBA_IDENTIDAD:
      /*-----------------------------------------------------------------------
        En el caso de querer enviar un mensaje de PETICION DE IDENTIDAD o de
          RESPUESTA_PRUEBA_IDENTIDAD estos dos octetos contendrán un valor
          aleatorio de 16bits (de 0 a 65535)
        Este valor llamado número de referencia, se utiliza para establecer
          una diferencia entre una serie de equipos de usuario que pueden
          solicitar simultámenamente la asignación de un valor IET.
      ------------------------------------------------------------------------*/
      *numRef  = rand();
      (*trama)[1] = (UBYTE)(MAKE_MSB_NUM_REF( *numRef));
      (*trama)[2] = (UBYTE)(MAKE_LSB_NUM_REF( *numRef));
      break;
  } /*switch(tipoMessage)*/
  (*trama)[3] = tipoMessage;
  (*trama)[4] = MAKE_BYTE_INDICADOR_ACCION( indicadorAccion);
} /*FormarTramaGED_UI*/


UWORD IdentificaTramaGED_UI( frame, frameLength, numRef, tipoMensaje,
  indicadorAccion)
UBYTE *frame, *tipoMensaje, *indicadorAccion;
UWORD frameLength, *numRef;
{
  if (frame[2] != IDENTIFICADOR_GED)
    return(FALSE);
  *numRef = GET_NUM_REF( frame[3], frame[4]);
  *indicadorAccion = GET_INDICADOR_ACCION(frame[6]);;
  switch(frame[5])
  {
    case PETICION_IDENTIDAD:
      if (*indicadorAccion != IET_DE_GRUPO)
        return(FALSE);
      *tipoMensaje = PETICION_IDENTIDAD;
      break;
    case IDENTIDAD_ASIGNADA:
      *tipoMensaje = IDENTIDAD_ASIGNADA;
      break;
    case IDENTIDAD_RECHAZADA:
      *tipoMensaje = IDENTIDAD_RECHAZADA;
      break;
    case PETICION_PRUEBA_IDENTIDAD:
      if ((frame[3] != OCTETO_NULO) || (frame[4] != OCTETO_NULO))
        return(FALSE);
      *tipoMensaje = PETICION_PRUEBA_IDENTIDAD;
      break;
    case RESPUESTA_PRUEBA_IDENTIDAD:
      *tipoMensaje = RESPUESTA_PRUEBA_IDENTIDAD;
      break;
    case SUPRESION_IDENTIDAD:
      if ((frame[3] != OCTETO_NULO) || (frame[4] != OCTETO_NULO))
        return(FALSE);
      *tipoMensaje = SUPRESION_IDENTIDAD;
      break;
    case VERIFICACION_IDENTIDAD:
      if ((frame[3] != OCTETO_NULO) || (frame[4] != OCTETO_NULO))
        return(FALSE);
      *tipoMensaje = VERIFICACION_IDENTIDAD;
      break;
    default:
      return(FALSE);
  }
  if ((frameLength > (ADR_LENGTH + LONG_GED_UI_INFO)) &&
    (*tipoMensaje != RESPUESTA_PRUEBA_IDENTIDAD))
```

```
    return(FALSE);
  return(TRUE);
} /*IdentificaTramaGED_UI*/


UBYTE GetNewIET(void)
{
  /*-------------------------------------------------------------------------
    Devolverá un valor libre de IET del rango 64 a 126 o el IET_DE_GRUPO (127)
       indicando que no hay ningún IET disponible
  -------------------------------------------------------------------------*/
  UBYTE i;
  for (i=0; i<MAX_IETS_ASIGNADOS; i++)
    if (vecIETsAsignados[i] == NO_ASIGNADO)
    {
      vecIETsAsignados[i] = ASIGNADO;
      return( i + DIFERENCIA_RANGO);
    }
 return( IET_DE_GRUPO);
}


void FormarMensaje23( message23, tipoMens, longDatos, datos)
MENSAJE23 *message23;
UWORD tipoMens, longDatos;
UBYTE *datos;
{
  message23->tipo      = tipoMens;
  message23->longDatos = longDatos;
  message23->datos     = datos;
}


void EnviarGEDliberacionMemoReq( void)
{
  FormarMensaje23( &mensaje23, GED_LIBERACION_MEMO_REQ, mensaje23.longDatos,
    mensaje23.datos);
  KEscribePipe( pipeEscGED, (void *)&mensaje23, LONG_MENSAJE23, KWAIT);
  KSignal( idProcesoN2, GED_LIBERACION_MEMO_REQ);
}


void EnviarGEDunidadDatosReq( void)
{
  FormarMensaje23( &mensaje23, GED_UNIDAD_DATOS_REQ, longTrama,
    trama);
  KEscribePipe( pipeEscGED, (void *)&mensaje23, LONG_MENSAJE23,
    KWAIT);
  KSignal( idProcesoN2, GED_UNIDAD_DATOS_REQ);
}


UWORD GuardaBverificacionesId( numIET)
UBYTE numIET;
{
  /*-------------------------------------------------------------------------
     Inserta un elemento en la cola de Verificaciones de Identidad en espera
     y devuelve TRUE. Sí la cola está llena, devuelve FALSE
  -------------------------------------------------------------------------*/
  /* Comprobamos sí cola llena: */
  if ((frenteBverificacionesId == finalBverificacionesId+1) ||
    ((frenteBverificacionesId == 0) &&
    (finalBverificacionesId == LONG_BUF_VERIFICACIONES_ID-1)))
    return(FALSE);
  else
  {
    bufferVerificacionesId[finalBverificacionesId] = numIET;
    if ((++finalBverificacionesId) == LONG_BUF_VERIFICACIONES_ID)
      finalBverificacionesId = 0;
    return(TRUE);
```

```
  }
}


UBYTE RecuperaBverificacionesId( void)
{
  /*-------------------------------------------------------------------------
     Devuelve el valor de IET de la primera conexión de Enlace de Datos
     que está esperando una verificación de identidad
  -------------------------------------------------------------------------*/
  UBYTE numIETaux;
  numIETaux = bufferVerificacionesId[frenteBverificacionesId];
  if (++frenteBverificacionesId == LONG_BUF_VERIFICACIONES_ID)
    frenteBverificacionesId = 0;
  return(numIETaux);
}

UWORD RecuperaBpeticionesId( void)
{
  /*-------------------------------------------------------------------------
    Devuelve el valor del número de referencia de la primera petición de
    identidad en espera
  -------------------------------------------------------------------------*/
  UWORD numRefAux;
  numRefAux = bufferPeticionesId[frenteBpeticionesId];
  if (++frenteBpeticionesId == LONG_BUF_PETICIONES_ID)
    frenteBpeticionesId = 0;
  return(numRefAux);
}


UWORD GuardaBpeticionesId( numRef)
UBYTE numRef;
{
  /*-------------------------------------------------------------------------
    Inserta un elemento en la cola de Peticiones de Identidad en espera
    y devuelve TRUE. Sí la cola está llena, devuelve FALSE
  -------------------------------------------------------------------------*/
  /* Comprobamos sí cola llena: */
  if ((frenteBpeticionesId == finalBpeticionesId+1) ||
    ((frenteBpeticionesId == 0) &&
    (finalBpeticionesId == LONG_BUF_PETICIONES_ID-1)))
    return(FALSE);
  else
  {
    bufferPeticionesId[finalBpeticionesId] = numRef;
    if ((++finalBpeticionesId) == LONG_BUF_PETICIONES_ID)
      finalBpeticionesId = 0;
    return(TRUE);
  }
}


UWORD Idle( void)
{
  UWORD numRef;
  UBYTE indicadorAccion;
  UBYTE newIET;
  UWORD i;
  KWait();
  KLeeSignal( &signal);
  switch(signal)
  {
    case GED_UNIDAD_DATOS_IND:
      KLeePipe( pipeLecGED, &mensaje23, LONG_MENSAJE23, KWAIT) ;
      tramaIdentificada = IdentificaTramaGED_UI( mensaje23.datos,
        mensaje23.longDatos, &numRef, &tipoMensaje, &indicadorAccion);
      EnviarGEDliberacionMemoReq();
      if (tramaIdentificada == TRUE)
      {
```

```
        switch(tipoMensaje)
        {
          case PETICION_IDENTIDAD:
            if ((newIET = GetNewIET()) != IET_DE_GRUPO)
            {
              /*-----------------------------------------------------------
                 Sí existe algún IET disponible para la asignación, se envía
                 a la entidad de gestión de capa del lado del terminal de red
                 y a la entidad de nivel 2:
              -----------------------------------------------------------*/
              FormarTramaGED_UI( (UBYTE **)trama, &longTrama, &numRef,
                IDENTIDAD_ASIGNADA, newIET);
              EnviarGEDunidadDatosReq();
              FormarMensaje23( &mensaje23, GED_ASIGNACION_IET_REQ,
                newIET, NULL);
              KEscribePipe( pipeEscGED, (void *)&mensaje23, LONG_MENSAJE23,
                KWAIT);
              KSignal( idProcesoN2, GED_ASIGNACION_IET_REQ);
              return(IDLE);
            }
            else
            {
              /*-----------------------------------------------------------
                 Sí no hay ningún IET disponible para asignación, se llevará
                 a cabo una Petición de Prueba de Identidad con IET_DE_GRUPO
                 y se evoluciona al estado EsperaRespPruebaIdIETdeGrupo:
              -----------------------------------------------------------*/
              FormarTramaGED_UI( (UBYTE **)trama, &longTrama, &numRef,
                PETICION_PRUEBA_IDENTIDAD , IET_DE_GRUPO);
              EnviarGEDunidadDatosReq();
              KActivaContador( timerGED_TR, T201);
              for(i=0; i<MAX_IETS_ASIGNADOS; i++)
                vecIETsAsignados[i] = NO_ASIGNADO;
              return(ESPERA_RESP_PRUEBA_ID_IET_DE_GRUPO);
            }
          case VERIFICACION_IDENTIDAD:
            /*-----------------------------------------------------------
              1.- El Indicador de Acción recibido en el mensaje de
                  Verificación de Identidad es el número de IET que hay
                  que probar
              2.- Guardamos el indicador de acción para luego recuperarlo
                  Esto es necesario para optimizar el código de la
                  función-estado EsperaRespPruebaIdIETconcreto:
            -----------------------------------------------------------*/
            GuardaBverificacionesId( indicadorAccion);
            return(ESPERA_RESP_PRUEBA_ID_IET_CONCRETO);
          default:
            EnviarGEDliberacionMemoReq();
            return(IDLE);
        } /*switch(tipoMensaje)*/
      }
      else /*tramaIdentificada == TRUE*/
        return(IDLE);
    case GED_LIBERACION_MEMO_IND:
      KLeePipe( pipeLecGED, &mensaje23, LONG_MENSAJE23, KWAIT) ;
      KLiberaMemoria( (UWORD *)mensaje23.datos);
      return(IDLE);
    default:
      return(IDLE);
  }
} /*Idle*/


UWORD EsperaRespPruebaIdIETdeGrupo( void)
{
  UWORD numRef;
  UBYTE tipoMensaje;
  UBYTE indicadorAccion;
  UBYTE newIET;
```

```
UWORD n204ByMe;
/* Número de respuestas al mensaje de prueba de identidad del IET concreto*/
UWORD respsPruebaIdentidad;
UWORD i;
for( i=0; i<MAX_IETS_ASIGNADOS; i++)
  vecIETsAsignados[i] = NO_ASIGNADO;
for( n204ByMe=1, respsPruebaIdentidad = 0; n204ByMe<N204_BY_ME; n204ByMe++)
{
  while(KEsperaContador(timerGED_TR) != KNO_ERROR)
  {
    KLeeSignal(&signal);
    switch(signal)
    {
      case GED_UNIDAD_DATOS_IND:
        KLeePipe( pipeLecGED, &mensaje23, LONG_MENSAJE23, KWAIT) ;
        tramaIdentificada = IdentificaTramaGED_UI( mensaje23.datos,
          mensaje23.longDatos, &numRef, &tipoMensaje, &indicadorAccion);
        if (tramaIdentificada == TRUE)
          switch(tipoMensaje)
          {
            case PETICION_IDENTIDAD:
              GuardaBpeticionesId( numRef);
              EnviarGEDliberacionMemoReq();
              break;
            case VERIFICACION_IDENTIDAD:
              GuardaBverificacionesId( indicadorAccion);
              EnviarGEDliberacionMemoReq();
              break;
            case RESPUESTA_PRUEBA_IDENTIDAD:
              do{
                i = POSICION_1ER_INDICADOR_ACCION;
                indicadorAccion = GET_INDICADOR_ACCION(trama[i]);
                respsPruebaIdentidad++;
                if (vecIETsAsignados[indicadorAccion - DIFERENCIA_RANGO] ==
                  NO_ASIGNADO)
                  vecIETsAsignados[indicadorAccion - DIFERENCIA_RANGO] =
                    ASIGNADO;
                else
                {
                  /* Enviar mensaje de Supresión de Identidad: */
                  FormarTramaGED_UI( (UBYTE **)trama, &longTrama, &numRef,
                    SUPRESION_IDENTIDAD, indicadorAccion);
                  EnviarGEDunidadDatosReq();
                  EnviarGEDunidadDatosReq();
                  vecIETsAsignados[indicadorAccion - DIFERENCIA_RANGO] =
                    NO_ASIGNADO;
                  KSignal( idProcesoN2, GED_SUPRESION_IET_REQ);
                }
                i++;
              }while((GET_LSBIT_OCTETO_INDICADOR_ACCION(trama[i])==INACTIVO)
                || (i<longTrama));
              EnviarGEDliberacionMemoReq();
              break;
            default:
              EnviarGEDliberacionMemoReq();
              break;
          } /*switch(tipoMensaje)*/
      case GED_LIBERACION_MEMO_IND:
        KLeePipe( pipeLecGED, &mensaje23, LONG_MENSAJE23, KWAIT) ;
        KLiberaMemoria( (UWORD *)mensaje23.datos);
        break;
    } /*switch(signal)*/
  } /*while(KEsperaContador(timerGED_TR) != KNO_ERROR)*/
  if (respsPruebaIdentidad == 0)
  {
    /*-------------------------------------------------------------------
      Sí no se recibe ninguna respuesta de prueba de identidad dentro del
      periodo T201, se repetirá una vez la petición de prueba de identidad
      y se rearrancará el temporizador T201:
    -------------------------------------------------------------------*/
```

```
      FormarTramaGED_UI( (UBYTE **)trama, &longTrama, &numRef,
        PETICION_PRUEBA_IDENTIDAD, IET_DE_GRUPO);
      EnviarGEDunidadDatosReq();
      KActivaContador( timerGED_TR, T201);
    }
  } /*for(n204ByMe=1 ;n204ByMe<N204_BY_ME; n204ByMe++)*/
  /*-------------------------------------------------------------------------
     Al terminar el (los) proceso de temporización T201 se comprueba:
       - Se responde a las peticiones de identidad pendientes
       - Sí no se puede asignar más IETs, se envía una petición de prueba de
         identidad con IET_DE_GRUPO y se liberan los buffer de peticiones y
         verificaciones pendientes
  -------------------------------------------------------------------------*/
  while( frenteBpeticionesId != finalBpeticionesId)
  {
    if ((newIET = GetNewIET()) != IET_DE_GRUPO)
    {
      numRef = RecuperaBpeticionesId();
      FormarTramaGED_UI( (UBYTE **)trama, &longTrama, &numRef,
        IDENTIDAD_ASIGNADA, newIET);
      EnviarGEDunidadDatosReq();
      FormarMensaje23( &mensaje23, GED_ASIGNACION_IET_REQ,
        newIET, NULL);
      KEscribePipe( pipeEscGED, (void *)&mensaje23, LONG_MENSAJE23,
        KWAIT);
      KSignal( idProcesoN2, GED_ASIGNACION_IET_REQ);
    }
    else
    {
      /* Sí no quedan más IETs libres, se llevará a cabo una Petición
         de Prueba de Identidad con IET_DE_GRUPO: */
      FormarTramaGED_UI( (UBYTE **)trama, &longTrama, &numRef,
        PETICION_PRUEBA_IDENTIDAD , IET_DE_GRUPO);
      EnviarGEDunidadDatosReq();
      KActivaContador( timerGED_TR, T201);
      /* Liberamos los dos buffers de peticiones y verificaciones: */
      frenteBpeticionesId     = finalBpeticionesId     = 0;
      frenteBverificacionesId = finalBverificacionesId = 0;
      for(i=0; i<MAX_IETS_ASIGNADOS; i++)
        vecIETsAsignados[i] = NO_ASIGNADO;
      return(ESPERA_RESP_PRUEBA_ID_IET_DE_GRUPO);
    }
  } /*while( frenteBpeticionesId != finalBpeticiones)*/
  /*-------------------------------------------------------------------------
     Sí existe alguna(-s) petición de verificación pendiente, pasamos al
     estado EsperaRespPruebaIdIETconcreto, sino, pasamos al estado Idle:
  -------------------------------------------------------------------------*/
  if (frenteBverificacionesId != finalBverificacionesId)
    return(ESPERA_RESP_PRUEBA_ID_IET_CONCRETO);
  else
    return(IDLE);
} /*EsperaRespPruebaIdIETdeGrupo*/


UWORD EsperaRespPruebaIdIETconcreto( void)
{
  //UWORD tramaIdentificada;
  UWORD numRef;
  UBYTE tipoMensaje;
  UBYTE indicadorAccion;
  UBYTE newIET;
  UBYTE numIETenPrueba;
  /* Número de respuestas al mensaje de prueba de identidad del IET concreto*/
  UWORD respsPruebaIdentidad;
  UWORD pruebaIdFinalizada;
  UWORD n204ByMe;
  UWORD i;

  /* Inicialización variables auxiliares: */
```

```
respsPruebaIdentidad = 0;
pruebaIdFinalizada = FALSE;

do{
  numIETenPrueba = RecuperaBverificacionesId();
  /*-------------------------------------------------------------------------
     Sí hay algún mensaje solicitud de verificación de identidad de un IET
     concreto en el bufferVerificacionesId, el GED del TR enviará un mensaje
     de PETICION_PRUEBA_IDENTIDAD al lado del Usuario esperando recibir un
     mensaje de RESPUESTA_IDENTIDAD:
       - Se activa el temporizador T201 sg.
       - Se enviará un segundo mensaje de PETICION_PRUEBA_IDENTIDAD en el
         caso que no hubiese contestación al primero después del periodo
         T201
  -------------------------------------------------------------------------*/
  FormarTramaGED_UI( (UBYTE **)trama, &longTrama, &numRef,
    PETICION_PRUEBA_IDENTIDAD, numIETenPrueba);
  EnviarGEDunidadDatosReq();
  KActivaContador( timerGED_TR, T201);
  for(n204ByMe=1, respsPruebaIdentidad=0; n204ByMe<N204_BY_ME; n204ByMe++)
  {
    while(KEsperaContador(timerGED_TR) != KNO_ERROR)
    {
      KLeeSignal(&signal);
      switch(signal)
      {
        case GED_UNIDAD_DATOS_IND:
          KLeePipe( pipeLecGED, &mensaje23, LONG_MENSAJE23, KWAIT) ;
          tramaIdentificada = IdentificaTramaGED_UI( mensaje23.datos,
            mensaje23.longDatos, &numRef, &tipoMensaje, &indicadorAccion);
          EnviarGEDliberacionMemoReq();
          if (tramaIdentificada == TRUE)
            switch(tipoMensaje)
            {
              case RESPUESTA_PRUEBA_IDENTIDAD:
                if (indicadorAccion == numIETenPrueba)
                  respsPruebaIdentidad++;
                break;
              case PETICION_IDENTIDAD:
                if ((newIET = GetNewIET()) != IET_DE_GRUPO)
                {
                  FormarTramaGED_UI( (UBYTE **)trama, &longTrama, &numRef,
                    IDENTIDAD_ASIGNADA, newIET);
                  EnviarGEDunidadDatosReq();
                  FormarMensaje23( &mensaje23, GED_ASIGNACION_IET_REQ,
                    newIET, NULL);
                  KEscribePipe( pipeEscGED, (void *)&mensaje23,
                    LONG_MENSAJE23, KWAIT);
                  KSignal( idProcesoN2, GED_ASIGNACION_IET_REQ);
                }
                else
                  GuardaBpeticionesId( numRef);
                break;
              case VERIFICACION_IDENTIDAD:
                GuardaBverificacionesId( indicadorAccion);
                break;
              default:
                break;
            } /*switch(tipoMensaje)*/
          break;
        case GED_LIBERACION_MEMO_IND:
          KLeePipe( pipeLecGED, &mensaje23, LONG_MENSAJE23, KWAIT) ;
          KLiberaMemoria( (UWORD *)mensaje23.datos);
          break;
        default:
          break;
      } /*switch(signal)*/
    } /*while(KEsperaContador(timerGED_TR) != KNO_ERROR)*/
    if (respsPruebaIdentidad == 1)
    {
```

```
        /*-------------------------------------------------------------------
          Cuando el procedimiento de prueba de IET se emplea para determinar
          sí se utiliza un valor IET concreto, este procedimiento se
          considera completado al recibirse el primer mensaje de respuesta
          de prueba de identidad IET, y se supone que el valor IET está
          siendo utilizado:
        -------------------------------------------------------------------*/
        n204ByMe = N204_BY_ME;  /* Para salir del bucle for */
        pruebaIdFinalizada = TRUE;
      }
      if (respsPruebaIdentidad > 1)
      {
        /*-------------------------------------------------------------------
          Sí se recibe más de una respuesta de prueba de identidad, se
          considera que existe múltiple asiganción de IET:
            Se enviará mensaje de Supresión de Identidad 2 veces seguidas
        -------------------------------------------------------------------*/
        KSignal( idProcesoN2, GED_SUPRESION_IET_REQ);*/
        vecIETsAsignados[numIETenPrueba - DIFERENCIA_RANGO] = NO_ASIGNADO;
        FormarTramaGED_UI( (UBYTE **)trama, &longTrama, &numRef,
          SUPRESION_IDENTIDAD, indicadorAccion );
        EnviarGEDunidadDatosReq();
        EnviarGEDunidadDatosReq();
        n204ByMe = N204_BY_ME;  /* Para salir del bucle for */
        pruebaIdFinalizada = TRUE;
        //respsPruebaIdentidad = 1;
        /* Para no ejecutar el código que dice respsPruebaIdentidad > 1 */
      }
      if (respsPruebaIdentidad == 0)
      {
        /*-------------------------------------------------------------------
          Sí no se recibe ninguna respuesta de prueba de identidad dentro del
          periodo T201, se repetirá una vez la petición de prueba de identidad
          y se rearrancará el temporizador T201:
        -------------------------------------------------------------------*/
        FormarTramaGED_UI( (UBYTE **)trama, &longTrama, &numRef,
          PETICION_PRUEBA_IDENTIDAD, numIETenPrueba);
        EnviarGEDunidadDatosReq();
        KActivaContador( timerGED_TR, T201);
      }
    } /*for(n204ByMe=1; n204ByMe<N204_BY_ME; n204ByMe++)*/
    if (pruebaIdFinalizada == FALSE)
    {
      if (respsPruebaIdentidad == 0)
        /*-------------------------------------------------------------------
          Sí no se recibe ninguna respuesta de prueba de identidad después de
          la segunda petición de prueba de identidad, se supondrá que el valor
          IET está libre y disponible para reasignación:
        -------------------------------------------------------------------*/
        vecIETsAsignados[numIETenPrueba - DIFERENCIA_RANGO] = NO_ASIGNADO;
      if (respsPruebaIdentidad > 1)
      {
        /*-------------------------------------------------------------------
          Sí se recibe más de una respuesta de prueba de identidad, se
          considera que existe múltiple asiganción de IET
        -------------------------------------------------------------------*/
        vecIETsAsignados[numIETenPrueba - DIFERENCIA_RANGO] = NO_ASIGNADO;
        FormarTramaGED_UI( (UBYTE **)trama, &longTrama, &numRef,
          SUPRESION_IDENTIDAD, indicadorAccion );
        /* Se envía el mensaje de Supresión de Identidad dos veces
           consecutivas: */
        EnviarGEDunidadDatosReq();
        EnviarGEDunidadDatosReq();
      }
    } /*if (pruebaIdFinalizada == FALSE)*/
  }while(frenteBverificacionesId != finalBverificacionesId);
  if (frenteBpeticionesId == finalBpeticionesId)
    return(IDLE);
  else
  {
```

```
    /*-------------------------------------------------------------------------
       En este caso el buffer de peticiones de Identidad no está vacío y, por
       tanto, se llevará a cabo una Petición de Prueba de Identidad con
       IET_DE_GRUPO:
    -------------------------------------------------------------------------*/
    FormarTramaGED_UI( (UBYTE **)trama, &longTrama, &numRef,
      PETICION_PRUEBA_IDENTIDAD, IET_DE_GRUPO);
    EnviarGEDunidadDatosReq();
    KActivaContador( timerGED_TR, T201);
    for(i=0; i<MAX_IETS_ASIGNADOS; i++)
      vecIETsAsignados[i] = NO_ASIGNADO;
    return(ESPERA_RESP_PRUEBA_ID_IET_DE_GRUPO);
  }
} /*EsperaRespPruebaIdIETconcreto*/


/*------------------- Fin definiciones de funciones ------------------------*/
```

```
/*************************************************************************/
/*                              MUX.C                                    */
/*        TRABAJO FIN DE CARRERA  SANTIAGO MARTIN ROMANI                 */
/*    Fichero implementación proceso multiplexor del lado del            */
/*    Terminal de Red: Este proceso se comunica con los procesos         */
/*    que representan entidades de nivel 2 del lado del Terminal de Red  */
/*************************************************************************/


#define UNIX

/*------------ FICHEROS INCLUIDOS: ------------------------------------------*/
#include "basico.h"    /* ..\kernel\basico.h */
#include "kernel.h"    /* ..\kernel\kernel.h */
#include "inicio.h"
#include "nivel1.h"
#include "nivel2.hp"
#include "n2n3.h"
#include "ged.hp"
#include "mux.hp"
#include "mux_tr.h"


/*------------ DECLARACION DE VARIABLES GLOBALES: --------------------------*/
UBYTE idProcesoMux;
UBYTE idSCC;

/*-------------------------------------------------------------------------
  Estructura que define los valores de los parametros del puerto y del
  protocolo HDLC que se intercambian con el nivel físico:
-------------------------------------------------------------------------*/
PARAM_SCC parametros;

UWORD signal;
UBYTE *header;
UWORD lengHeader;
UBYTE *info;
UWORD longInfo;

/*-------------------------------------------------------------------------
   Vector cuyos elementos identifican a cada uno de los procesos de Terminal
     de Red que están activos:
   Cada uno de los elementos del vector contendrá cuatro campos:
     1.- libre
     2.- numIETasignado
     3.- idProcesoTR
     4.- pipeEscMuxTR
-------------------------------------------------------------------------*/
ELEM_VEC_MUX vecMux[LONG_VEC_MUX];


/*-------------------------------------------------------------------------
   Pipe de lectura para el Proceso Multiplexor, que es el pipe de escritura
     para los procesos de Nivel 2 del lado del Terminal de Red:
   Todos los procesos que implementan el Nivel 2 del lado de Terminal de Red
     escriben sus mensajes sobre este pipe:
-------------------------------------------------------------------------*/
UWORD pipeLecMux;

MENSAJE_MUX_N2 mensajeMuxN2;

/*-------------------------------------------------------------------------
  El buffer de transmisión del multiplexor (bufferTxMux) junto con la función
  RecuperaBtxMux son necesarios para mantener un control del orden en el que
  se liberará la memoria dinámica correspondiente a las tramas entregadas al
  nivel físico y confirmadas por este con cada signal FI_DATOS_ENVIADOS:
-------------------------------------------------------------------------*/
TRAMA_BUFFER_TX_MUX bufferTxMux[LONG_BUF_TX];
UWORD               frenteBtxMux, finalBtxMux;

/* bufferTramasAtx: */
ELEM_BUF_TRAMAS_A_TX bufferTramasAtx[LONG_BUF_TRAMAS_A_TX];
```

```
UWORD                frenteBtramasAtx, finalBtramasAtx;

/*-------------------------------------------------------------------------
  Posición relativa del elemento del vector vecMux que corresponde con
  el proceso de nivel 2 master (aquel que recibe y envía las tramas UI
  del proceso Gestor de Enlace de Datos) y recibe las tramas UI con destino
  el nivel 3:
    (Para saber a qué proceso TR hay que enviar todas las tramas UI)
-------------------------------------------------------------------------*/
UWORD indiceTRmaster;

/*-------------------------------------------------------------------------
  Posición relativa del elemento del vector vecMux que corresponde con
  el proceso de nivel 2 del lado del Terminal de Red al que hay que enviar
  una determinada trama según su IET:
-------------------------------------------------------------------------*/
UBYTE indiceTRdestino;

/*Identificador de Proceso Terminal de Red emisor de una determinada trama:*/
UBYTE idProcesoTRemisor;



/*------------ DEFINICIONES DE PROTOTIPOS DE FUNCIONES: --------------------*/
void  N2_MuxTRmain( UWORD pipeLectura, UWORD idPuerto);
void  InicializarParametros( void);

UWORD TratarFIdatosRecibidos( UBYTE indiceTRmaster);

UWORD IdentificaTramaRecibida( UBYTE *frame, UWORD frameLength,
  UBYTE *numIETrecibido);

UWORD CheckTRdisponible( UBYTE numIETrecibido, UWORD *indiceTRdestino);
UWORD CheckIETrecibido( UBYTE numIETrecibido, UWORD *indiceTRdestino);

void  FormarMensajeMuxN2( MENSAJE_MUX_N2 *messageMuxTR, UWORD tipoMessage,
  UBYTE idProcesoTR, UBYTE *header, UWORD lengHeader,
  UBYTE *info, UWORD longInfo);

void  EnviarTramaAlTRdestino( UWORD );
void  EnviarTramaUIalTRmaster( UWORD indiceTRmaster);

UWORD GuardarBtxMux( UBYTE idProcesoTRemisor);
UWORD RecuperarBtxMux( void);

UWORD GuardarBtramasAtx( UBYTE idProcesoTRemisor,
  UBYTE *header, UWORD lengHeader, UBYTE *info, UWORD longInfo);
UWORD RecuperarBtramasAtx( UBYTE *idProcesoTRemisor,
  UBYTE **header, UWORD *lengHeader, UBYTE **info, UWORD *longInfo );



/*------------ DECLARACION DE FUNCIONES: -----------------------------------*/

void N2_MuxTRmain( pipeLectura, idPuerto)
UWORD pipeLectura;
/* Para la communicacion con los procesos de nivel 2 del lado
   del terminal de red */
UWORD idPuerto;
/* Puerto físico que controla */
{
  /*-----------------------------------------------------------------------
    Esta función es la implementación del proceso multiplexor o concentrador
      del lado del Terminal de Red, que es un proceso intermedio entre los
      procesos de Capa de Enlace de Datos y el nivel físico.
    Este proceso es necesario porque el nivel físico por si solo, no está
      preparado para enviar las distintas tramas que recibe a los distintos
      procesos de nivel 2 que corresponda en cada caso
  -----------------------------------------------------------------------*/
  UWORD i;
```

```
UBYTE numIETrecibido;
/* Tipo de trama que se ha recibido o que se transmitirá: */
UWORD tipoTrama;

/* Tipo de mensaje intercambiado entre el proceso multiplexor y los procesos
   que representan el nivel 2 del lado del Terminal de Red: */
UWORD tipoMensaje;

/*-------------------------------------------------------------------------
  Parámetros de Entrada al Proceso que representa al multiplexor auxiliar
  que se comunica con las entidades de nivel 2 del lado del Terminal de Red
-------------------------------------------------------------------------*/
pipeLecMux = pipeLectura;
idSCC      = idPuerto;

idProcesoMux = KIDProceso();

InicializarParametros();

while(TRUE)  /*El proceso vive para siempre*/
{
  KWait();
  KLeeSignal( &signal);
  switch(signal)
  {
    case FI_ERROR_IND:
    case FI_DATOS_RECIBIDOS:
      TratarFIdatosRecibidos( indiceTRmaster);
      break;
    case FI_DATOS_ENVIADOS:
      /* Para indicar al proceso de Terminal de Red correspondiente que el
         nivel físico envío ya la 1ª trama que tenía pendiente: */
      RecuperarBtxMux();
      if (RecuperarBtramasAtx( &idProcesoTRemisor, &header,
        &lengHeader, &info, &longInfo) == TRUE)
        if (FEnviaDatos( idSCC, (void *)header, lengHeader, (void *)info,
          longInfo) == TRUE)
          GuardarBtxMux( idProcesoTRemisor);
        else
          GuardarBtramasAtx( idProcesoTRemisor, header, lengHeader,
            info, longInfo);
      break;
    case MUX_TR_DATOS_REQ:
      KLeePipe( pipeLecMux, &mensajeMuxN2, LONG_MENSAJE_MUX_N2, KWAIT);
      idProcesoTRemisor = mensajeMuxN2.idProcesoTR;
      switch( mensajeMuxN2.tipoMensaje)
      {
        case MUX_TR_ID_PROCESO_REQ:
          /*-----------------------------------------------------------------
             Tratamiento primer mensaje procedente de las entidades-procesos
             de Capa de Enlace de Datos
          -----------------------------------------------------------------*/
          for( i=0; i<LONG_VEC_MUX; i++)
            if (vecMux[i].libre == TRUE)
            {
              vecMux[i].libre        = FALSE;
              vecMux[i].idProcesoTR  = mensajeMuxN2.idProcesoTR;
              vecMux[i].pipeEscMuxTR = mensajeMuxN2.lengHeader;
              if (mensajeMuxN2.longInfo == PROCESO_MASTER)
                indiceTRmaster = i;
              break;
            }
          break;
        case MUX_TR_ENLACE_LIBERADO_REQ:
          for( i=0; i<LONG_VEC_MUX; i++)
            if (vecMux[i].idProcesoTR == mensajeMuxN2.idProcesoTR)
            {
              vecMux[i].numIETasignado = IET_DE_GRUPO;
              break;
            }
          break;
        case MUX_TR_DATOS_REQ:
```

```
            if (frenteBtramasAtx == finalBtramasAtx)
              /* si no hay ninguna trama esperando para ser transmitida, se
                 intentará transmitir: */
              if (FEnviaDatos(idSCC, (void *)mensajeMuxN2.header,
                mensajeMuxN2.lengHeader, (void *)mensajeMuxN2.info,
                mensajeMuxN2.longInfo) == TRUE)
              {
                GuardarBtxMux( idProcesoTRemisor);
                break;
              }
            GuardarBtramasAtx( idProcesoTRemisor, header, lengHeader,
              info, longInfo);
            break;
          } /*switch(mensajeMuxN2.tipoMensaje)*/
          break;
        default: /*signal*/
          break;
      } /*switch(signal)*/
    } /*while(TRUE)*/
} /*MuxMain*/


void InicializarParametros( void)
{
  /* Inicializa variables globales que se utilizarán en el proceso de
     comunicación */
  UWORD i;
  /*--------- Inicializar el puerto fisico: ---------------------------------*/
  parametros.hdlc.IDProceso  = idProcesoMux;
  parametros.hdlc.signal[0]  = FI_DATOS_ENVIADOS;
  parametros.hdlc.signal[1]  = FI_DATOS_RECIBIDOS;
  parametros.hdlc.signal[2]  = FI_ERROR_IND;
  parametros.hdlc.caracRecep = LONG_MAX_TRAMA_I;
  FIniciaSCC( idSCC, (UWORD)(PROTOCOLO_HDLC), &parametros);

  /*-----------------------------------------------------------------------
     Inicialización elementos del vector del multiplexor (vecMux) que contiene
     información sobre el estado de cada numIET utilizado y su correspondencia
     con un determinado proceso de TR y un pipe de escritura para el proceso
     multiplexor:
  -----------------------------------------------------------------------*/
  for( i=0; i<LONG_VEC_MUX; i++)
  {
    vecMux[i].libre          = TRUE;
    vecMux[i].numIETasignado = IET_DE_GRUPO;
  }
  /*-------- Inicialización índices buffers utilizados: --------------------*/
  frenteBtxMux     = finalBtxMux     = 0;
  frenteBtramasAtx = finalBtramasAtx = 0;
}


UWORD TratarFIdatosRecibidos( indiceTRmaster)
UBYTE indiceTRmaster;
{
  /*-----------------------------------------------------------------------
    Se ejecutará esta función cada vez que se reciba una señal
    FI_DATOS_RECIBIDOS o FI_ERROR_IND que indica que se ha recibido una trama
    del nivel físico. Se procesa la trama sino hubo error al recibirla y según
    su tipo, se enviará al proceso adecuado:
      Al proceso N2 master si se trata de una trama de información sin
        confirmar o una trama dirigida al Gestor de Enlace de Datos
      O al proceso de Nivel 2 que corresponda con el IET utilizado en el campo
        de dirección de esa trama recibida
      Si la trama recibida es una trama SABME, se envía al proceso con IET
        asociado igual al IET recibido o a un proceso de nivel 2 libre.
      Si no hay ningún proceso libre, se descarta
  -----------------------------------------------------------------------*/
  UWORD i;
  UWORD error;  /* Error devuelto por el nivel físico */
  UWORD tipoTrama;
  UBYTE numIETrecibido;
```

```
UWORD indiceTRdestino;
if (FRecibeDatos( idSCC, (void **)&header, &lengHeader, &error)
  == TRUE)
{
  /*-------------------------------------------------------------------------
    si se ha producido algún error en la recepción de la trama, ésta
    se descarta. Los errores que el nivel físico puede detectar e
    indicar al Proceso Multiplexor del lado del Terminal de Red son:
      ERROR_OVERRUN
      ERROR_FRAMING
      ERROR_PARITY
      ERROR_BREAK
      ERROR_ABORT
      ERROR_NONOCTET
      ERROR_CRC
      ERORR_DESACT
  -------------------------------------------------------------------------*/
  if (error != 0)
  {
    if (header != NULL)
      /*-----------------------------------------------------------------------
        si se recibió una trama errónea (es decir, si el puntero está
        apuntando a alguna posición de memoria), será el multiplexor
        el encargado de liberarar la memoria que ocupa:
      -----------------------------------------------------------------------*/
      KLiberaMemoria( (UWORD *)header);
    if (error & ERROR_DESACT)
    {
      /*-----------------------------------------------------------------------
        si se produce un error de desactivación del nivel físico, se
        indicará a todos los procesos de nivel 2 del lado del Terminal
        de Red (que estén comunicándose con su respectivo remoto), que
        se ha caído el "enlace de nivel físico" y por tanto deberán
        pasar al estado "EnlaceLiberado":
      -----------------------------------------------------------------------*/
      for( i=0; i<LONG_VEC_MUX; i++)
      if ((vecMux[i].numIETasignado != IET_DE_GRUPO) &&
        (vecMux[i].libre == FALSE))
      {
        KSignal( vecMux[i].idProcesoTR, MUX_TR_ENLACE_LIBERADO_IND);
        vecMux[i].libre          = TRUE;
        vecMux[i].numIETasignado = IET_DE_GRUPO;
      }
      return(TRUE);
    } /*if (error & ERROR_DESACT)*/
  }
  else  /*if (error != 0)*/
  {
    /* si no hubo error al recibir la trama, se procesa: */
    tipoTrama = IdentificaTramaRecibida( header, lengHeader,
      &numIETrecibido);
    switch(tipoTrama)
    {
      /*-------------------------------------------------------------------------
        Si la trama es tipo GED_UI o TR_UI se envía al proceso TR master
          (proceso de nivel 2 del lado del Terminal de Red master)
        Si la trama es tipo SABME se enviará al proceso con el IET
          recibido o a un proceso TR libre. Si no hay ningún proceso
          libre, se descarta.
        El resto de tramas del protocolo LAP-D que se reciban
          se envían al proceso TR correspondiente con el IET recibido
      -------------------------------------------------------------------------*/
      case SABME:
        if(CheckTRdisponible( numIETrecibido, &indiceTRdestino)
          == DISPONIBLE)
          EnviarTramaAlTRdestino( indiceTRdestino);
        else
          KLiberaMemoria( (UWORD*)header);
        return(TRUE);
      case TRAMA_GED_UI:
      case TRAMA_TR_UI:
        EnviarTramaUIalTRmaster( indiceTRmaster);
```

```
          return(TRUE);
        case TRAMA_TR:
          if (CheckIETrecibido( numIETrecibido, &indiceTRdestino)
            == INCLUIDO)
            EnviarTramaAlTRdestino( indiceTRdestino);
          else
            KLiberaMemoria( (UWORD*)header);
          return(TRUE);
        default:
          KLiberaMemoria( (UWORD*)header);
      } /*switch(tipoTrama)*/
    } /*else if (error != 0)*/
  } /*if(FRecibeDatos(..) == TRUE*/
  return(TRUE);
} /*TratarFIdatosRecibidos*/


void EnviarTramaAlTRdestino( indiceTRdestino)
UWORD indiceTRdestino;
{
  /*-------------------------------------------------------------------------
    Se envía un mensaje (mensajeMuxN2) al proceso-entidad de Capa de Enlace de
    Datos del lado del Terminal de Red (a el indicado en la variable
    indiceTRdestino) y se le indica con el signal MUX_TR_DATOS_IND que
    equivale al signal FI_DATOS_ENVIADOS o FI_ERROR_IND
  -------------------------------------------------------------------------*/
  FormarMensajeMuxN2( &mensajeMuxN2, MUX_TR_DATOS_IND,
    vecMux[indiceTRdestino].idProcesoTR, header, lengHeader,
    NULL, LONG_CERO);
  KEscribePipe( vecMux[indiceTRdestino].pipeEscMuxTR,
    (void *)&mensajeMuxN2, LONG_MENSAJE_MUX_N2, KWAIT);
  KSignal( vecMux[indiceTRdestino].idProcesoTR, MUX_TR_DATOS_IND);
}

void EnviarTramaUIalTRmaster( indiceTRmaster)
UWORD indiceTRmaster;
{
  /*-------------------------------------------------------------------------
    Se envía un mensaje (mensajeMuxN2) al proceso-entidad master conteniendo
    la trama UI recibida del nivel físico y que se encuentra en las variables
    globales header y lengHeader (trama recibida y su longitud)
  -------------------------------------------------------------------------*/
  FormarMensajeMuxN2( &mensajeMuxN2, MUX_TR_DATOS_IND,
    vecMux[indiceTRmaster].idProcesoTR, header, lengHeader,
    NULL, LONG_CERO);
  KEscribePipe( vecMux[indiceTRmaster].pipeEscMuxTR,
    (void *)&mensajeMuxN2, LONG_MENSAJE_MUX_N2, KWAIT);
  KSignal( vecMux[indiceTRmaster].idProcesoTR, MUX_TR_DATOS_IND);
}

void  FormarMensajeMuxN2( messageMuxN2, tipoMessage, idProcesoTR,
  header, lengHeader, info, longInfo)
MENSAJE_MUX_N2 *messageMuxN2;
UWORD tipoMessage;
UBYTE idProcesoTR;
UBYTE *header, *info;
UWORD lengHeader, longInfo;
{
  /*-------------------------------------------------------------------------
    Esta función sirve para formar un mensaje (mensajeMuxN2) que luego será
    enviado a la entidad de Capa de Enlace de Datos correspondiente
  -------------------------------------------------------------------------*/
  messageMuxN2->tipoMensaje = tipoMessage;
  messageMuxN2->idProcesoTR = idProcesoTR;
  messageMuxN2->header      = header;
  messageMuxN2->lengHeader  = lengHeader;
  messageMuxN2->info        = info;
  messageMuxN2->longInfo    = longInfo;
}


UWORD IdentificaTramaRecibida( frame, frameLength, numIETrecibido)
```

```
UBYTE *frame;
UWORD frameLength;
UBYTE *numIETrecibido;
{
  /*-------------------------------------------------------------------------
    Identifica el tipo de trama recibida del nivel físico y extrae el valor
    del IET del campo de dirección de la trama recibida
  -------------------------------------------------------------------------*/
  UBYTE numIPASrecibido;
  if ((GET_BIT_ED(frame[0]) != 0) || (GET_BIT_ED(frame[1]) != 1))
    return(DESCONOCIDO);
  numIPASrecibido  = GET_IPAS(frame[0]);

  printf("\n frame[1] = %x", frame[1]);

  *numIETrecibido  = GET_IET(frame[1]);

  printf("\n numIETrecibido = %x", *numIETrecibido);

  if (numIPASrecibido == IPAS_CERO)
  {
    if (*numIETrecibido == IET_DE_GRUPO)
      return(TRAMA_TR_UI);
    if (frameLength == SHORT_HEADER)
      if ((GET_CTRL_TRAMAS_U(frame[2]) == SABME_MASK) &&
        (GET_BIT_PF_TRAMAS_U(frame[2]) == ACTIVO))
        return(SABME);
    return(TRAMA_TR);
  }
  if ((numIPASrecibido == IPAS_GED) && (*numIETrecibido == IET_GED))
    return(TRAMA_GED_UI);
  /* En cualquier otro caso, la trama es desconocida: */
  return(DESCONOCIDO);
} /*IdentificaTramaRecibida*/


UWORD CheckTRdisponible( numIETrecibido, indiceTRdestino)
UBYTE numIETrecibido;
UWORD *indiceTRdestino;
{
  UWORD i;
  /*-------------------------------------------------------------------------
    Cuando se recibe una trama SABME se ejecutará esta función:
      1.- Si existe un proceso de nivel 2 del lado TR que está comunicándose
          con un proceso de nivel 2 del lado ET con un numIETasignado igual
          al IET recibido en la trama, la enviamos a ese proceso
      2.- Si no se cumple lo anterior, pero existe un proceso del lado del TR
          que no está comunicándose con ningún proceso del lado del ET, se
          asignará a la nueva comunicación el IET recibido
      3.- Si no se cumple ninguna de las dos anteriores posibilidades, la
          función devuelve el valor NO_DISPONIBLE pués no existe ningún
          proceso comunicándose "a través" de ese IET ni tampoco se le puede
          asignar a otro pués están todos temporalmente ocupados.

      NOTA: La necesidad de dimensionar el vector vecMux a LONG_VEC_MUX = 12,
            se debe a la posibilidad de caída de un ET sin notificación al TR;
            en ese caso puede pasar (T203 + (3 * T200))sg antes de que
            el lado TR decida que la comunicación que emplea ese determinado
            IET ha dejado de existir y permitir que ese proceso TR pueda
            comunicarse utilizando otro IET.
              Para que una posible comunicación entre un TR y un ET pueda
            emplear un nuevo IET cuando por ejemplo, se enciende y apaga un
            ET, es necesario dimensionar el vecMux a un valor mayor que 8.
              Por otra parte podría ser que un determinado ET utilizase más de
            un IET comunicándose con varios procesos TR (cada uno con un IET
            diferente):
  -------------------------------------------------------------------------*/
  for( i=0; i<LONG_VEC_MUX; i++)
    if (vecMux[i].numIETasignado == numIETrecibido)
    {
      *indiceTRdestino = i;
      return(DISPONIBLE);
```

```
    }
  for( i=0; i<LONG_VEC_MUX; i++)
  {
    if ((vecMux[i].libre == FALSE) &&
      (vecMux[i].numIETasignado == IET_DE_GRUPO))
    {
      vecMux[i].numIETasignado = numIETrecibido;
      *indiceTRdestino = i;
      return(DISPONIBLE);
    }
  }
  *indiceTRdestino = NO_DISPONIBLE;
  return(NO_DISPONIBLE);
}


UWORD CheckIETrecibido( numIETrecibido, indiceTRdestino)
UBYTE numIETrecibido;
UWORD *indiceTRdestino;
{
  /*-------------------------------------------------------------------------
    Comprueba si alguno de los procesos de nivel 2 Terminal de Red, está
    comuncicándose con un proceso de nivel 2 remoto (Equipo Terminal)
    utilizando el IET recibido. Si es así, devuelve el índice de ese TR
    en la variable indiceTRdestino y la función toma el valor INCLUIDO.
    En caso contrario la función toma el valor NO_INCLUIDO.
  -------------------------------------------------------------------------*/
  UWORD i;
  for (i=0; i<LONG_VEC_MUX; i++)
    if (vecMux[i].numIETasignado == numIETrecibido)
    {
      *indiceTRdestino = i;
      return(INCLUIDO);
    }
  return(NO_INCLUIDO);
}


UWORD GuardarBtxMux( idProcesoTRemisor)
UWORD idProcesoTRemisor;
{
  /*-------------------------------------------------------------------------
     Inserta un elemento (idProcesoTRemisor) en la cola bufferTx
     y devuelve TRUE. Si la cola está llena, devuelve FALSE
  -------------------------------------------------------------------------*/
  /* Comprobamos si cola llena: */
  if ((frenteBtxMux == finalBtxMux+1) ||
    ((frenteBtxMux == 0) && (finalBtxMux == LONG_BUF_TX_MUX-1)))
    return(FALSE);
  else
  {
    bufferTxMux[finalBtxMux].idProcesoTRemisor = idProcesoTRemisor;
    if ((++finalBtxMux) == LONG_BUF_TX_MUX)
      finalBtxMux = 0;
    return(TRUE);
  }
} /*GuardaBtxMux*/


UWORD RecuperarBtxMux( void)
{
  /*-------------------------------------------------------------------------
     Por cada signal recibido FI_DATOS_ENVIADOS, se ejecutará esta función:
     Se envía al proceso TR emisor la señal MUX_TR_DATOS_ENVIADOS_IND para
     indicar que el nivel físico ya envío la trama correspondiente
  -------------------------------------------------------------------------*/
  /* Comprobación cola vacía: */
  if (frenteBtxMux == finalBtxMux)
    return(FALSE);
  else
  {
    KSignal( bufferTxMux[frenteBtxMux].idProcesoTRemisor,
```

```
      MUX_TR_DATOS_ENVIADOS_IND);
    if ((++frenteBtxMux) == LONG_BUF_TX_MUX)
      frenteBtxMux = 0;
    return(TRUE);
  }
} /*RecuperarBtxMux*/


UWORD GuardarBtramasAtx( idProcesoTRemisor, header, lengHeader,
  info, longInfo)
UBYTE idProcesoTRemisor;
UBYTE *header, *info;
UWORD lengHeader, longInfo;
{
  /*-------------------------------------------------------------------------
    Inserta un elemento en el buffer de tramas a transmitir:
      (identificador del proceso TR emisor,cabecero y su longitud,
      información y su longitud. Si la cola está llena, devuelve FALSE
  -------------------------------------------------------------------------*/
  /* Comprobamos si cola llena: */
  if ((frenteBtramasAtx == finalBtramasAtx+1) ||
    ((frenteBtramasAtx == 0) && (finalBtramasAtx == LONG_BUF_TRAMAS_A_TX-1)))
    return(FALSE);
  else
  {
    bufferTramasAtx[finalBtramasAtx].idProcesoTRemisor = idProcesoTRemisor;
    bufferTramasAtx[finalBtramasAtx].header            = header;
    bufferTramasAtx[finalBtramasAtx].lengHeader        = lengHeader;
    bufferTramasAtx[finalBtramasAtx].info              = info;
    bufferTramasAtx[finalBtramasAtx].longInfo          = longInfo;
    if ((++finalBtramasAtx) == LONG_BUF_TRAMAS_A_TX)
      finalBtramasAtx = 0;
    return(TRUE);
  }
} /*GuardarBtramasAtx*/


UWORD RecuperarBtramasAtx( idProcesoTRemisor, header, lengHeader,
  info, longInfo)
UBYTE *idProcesoTRemisor;
UBYTE **header, **info;
UWORD *lengHeader, *longInfo;
{
  /*-------------------------------------------------------------------------
    Cuando se ha recibido un FI_DATOS_ENVIADOS y se puede transmitir una trama
    en cola de espera, se recupera esa trama (cabecero e información con sus
    respectivas longitudes y el identificador de proceso TR que quiere
    enviarla. Si se logra enviar al nivel físico, se guardará el identificador
    de proceso, para que al recibir la señal FI_DATOS_ENVIADOS correspondiente
    a esa trama, se lo indiquemos (MUX_TR_DATOS_ENVIADOS_IND a ese proceso TR.
  -------------------------------------------------------------------------*/
  /* Comprobación cola vacía: */
  if (frenteBtramasAtx == finalBtramasAtx)
    return(FALSE);
  else
  {
    *idProcesoTRemisor = bufferTramasAtx[frenteBtramasAtx].idProcesoTRemisor;
    *header            = bufferTramasAtx[frenteBtramasAtx].header;
    *lengHeader        = bufferTramasAtx[frenteBtramasAtx].lengHeader;
    *info              = bufferTramasAtx[frenteBtramasAtx].info;
    *longInfo          = bufferTramasAtx[frenteBtramasAtx].longInfo;
    if ((++frenteBtramasAtx) == LONG_BUF_TRAMAS_A_TX)
      frenteBtramasAtx = 0;
    return(TRUE);
  }
} /*RecuperarBtramasAtx*/

/*------------------- Fin definiciones de funciones ------------------------*/
```

```
/*****************************************************************************/
/* Con estas funciones se simulan los servicios ofrecidos por el módulo      */
/* Kernel y por el nivel físico para poder ejecutar uno a uno los programas */
/* creados                                                                   */
/*                                                                           */
/* Estas funciones se compilarán junto con el código del fichero a partir   */
/* del cual se simulará un proceso                                           */
/* Cada proceso (o programa asociado) se podrá simular individualmente       */
/*****************************************************************************/



/*********** FUNCIONES DEL KERNEL PARA SIMULACION BAJO MS-DOS: **************/

UWORD KLeeSignal( signal)
UWORD *signal;
{
  /*-------------------------------------------------------------------------
    Para simular la lectura de un signal del pipe de signals recibidos.
    Se pide la selección del signal (por teclado) empleando la función
    auxiliar "EligeSignal"
  -------------------------------------------------------------------------*/
  UWORD c;
  printf("\n\n\nUWORD KLeeSignal(UWORD *signal)");
  printf("\n            -OUT- Parameters :");
  printf("\n               [Dir]<signal> : [%p]<%%x>...",signal);
  EligeSignal( &(*signal));
  printf(" -RETURN-            <UWORD> : KNO_ERROR");
  Espera();
  return( KNO_ERROR );
} /*KLeeSignal*/


UWORD KSignal( proceso, signal)
UBYTE proceso;
UWORD signal;
{
  /*-------------------------------------------------------------------------
     Muestra el número de proceso (identificador de proceso) al que va
     dirigida el signal y el valor hexadecimal del signal y su equivalente en
     letras definido con el "define":
  -------------------------------------------------------------------------*/
  printf("\n\nUWORD KSignal(UBYTE proceso, UWORD signal)");
  printf("\n             -IN- Parameters :");
  printf("\n             <UBYTE:proceso> : %d", proceso);
  printf("\n              <UWORD:signal> : %04xx = ", signal );
  DisplaySignal( signal);
  printf("\n -RETURN-           <UWORD> : KNO_ERROR");
  Espera();
  return( KNO_ERROR );
} /*KSignal*/


void DisplaySignal( signal)
UWORD signal;
{
  /* Muestra el tipo de signal por pantalla: */
  switch(signal)
  {
    case ED_ESTABLEC_REQ:          printf("ED_ESTABLEC_REQ");          break;
    case ED_ESTABLEC_IND:          printf("ED_ESTABLEC_IND");          break;
    case ED_ESTABLEC_CONF:         printf("ED_ESTABLEC_CONF");         break;
    case ED_LIBERACION_REQ:        printf("ED_LIBERACION_REQ");        break;
    case ED_LIBERACION_IND:        printf("ED_LIBERACION_IND");        break;
    case ED_LIBERACION_CONF:       printf("ED_LIBERACION_CONF");       break;
    case ED_DATOS_REQ:             printf("ED_DATOS_REQ");             break;
    case ED_DATOS_IND:             printf("ED_DATOS_IND");             break;
    case ED_UNIDAD_DATOS_REQ:      printf("ED_UNIDAD_DATOS_REQ");      break;
    case ED_UNIDAD_DATOS_IND:      printf("ED_UNIDAD_DATOS_IND");      break;
    case ED_LIBERACION_MEMO_REQ:   printf("ED_LIBERACION_MEMO_REQ");   break;
    case ED_LIBERACION_MEMO_IND:   printf("ED_LIBERACION_MEMO_REQ");   break;
```

```
    case GED_LIBERACION_MEMO_REQ:   printf("GED_LIBERACION_MEMO_REQ");  break;
    case GED_LIBERACION_MEMO_IND:   printf("GED_LIBERACION_MEMO_IND");  break;
    case GED_ASIGNACION_IET_IND:    printf("GED_ASIGNACION_IET_IND");   break;
    case GED_ASIGNACION_IET_REQ:    printf("GED_ASIGNACION_IET_REQ");   break;
    case GED_NO_DISPONIBLE_IET_REQ: printf("GED_NO_DISPONIBLE_IET_REQ");break;
    case GED_SUPRESION_IET_REQ:     printf("GED_SUPRESION_IET_REQ");    break;
    case GED_ERROR_IND:             printf("GED_ERROR_IND");            break;
    case GED_UNIDAD_DATOS_REQ:      printf("GED_UNIDAD_DATOS_REQ");     break;
    case GED_UNIDAD_DATOS_IND:      printf("GED_UNIDAD_DATOS_IND");     break;

    case FI_DATOS_ENVIADOS:      printf("FI_DATOS_ENVIADOS");   break;
    case FI_DATOS_RECIBIDOS:     printf("FI_DATOS_RECIBIDOS");  break;
    case FI_ERROR_IND:           printf("FI_ERROR_IND");        break;

    case MUX_TR_DATOS_REQ:       printf("MUX_TR_DATOS_REQ"); break;
    case MUX_TR_DATOS_IND:       printf("MUX_TR_DATOS_IND"); break;
    case MUX_TR_ENLACE_LIBERADO_IND:
                                 printf("MUX_TR_ENLACE_LIBERADO_IND"); break;
    case MUX_TR_DATOS_ENVIADOS_IND:
                                 printf("MUX_TR_DATOS_ENVIADOS_IND"); break;

    default:     printf("SIGNAL NO CONTEMPLADO");  break;
  }
} /*DisplaySignal*/


void EligeSignal( signal)
UWORD *signal;
{
  /* Esta función se utiliza par simplificar la selección de un signal: */
  UWORD c;
  printf("\n\n... Tipo Signal a leer (1.-ED  2.-GED  3.-FI  4.-MUX_TR  5.-Del
teclado):  ");
  do{
    scanf("%d", &c);
  }while((c != 1) && (c != 2) && (c != 3) && (c != 4) && (c != 5));
  switch(c)
  {
    case 1:
      printf("\n    ED_ESTABLEC_REQ ........... 1");
      printf("\n    ED_ESTABLEC_IND ........... 2");
      printf("\n    ED_ESTABLEC_CONF .......... 3");
      printf("\n    ED_LIBERACION_REQ ......... 4");
      printf("\n    ED_LIBERACION_IND ......... 5");
      printf("\n    ED_LIBERACION_CONF ........ 6");
      printf("\n    ED_DATOS_REQ .............. 7");
      printf("\n    ED_DATOS_IND .............. 8");
      printf("\n    ED_UNIDAD_DATOS_REQ ....... 9");
      printf("\n    ED_UNIDAD_DATOS_IND ....... 10");
      printf("\n    ED_LIBERACION_MEMO_REQ .... 11");
      printf("\n    ED_LIBERACION_MEMO_IND .... 12");
      printf("\n\nTipo de Signal que se desea leer:  ");
      do{
        scanf("%d", &c);
      }while((c < 1) || (c > 14));
      switch(c)
      {
        case 1:  *signal = ED_ESTABLEC_REQ; break;
        case 2:  *signal = ED_ESTABLEC_IND; break;
        case 3:  *signal = ED_ESTABLEC_CONF; break;
        case 4:  *signal = ED_LIBERACION_REQ; break;
        case 5:  *signal = ED_LIBERACION_IND; break;
        case 6:  *signal = ED_LIBERACION_CONF; break;
        case 7:  *signal = ED_DATOS_REQ; break;
        case 8:  *signal = ED_DATOS_IND; break;
        case 9:  *signal = ED_UNIDAD_DATOS_REQ; break;
        case 10: *signal = ED_UNIDAD_DATOS_IND; break;
        case 11: *signal = ED_LIBERACION_MEMO_REQ; break;
        case 12: *signal = ED_LIBERACION_MEMO_IND; break;
```

```
      }
      break;
    case 2:
      printf("\n     GED_LIBERACION_MEMO_REQ ........ 1");
      printf("\n     GED_LIBERACION_MEMO_IND ........ 2");
      printf("\n     GED_ASIGNACION_IET_IND ......... 3");
      printf("\n     GED_ASIGNACION_IET_REQ ......... 4");
      printf("\n     GED_NO_DISPONIBLE_IET_REQ ...... 5");
      printf("\n     GED_SUPRESION_IET_REQ .......... 6");
      printf("\n     GED_ERROR_IND .................. 7");
      printf("\n     GED_UNIDAD_DATOS_REQ ........... 8");
      printf("\n     GED_UNIDAD_DATOS_IND ........... 9");
      printf("\n\nTipo de Signal que se desea leer:  ");
      do{
        scanf("%d", &c);
      }while((c < 1) || (c > 9));
      switch(c)
      {
        case 1:  *signal = GED_LIBERACION_MEMO_REQ; break;
        case 2:  *signal = GED_LIBERACION_MEMO_IND; break;
        case 3:  *signal = GED_ASIGNACION_IET_IND; break;
        case 4:  *signal = GED_ASIGNACION_IET_REQ; break;
        case 5:  *signal = GED_NO_DISPONIBLE_IET_REQ; break;
        case 6:  *signal = GED_SUPRESION_IET_REQ; break;
        case 7:  *signal = GED_ERROR_IND; break;
        case 8:  *signal = GED_UNIDAD_DATOS_REQ; break;
        case 9:  *signal = GED_UNIDAD_DATOS_IND; break;
      }
      break;
    case 3:
      printf("\n    FI_DATOS_ENVIADOS ...... 1");
      printf("\n    FI_DATOS_RECIBIDOS ..... 2");
      printf("\n    FI_ERROR_IND ........... 3 ");
      printf("\n\nTipo de Signal que se desea leer:  ");
      do{
        scanf("%d", &c);
      }while((c < 1) || (c > 3));
      switch(c)
      {
        case 1: *signal = FI_DATOS_ENVIADOS;  break;
        case 2: *signal = FI_DATOS_RECIBIDOS; break;
        case 3: *signal = FI_ERROR_IND;       break;
      }
      break;
    case 4:
      printf("\n    MUX_TR_DATOS_REQ (único signal)... 1");
      printf("\n      NOTA: El resto son mensajes (tipoMensajeMuxTR)");
      printf("\n    MUX_TR_DATOS_IND ................ 2");
      printf("\n    MUX_TR_ENLACE_LIBERADO_REQ ..... 3");
      printf("\n    MUX_TR_ENLACE_LIBERADO_IND ..... 4");
      printf("\n    MUX_TR_ID_PROCESO_REQ .......... 5");
      printf("\n    MUX_TR_DATOS_ENVIADOS_IND ...... 6");
      printf("\n\nTipo de Signal que se desea leer:  ");
      do{
        scanf("%d", &c);
      }while((c < 1) || (c > 6));
      switch(c)
      {
        case 1: *signal = MUX_TR_DATOS_REQ;           break;
        case 2: *signal = MUX_TR_DATOS_IND;           break;
        case 3: *signal = MUX_TR_ENLACE_LIBERADO_REQ; break;
        case 4: *signal = MUX_TR_ENLACE_LIBERADO_IND; break;
        case 5: *signal = MUX_TR_ID_PROCESO_REQ;      break;
        case 6: *signal = MUX_TR_DATOS_ENVIADOS_IND;  break;
      }
      break;
    case 5:
      printf("\n\nTipo de Signal que se desea leer (valor decimal): ");
      scanf("%d", &*signal);
      break;
  } /*switch(c)*/
} /*EligeSignal(signal)*/
```

```
void EligeError( error)
UWORD *error;
{
  /*-------------------------------------------------------------------------
    Función auxiliar a la función FRecibeDatos. Se utiliza para poder
    seleccionar el tipo de error asociado a una trama recibida que podría
    recibirse del nivel físico
  -------------------------------------------------------------------------*/
  UWORD c;
  printf("\n      KNO_ERROR     ..................... 0");
  printf("\n      ERROR_DESACT  ..................... 1");
  printf("\n      Cualquier otro tipo de Error  .... 2");
  printf("\n... Tipo de error que desea leer : ");
  do{
    scanf("%d", &c);
  }while(c > 2);
  switch(c)
  {
    case 0:  *error = KNO_ERROR; break;
    case 1:  *error = ERROR_DESACT; break;
    case 2:  *error = ERROR_OVERRUN; break;
  }
} /*EligeError( error)*/


UWORD KWait(void)
{
  /* Unicamente se utiliza para devolver el código que inidca que se ha
     recibido un signal */
  return( KINTERRUP );
}


void  Espera(void)
{
  /* Para realizar una parada en la ejecución del código en la simulación */
  printf("\n . . . Pulse una tecla para continuar . . .  ");
  getch();
  printf("\n\n");
}


void DisplayData( data, dataLength, tipoDisplay)
UBYTE *data;
UWORD dataLength;
char tipoDisplay;
{
  /*-------------------------------------------------------------------------
    Muestra en la pantalla los bytes u octetos que hay a partir de la
    posición a la que apunta el puntero data. Utilizaremos siempre la opción
    de visualización de octetos como pareja de números hexadecimales
  -------------------------------------------------------------------------*/
  UWORD c;
  printf("<");
  for( c=0; c<dataLength; c++ )
  {
    if (tipoDisplay == 'x' || tipoDisplay == 'X')
      printf("%02x", data[c]);
    else
      printf("%c", data[c] );
    if ( c+1 != dataLength )
      printf(",");
  }
  printf("> : %d",c);
} /*DisplayData*/


UBYTE KIDProceso(void)
{
  /* Se solicita un valor para identificar el proceso simulado*/
```

```
  UBYTE valor;
  printf("\n\n\nUBYTE KIDProceso(void)");
  printf("\n -RETURN-           <UWORD> : <%%d>... ");
  scanf("%d", &valor );
  Espera();
  return( valor);
}


UWORD KPideMemoria( longDatos, datos)
UWORD longDatos;
UWORD **datos;
{
  /*-------------------------------------------------------------------------
    Esta función devuelve una cantidad de memoria dinámica a la que apuntará el
    puntero datos. Esa cantidad es el número de octetos especificado en la
    variable longDatos
  -------------------------------------------------------------------------*/
  printf("\n\nUWORD KPideMemoria(UWORD longDatos, UWORD **datos)");
  printf("\n        -IN\\OUT- Parameters :");
  printf("\n          <UWORD:longDatos> : %d", longDatos);
  *datos= (UWORD *)malloc(longDatos);
  printf("\n                  [[datos]] : [%p]", *datos);
  if (*datos == NULL)
  {
    printf("\n -RETURN-           <UWORD> : KNO_MEMORIA");
    Espera();
    return( KNO_MEMORIA );
  }
  else
  {
    printf("\n -RETURN-           <UWORD> : KNO_ERROR");
    Espera();
    return( KNO_ERROR );
  }
} /*KPideMemoria*/


UWORD KLiberaMemoria( datos)
UWORD *datos;
{
  /* Libera la zona de memoria a la que apunta el puntero datos */
  printf("\n\nUWORD KLiberaMemoria(UWORD *datos)");
  printf("\n            -IN- Parameters :");
  printf("\n                    [datos] : [%p]", datos );
  printf("\n -RETURN-           <UWORD> : KNO_ERROR");
  free( datos);
  Espera();
  return( KNO_ERROR );
}


UWORD KEscribePipe( pipe, data, longData, espera)
UWORD pipe, longData, espera;
void *data;
{
  /* Se escribe en pantalla los distintos campos del mensaje que se quiere
     "guardar" en el pipe: */

  MENSAJE23 *p;
  /* p apunta a una estructura tipo MENSAJE23 que empieza en la posición
     apuntada por el puntero data: */
  p = (MENSAJE23 *)(data);

  printf("\n\nUWORD KEscribePipe(UWORD pipe, void *data, UWORD longData, UWORD
espera)");
  printf("\n -IN/OUT- Parameters :");
  printf("\n        <UWORD:pipe> : %d", pipe);
  printf("\n         <data> Tipo : mensaje23.tipo      = ");
  printf("%04xx = ", p->tipo);
  DisplaySignal( p->tipo);
  printf("\n                       mensaje23.longDatos = %d", p->longDatos);
```

```
  printf("\n                          mensaje23.datos     = ");
  DisplayData( p->datos, p->longDatos, 'x');
  printf("\n    <UWORD:longitud> : %d",longData);
  if ( espera == KWAIT )
    printf("\n      <UWORD:espera> : KWAIT");
  else
    printf("\n      <UWORD:espera> : KNOWAIT");
  printf("\n -RETURN-    <UWORD> : KNO_ERROR");
  Espera();
  return( KNO_ERROR );
} /*KEscribePipe*/


UWORD KLeePipe( pipe, data, longData, espera)
UWORD pipe, longData, espera;
void *data;
{
  /*-------------------------------------------------------------------------
    Se leerá del teclado el supuesto mensaje que, utilizando el servicio del
    kernel, se leería del pipe
  -------------------------------------------------------------------------*/
  UWORD i;
  UWORD lengthData;
  UBYTE *memo, dato;
  UWORD resp;
  MENSAJE23 *p;
  /* p apunta a una estructura tipo MENSAJE23 que empieza en la posición
     apuntada por el puntero data: */
  p = (MENSAJE23 *)(data);

  printf("\n\nUWORD KLeePipe(UWORD pipe, void *data, UWORD longData, UWORD
espera)");
  printf("\n    -IN/OUT- Parameters :");
  printf("\n           <UWORD:pipe> : %d", pipe);
  printf("\n       <UWORD:longData> : %d", longData);
  printf("\n            <data> Tipo : mensaje23.tipo = ");
  EligeSignal( &p->tipo);   /* scanf("%d", &p->tipo);*/
  printf("\n            <data> Tipo : mensaje23.tipo = ");
  printf("%04xx = ", p->tipo);
  DisplaySignal( p->tipo);
  printf("\n                              mensaje23.longDatos = ");
  scanf("%d", &(lengthData));
  p->longDatos = lengthData;
  p->datos = (UBYTE *)(malloc(lengthData));

  printf("   Desea leer los datos y visualizarlos después (1/0): ");
  scanf("%d", &resp);
  if (resp == 1)
  {
    printf("\n   Introduzca ahora los datos:");
    for ( i=0; i < (p->longDatos); i++)
    {
      printf("\n     Dato en posición relativa [%03d] (hex): ", i);
      scanf("%x", &dato);   /*dato = getche(); */
      (p->datos)[i] = dato;
    }
    printf("\n                              mensaje23.datos     = ");
    DisplayData( p->datos, p->longDatos, 'x');
  }

  printf("\n       <UWORD:longData> : %d", longData);
  if (espera == KWAIT)
    printf("\n         <UWORD:espera> : KWAIT");
  else
    printf("\n         <UWORD:espera> : KNOWAIT");
  printf("\n -RETURN-       <UWORD> : KNO_ERROR");
  Espera();
  return( KNO_ERROR );
} /*KLeePipe*/


UWORD KEsperaContador( contador)
```

```
UWORD contador;
{
  /*-------------------------------------------------------------------------
    Se solicita elegir entre dos opciones: devolver el código KINTERRUP (para
    indicar que queremos dar un signal al proceso) o devolver el código
    KNO_ERROR indicando que ha terminado la supuesta cuenta del temporizador
    o contador de tiempo
  -------------------------------------------------------------------------*/
  char c;
  printf("\n\n\nUWORD KEsperaContador(UWORD contador)");
  printf("\n            -IN- Parameters :");
  printf("\n           <UWORD:contador> : %d",contador);
  do{
    printf("\n             1. KNO_ERROR.\n            2. KINTERRUP.\n");
    printf("                          Choose return value :");
    c = getche();
  }while ( c != '1'  &&  c != '2');
  if ( c == '1')
  {
    printf("\n -RETURN-           <UWORD> : KNO_ERROR");
    Espera();
    return( KNO_ERROR );
  }
  else
  {
    printf("\n -RETURN-           <UWORD> : KINTERRUP");
    Espera();
    return( KINTERRUP );
  }
} /*KEsperaContador*/


UWORD KSolicitaContador( contador)
UWORD *contador;
{
  /* Se solicita introducir un valor (0 a 65535) para identificar el contador
     utilizado: */
  printf("\n\n\nUWORD KSolicitaContador(UWORD *contador)");
  printf("\n            -IN- Parameters :");
  printf("\n            [Dir]<contador> : [%p]<%%d>... ",contador);
  scanf("%d",contador);
  printf(" -RETURN-           <UWORD> : KNO_ERROR");
  Espera();
  return( KNO_ERROR );
}


UWORD KActivaContador( contador, tiempo)
UWORD contador;
UWORD tiempo;
{
  /* Se muestra en pantalla el valor del contador y el tiempo que se quiere
     que cuente */
  printf("\n\nUWORD KActivaContador(UWORD contador, UWORD tiempo)");
  printf("\n            -IN- Parameters :");
  printf("\n           <UWORD:contador> : %d", contador);
  printf("\n        <UWORD:tiempo>(seg) : %d", tiempo);
  printf("\n -RETURN-           <UWORD> : KNO_ERROR");
  Espera();
  return( KNO_ERROR );
}


/*------------ FUNCIONES RELACIONADAS CON EL NIVEL FISICO: -----------------*/
UWORD FIniciaSCC( scc, protocolo, param)
UWORD scc;
UWORD protocolo;
PARAM_SCC *param;
{
  /*-------------------------------------------------------------------------
    Para iniciar los parámetros del Controlador Serie de Comunicaciones (que
```

```
    llamaremos Puerto Físico). Se muestra en pantalla los parámetros con los
    que se iniciaría el puerto físico
  ------------------------------------------------------------------------*/
  printf("\n\nFIniciaSCC(UWORD scc, UWORD protocolo, PARAM_SCC *param)");
  printf("\n    -IN/OUT- Parameters :");
  printf("\n    Puerto físico idSCC : %d", scc);
  printf("\n    param->hdlc.IDProceso  = %d", param->hdlc.IDProceso);

  printf("\n    param->hdlc.signal[0]  = ");
  DisplaySignal(param->hdlc.signal[0]);
  printf("\n    param->hdlc.signal[1]  = ");
  DisplaySignal(param->hdlc.signal[1]);
  printf("\n    param->hdlc.signal[2]  = ");
  DisplaySignal(param->hdlc.signal[2]);

  printf("\n -RETURN-       <UWORD> : TRUE");
  Espera();
  return(TRUE);
}


UWORD FActivaSCC( scc)
UWORD scc;
{
  /*------------------------------------------------------------------------
     Para simular la activación del puerto físico. Se pedirá seleccionar
     entre las dos posible opciones: que si se active el nivel físico o
     que no se active
  ------------------------------------------------------------------------*/
  UWORD resp;
  printf("\n\nFActivaSCC( UWORD scc)");
  printf("\n    -IN/OUT- Parameters :");
  printf("\n    Puerto físico idSCC : %d", scc);
  printf("\n ¿Desea:     -RETURN-    <UWORD> : TRUE (SI/NO = 1/0)... ?:  ");
  do{
    scanf("%d", &resp);
  }while((resp != 0) && (resp != 1));
  if (resp == 0)
  {
    printf("\n -RETURN-       <UWORD> : FALSE");
    Espera();
    return(FALSE);
  }
  else
  {
    printf("\n -RETURN-       <UWORD> : TRUE");
    Espera();
    return(TRUE);
  }
}


UWORD FEnviaDatos( scc, datos1, longitud1, datos2, longitud2)
UWORD scc;
void *datos1, *datos2;
UWORD longitud1, longitud2;
{
  /*------------------------------------------------------------------------
    El objetivo de la simulación de esta función es mostrar en la pantalla los
    datos (trama con su cabecero y campo de información) que se enviaría al
    puerto físico de destino
  ------------------------------------------------------------------------*/
  UWORD resp;
  printf("\n\nFEnviaDatos(UWORD scc, void *datos1, UWORD longitud1, ");
  printf("\n  void *datos2, UWORD longitud2)");
  printf("\n             -IN/OUT- Parameters :");
  printf("\n  <UWORD>  Puerto físico destino : %d",scc);
  printf("\n ¿Desea:     -RETURN-    <UWORD> : TRUE (SI/NO = 1/0)... ?:  ");
  do{
    scanf("%d", &resp);
  }while((resp != 0) && (resp != 1));
  if (resp == 0)
```

```
  {
    printf("\n -RETURN-       <UWORD> : FALSE");
    return(FALSE);
  }
  printf("\n            <datos1>:<longitud1> : \n");
  DisplayData( datos1, longitud1, 'x');
  printf("\n            <datos2>:<longitud2> : \n");
  DisplayData( datos2, longitud2, 'x');
  printf("\n -RETURN-       <UWORD> : TRUE");
  Espera();
  return(TRUE);
} /*FEnviaDatos*/


UWORD FRecibeDatos( scc, datos, longitud, error)
UWORD scc;
void  **datos;
UWORD *longitud;
UWORD *error;
{
  /*-------------------------------------------------------------------------
    El objetivo de la simulación de esta función es leer del teclado la
    supuesta trama recibida del nivel físico y el tipo de error que el nivel
    físico indicaría utilizando la función auxiliar EligeError
    (se leerá KNO_ERROR o alguno de los erorres contemplados)
  -------------------------------------------------------------------------*/
  UWORD i;
  UBYTE dato;
  UBYTE *p;
  printf("\n\nFRecibeDatos(UWORD scc, void **datos, UWORD *longitud, UWORD
*error)");
  printf("\n           -IN/OUT- Parameters :");
  printf("\n  <UWORD> Puerto físico origen : %d", scc);
  printf("\n            <UWORD> Tipo error : ");
  EligeError( &*error);
  printf("\n            <UWORD> Tipo error : %x", *error);
  printf("\n            <UWORD>   longitud : ");
  scanf( "%d", &*longitud);
  *datos = (UBYTE *)(malloc(*longitud));
  p = (UBYTE *)(*datos);
  printf("   Introduzca ahora los datos:\n");
  for (i=0; i < (*longitud); i++)
  {
    printf("     Dato en posición relativa [%03d] (hex): ", i);
    scanf("%x", &(p[i]));   /*dato = getche(); */
  }
  *datos = p;
  printf("\n            <datos>:<longitud> :");
  DisplayData( *datos, *longitud, 'x');
  printf("\n -RETURN-       <UWORD> : TRUE");
  Espera();
  return(TRUE);
} /*FRecibeDatos*/
```

```
/*****************************************************************************/
/*                              N3_SIM.C                                     */
/*          TRABAJO FIN DE CARRERA  SANTIAGO MARTIN ROMANI:                  */
/*              IMPLEMENTACION PROTOCOLO LAP-D PARA EL                       */
/*            MICROCONTROLADOR DE COMUNICACIONES MC68302                     */
/*                                                                           */
/*                SIMULACION ENTIDAD-PROCESO NIVEL 3                         */
/*****************************************************************************/


#define UNIX
/*#define $CODESEG S105*/
/*#define $INITSEG S205*/
/*#define $DATASEG S305*/

/*------------ FICHEROS INCLUIDOS: ------------------------------------------*/
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <alloc.h>
#include "basico.h"    /* ..\kernel\basico.h */
#include "kernel.h"    /* ..\kernel\kernel.h */
#include "inicio.h"
#include "n2n3.h"
#include "ged.hp"
#include "nivel2.hp"
#include "nivel1.h"
#include "mux_tr.h"
#include "mux.hp"



/*------------ DECLARACION DE VARIABLES GLOBALES: --------------------------*/
UWORD pipeEscN3N2;
UWORD pipeLecN3N2;

/*----------- Identificadores utilizados en el proceso de comunicacion: ----*/
UBYTE idProcesoN3;  /*id. proceso nivel 3 para la comunicacion con nivel 2*/
UBYTE idProcesoN2;  /*id. proceso nivel 2 con el que nos comunicaremos*/

MENSAJE23 mensaje23;
UWORD signal;
UBYTE *datos;
UWORD longDatos;


/*------------ DEFINICIONES DE PROTOTIPOS DE FUNCIONES: --------------------*/
void  main( void);
void  SimulacionN3main( UBYTE identificadorN2, UWORD pipeEscrituraN3N2,
  UWORD pipeLecturaN3N2);
void  PresentaPantalla( void);
void  FormarMensaje23( MENSAJE23 *mensaje23, UWORD tipoMens, UWORD longDatos,
  UBYTE *datos);


/*------------- PROTOTIPOS DE FUNCIONES DEL KERNEL: ------------------------*/
void  Espera(void);
void  DisplayData( UBYTE *data, UWORD dataLength , char tipoDisplay);
void  DisplaySignal( UWORD signal);
UWORD KSignal( UBYTE proceso, UWORD signal);
UWORD KLeeSignal( UWORD *signal);
void  EligeSignal( UWORD *signal);

UWORD KWait( void);

UBYTE KIDProceso( void);

UWORD KSolicitaContador( UWORD *contador);
UWORD KActivaContador( UWORD contador, UWORD tiempo);
UWORD KEsperaContador( UWORD contador);
```

```
UWORD KPideMemoria( UWORD longDatos, UWORD **datos);
UWORD KLiberaMemoria( UWORD *datos);

UWORD KSolicitaPipe( UWORD *pipe);
UWORD KEscribePipe( UWORD pipe, void *data, UWORD longData, UWORD espera);
UWORD KLeePipe( UWORD pipe, void *data, UWORD longData, UWORD espera);



/*------------ DECLARACION DE FUNCIONES: ----------------------------------*/
void  main( void)
{
  SimulacionN3main( 2, 10, 11);
}

void  SimulacionN3main( identificadorN2,
  pipeEscrituraN3N2, pipeLecturaN3N2)
UBYTE identificadorN2;
UWORD pipeEscrituraN3N2;
UWORD pipeLecturaN3N2;
{
  UWORD opcion;
  UWORD i;

  /*Parámetros de Entrada al Proceso que representa a la entidad de nivel 3:*/
  idProcesoN2 = identificadorN2;
  pipeEscN3N2 = pipeEscrituraN3N2;
  pipeLecN3N2 = pipeLecturaN3N2;

  /* Se solicita al Kernel que nos indique cuál es núestro identificador de
     proceso: */
  idProcesoN3 = KIDProceso();

  /*---------- Se forma y envía el primer mensaje al nivel 2: ---------------
    Se utiliza para indicar a la entidad de Capa de Enlace de Datos cuál es
    el identificador de la entidad de capa 3 y el pipe de lectura (pipe de
    escritura para el nivel 2)
  ------------------------------------------------------------------------*/
  FormarMensaje23( &mensaje23, idProcesoN3, pipeLecN3N2, NULL);
  KEscribePipe( pipeEscN3N2, (void *)(&mensaje23), LONG_MENSAJE23, KWAIT);

  while(TRUE)  /* El proceso vive para siempre */
  {
    PresentaPantalla();
    scanf("%d", &opcion);
    switch( opcion)
    {
      case 1:  /*Leer signal*/
        if (KLeeSignal( &signal) == KNO_ERROR)
        {
          printf("\n\nSignal recibido del nivel 2:    ");
          DisplaySignal( signal);
          switch( signal)
          {
            case ED_UNIDAD_DATOS_IND:
            case ED_DATOS_IND:
              printf("\n\nSe realiza lectura automática del pipe:");
              KLeePipe( pipeLecN3N2, (VOID *)(&mensaje23), LONG_MENSAJE23,
                KWAIT);
              printf("\nPresentación mensaje recibido:");
              printf("\n    mensaje23.tipo      = %04xx = ", mensaje23.tipo);
              DisplaySignal( mensaje23.tipo);
              printf("\n    mensaje23.longDatos = %d", mensaje23.longDatos);
              printf("\n    mensaje23.datos     = ");
              DisplayData( mensaje23.datos, mensaje23.longDatos, 'x');

              FormarMensaje23( &mensaje23, ED_LIBERACION_MEMO_REQ,
                mensaje23.longDatos, mensaje23.datos);
              KEscribePipe( pipeEscN3N2, (void *)&mensaje23, LONG_MENSAJE23,
```

```
                KWAIT);
              KSignal( idProcesoN2, ED_LIBERACION_MEMO_REQ);
              printf("\n\nLiberando la memoria correpondiente a la información");
              printf("\n  contenida en el mensaje asociado");
              printf("\n\nSe envía la señal ED_LIBERACION_MEMO_REQ con su
respectivo mensaje");
              break;
            case ED_LIBERACION_MEMO_IND:
              printf("\nSe realiza lectura automática del pipe para liberar la
información contenida");
              printf("\n  en el mensaje asociado");
              KLeePipe( pipeLecN3N2, (VOID *)(&mensaje23), LONG_MENSAJE23,
                KWAIT);
              KLiberaMemoria( (UWORD *)mensaje23.datos);
              break;
            default:
              break;
          }
        }
        else /*KLeeSignal( &signal) == KSIG_ERROR*/
          printf("\n\nADVERTENCIA: ¡No hay ningún signal esperando en la cola!");
        break;
      case 2:  /*Leer mensaje del pipe*/
        KLeePipe( pipeLecN3N2, (VOID *)(&mensaje23), LONG_MENSAJE23, KWAIT);
        printf("\n\nLectura del mensaje contenido en el pipe:");
        printf("\n    mensaje23.tipo      = %04xx = ", mensaje23.tipo);
        DisplaySignal( mensaje23.tipo);
        printf("\n    mensaje23.longDatos = %d", mensaje23.longDatos);
        printf("\n    mensaje23.datos     = ");
        DisplayData( mensaje23.datos, mensaje23.longDatos, 'x');
        break;
      case 3:  /*Establecer conexión de Capa de Enlace de Datos*/
        KSignal( idProcesoN2, ED_ESTABLEC_REQ);
        printf("\n\nEstableciendo conexión de Capa de Enlace de Datos");
        printf("\n\nSe envía la señal ED_ESTABLEC_REQ");
        break;
      case 4:  /*Liberar conexión de Capa de Enlace de Datos*/
        KSignal( idProcesoN2, ED_LIBERACION_REQ);
        printf("\n\nLiberando conexión de Capa de Enlace de Datos");
        printf("\n\nSe envía la señal ED_LIBERACION_REQ");
        break;
      case 5:  /*Enviar información sin confirmación (ED_UNIDAD_DATOS_REQ)*/
        printf("\n\nIntroduzca la información sin confirmación que desea enviar:
");
        printf("\n    Introduzca longitud de la información: ");
        scanf("%d", &longDatos);
        datos = (UBYTE *)(malloc(longDatos));
        printf("\n    Introduzca ahora los datos:");
        for ( i=0; i < longDatos; i++)
        {
          printf("\n    Dato en posición relativa [%03d] (hex): ", i);
          scanf("%x", &datos[i]);   /*dato = getche();*/
        }
        printf("\n");
        FormarMensaje23( &mensaje23, ED_UNIDAD_DATOS_REQ, longDatos, datos);
        KEscribePipe( pipeEscN3N2, (void *)&mensaje23, LONG_MENSAJE23, KWAIT);
        KSignal( idProcesoN2, ED_UNIDAD_DATOS_REQ);
        printf("\n\nSe envía el mensaje conteniendo la información sin
confirmación");
        printf("\n  y se envía la señal ED_UNIDAD_DATOS_REQ");
        break;
      case 6:  /*Enviar información con confirmación (ED_DATOS_REQ)*/
        printf("\n\nIntroduzca la información con confirmación que desea enviar:
");
        printf("\nIntroduzca longitud de la información: ");
        scanf("%d", &longDatos);
        datos = (UBYTE *)(malloc(longDatos));
        printf("\n    Introduzca ahora los datos:");
        for ( i=0; i < longDatos; i++)
        {
```

```
          printf("\n    Dato en posición relativa [%03d] (hex): ", i);
          scanf("%x", &datos[i]);   /*dato = getche();*/
        }
        printf("\n");
        FormarMensaje23( &mensaje23, ED_DATOS_REQ, longDatos, datos);
        KEscribePipe( pipeEscN3N2, (void *)&mensaje23, LONG_MENSAJE23, KWAIT);
        KSignal( idProcesoN2, ED_DATOS_REQ);
        printf("\n\nSe envía el mensaje conteniendo la información con
confirmación");
        printf("\n  y se envía la señal ED_DATOS_REQ");
        break;
      default:
        break;
    } /*switch(opcion)*/
  } /*while(TRUE)*/
} /*SimulacionN3main*/


void PresentaPantalla( void)
{
  printf("\n\n");

printf("\n*********************************************************************
*******");
  printf("\n*                  SIMULACION ENTIDAD-PROCESO DE NIVEL 3
       *");
  printf("\n*      TRABAJO FIN DE CARRERA  SANTIAGO MARTIN ROMANI (SEPTIEMBRE
1995)       *");
  printf("\n*
       *");
  printf("\n*                ¡¡ Bienvenido al menú de simulación !!
       *");

printf("\n*********************************************************************
*******");
  printf("\n\n       1.- Leer signal");
  printf("\n       2.- Leer del pipe");
  printf("\n       3.- Establecer Conexión de Capa de Enlace de Datos o Nivel 2");
  printf("\n       4.- Liberar Conexión de Capa de Enlace de Datos");
  printf("\n       5.- Enviar información sin confirmación
(ED_UNINDAD_DATOS_REQ)");
  printf("\n       6.- Enviar información con confirmación (ED_DATOS_REQ)");
  printf("\n\n                Por favor, elija su opción:  ");
}


void FormarMensaje23( mensaje23, tipoMens, longDatos, datos)
MENSAJE23 *mensaje23;
UWORD tipoMens, longDatos;
UBYTE *datos;
{
  mensaje23->tipo      = tipoMens;
  mensaje23->longDatos = longDatos;
  mensaje23->datos     = datos;

  /*OJO*/
  printf("\nFormando mensaje23...  <mensaje23->datos> <mensaje->longDatos>:\n");
  DisplayData( mensaje23->datos, mensaje23->longDatos, 'x');

  Espera();
}
```

```
UWORD indiceTRdestino;
if (FRecibeDatos( idSCC, (void **)&header, &lengHeader, &error)
  == TRUE)
{
  /*-------------------------------------------------------------------------
    si se ha producido algún error en la recepción de la trama, ésta
    se descarta. Los errores que el nivel físico puede detectar e
    indicar al Proceso Multiplexor del lado del Terminal de Red son:
      ERROR_OVERRUN
      ERROR_FRAMING
      ERROR_PARITY
      ERROR_BREAK
      ERROR_ABORT
      ERROR_NONOCTET
      ERROR_CRC
      ERORR_DESACT
  -------------------------------------------------------------------------*/
  if (error != 0)
  {
    if (header != NULL)
      /*-----------------------------------------------------------------------
        si se recibió una trama errónea (es decir, si el puntero está
        apuntando a alguna posición de memoria), será el multiplexor
        el encargado de liberarar la memoria que ocupa:
      -----------------------------------------------------------------------*/
      KLiberaMemoria( (UWORD *)header);
    if (error & ERROR_DESACT)
    {
      /*-----------------------------------------------------------------------
        si se produce un error de desactivación del nivel físico, se
        indicará a todos los procesos de nivel 2 del lado del Terminal
        de Red (que estén comunicándose con su respectivo remoto), que
        se ha caído el "enlace de nivel físico" y por tanto deberán
        pasar al estado "EnlaceLiberado":
      -----------------------------------------------------------------------*/
      for( i=0; i<LONG_VEC_MUX; i++)
      if ((vecMux[i].numIETasignado != IET_DE_GRUPO) &&
        (vecMux[i].libre == FALSE))
      {
        KSignal( vecMux[i].idProcesoTR, MUX_TR_ENLACE_LIBERADO_IND);
        vecMux[i].libre          = TRUE;
        vecMux[i].numIETasignado = IET_DE_GRUPO;
      }
      return(TRUE);
    } /*if (error & ERROR_DESACT)*/
  }
  else  /*if (error != 0)*/
  {
    /* si no hubo error al recibir la trama, se procesa: */
    tipoTrama = IdentificaTramaRecibida( header, lengHeader,
      &numIETrecibido);
    switch(tipoTrama)
    {
      /*-----------------------------------------------------------------------
        Si la trama es tipo GED_UI o TR_UI se envía al proceso TR master
          (proceso de nivel 2 del lado del Terminal de Red master)
        Si la trama es tipo SABME se enviará al proceso con el IET
          recibido o a un proceso TR libre. Si no hay ningún proceso
          libre, se descarta.
        El resto de tramas del protocolo LAP-D que se reciban
          se envían al proceso TR correspondiente con el IET recibido
      -----------------------------------------------------------------------*/
      case SABME:
        if(CheckTRdisponible( numIETrecibido, &indiceTRdestino)
          == DISPONIBLE)
          EnviarTramaAlTRdestino( indiceTRdestino);
        else
          KLiberaMemoria( (UWORD*)header);
        return(TRUE);
      case TRAMA_GED_UI:
      case TRAMA_TR_UI:
        EnviarTramaUIalTRmaster( indiceTRmaster);
```

```
          return(TRUE);
        case TRAMA_TR:
          if (CheckIETrecibido( numIETrecibido, &indiceTRdestino)
            == INCLUIDO)
            EnviarTramaAlTRdestino( indiceTRdestino);
          else
            KLiberaMemoria( (UWORD*)header);
          return(TRUE);
        default:
          KLiberaMemoria( (UWORD*)header);
      } /*switch(tipoTrama)*/
    } /*else if (error != 0)*/
  } /*if(FRecibeDatos(..) == TRUE*/
  return(TRUE);
} /*TratarFIdatosRecibidos*/


void EnviarTramaAlTRdestino( indiceTRdestino)
UWORD indiceTRdestino;
{
  /*-------------------------------------------------------------------------
    Se envía un mensaje (mensajeMuxN2) al proceso-entidad de Capa de Enlace de
    Datos del lado del Terminal de Red (a el indicado en la variable
    indiceTRdestino) y se le indica con el signal MUX_TR_DATOS_IND que
    equivale al signal FI_DATOS_ENVIADOS o FI_ERROR_IND
  -------------------------------------------------------------------------*/
  FormarMensajeMuxN2( &mensajeMuxN2, MUX_TR_DATOS_IND,
    vecMux[indiceTRdestino].idProcesoTR, header, lengHeader,
    NULL, LONG_CERO);
  KEscribePipe( vecMux[indiceTRdestino].pipeEscMuxTR,
    (void *)&mensajeMuxN2, LONG_MENSAJE_MUX_N2, KWAIT);
  KSignal( vecMux[indiceTRdestino].idProcesoTR, MUX_TR_DATOS_IND);
}

void EnviarTramaUIalTRmaster( indiceTRmaster)
UWORD indiceTRmaster;
{
  /*-------------------------------------------------------------------------
    Se envía un mensaje (mensajeMuxN2) al proceso-entidad master conteniendo
    la trama UI recibida del nivel físico y que se encuentra en las variables
    globales header y lengHeader (trama recibida y su longitud)
  -------------------------------------------------------------------------*/
  FormarMensajeMuxN2( &mensajeMuxN2, MUX_TR_DATOS_IND,
    vecMux[indiceTRmaster].idProcesoTR, header, lengHeader,
    NULL, LONG_CERO);
  KEscribePipe( vecMux[indiceTRmaster].pipeEscMuxTR,
    (void *)&mensajeMuxN2, LONG_MENSAJE_MUX_N2, KWAIT);
  KSignal( vecMux[indiceTRmaster].idProcesoTR, MUX_TR_DATOS_IND);
}

void  FormarMensajeMuxN2( messageMuxN2, tipoMessage, idProcesoTR,
  header, lengHeader, info, longInfo)
MENSAJE_MUX_N2 *messageMuxN2;
UWORD tipoMessage;
UBYTE idProcesoTR;
UBYTE *header, *info;
UWORD lengHeader, longInfo;
{
  /*-------------------------------------------------------------------------
    Esta función sirve para formar un mensaje (mensajeMuxN2) que luego será
    enviado a la entidad de Capa de Enlace de Datos correspondiente
  -------------------------------------------------------------------------*/
  messageMuxN2->tipoMensaje = tipoMessage;
  messageMuxN2->idProcesoTR = idProcesoTR;
  messageMuxN2->header      = header;
  messageMuxN2->lengHeader  = lengHeader;
  messageMuxN2->info        = info;
  messageMuxN2->longInfo    = longInfo;
}


UWORD IdentificaTramaRecibida( frame, frameLength, numIETrecibido)
```

```
UBYTE *frame;
UWORD frameLength;
UBYTE *numIETrecibido;
{
  /*-------------------------------------------------------------------------
    Identifica el tipo de trama recibida del nivel físico y extrae el valor
    del IET del campo de dirección de la trama recibida
  -------------------------------------------------------------------------*/
  UBYTE numIPASrecibido;
  if ((GET_BIT_ED(frame[0]) != 0) || (GET_BIT_ED(frame[1]) != 1))
    return(DESCONOCIDO);
  numIPASrecibido  = GET_IPAS(frame[0]);

  printf("\n frame[1] = %x", frame[1]);

  *numIETrecibido  = GET_IET(frame[1]);

  printf("\n numIETrecibido = %x", *numIETrecibido);

  if (numIPASrecibido == IPAS_CERO)
  {
    if (*numIETrecibido == IET_DE_GRUPO)
      return(TRAMA_TR_UI);
    if (frameLength == SHORT_HEADER)
      if ((GET_CTRL_TRAMAS_U(frame[2]) == SABME_MASK) &&
        (GET_BIT_PF_TRAMAS_U(frame[2]) == ACTIVO))
        return(SABME);
    return(TRAMA_TR);
  }
  if ((numIPASrecibido == IPAS_GED) && (*numIETrecibido == IET_GED))
    return(TRAMA_GED_UI);
  /* En cualquier otro caso, la trama es desconocida: */
  return(DESCONOCIDO);
} /*IdentificaTramaRecibida*/


UWORD CheckTRdisponible( numIETrecibido, indiceTRdestino)
UBYTE numIETrecibido;
UWORD *indiceTRdestino;
{
  UWORD i;
  /*-------------------------------------------------------------------------
    Cuando se recibe una trama SABME se ejecutará esta función:
      1.- Si existe un proceso de nivel 2 del lado TR que está comunicándose
          con un proceso de nivel 2 del lado ET con un numIETasignado igual
          al IET recibido en la trama, la enviamos a ese proceso
      2.- Si no se cumple lo anterior, pero existe un proceso del lado del TR
          que no está comunicándose con ningún proceso del lado del ET, se
          asignará a la nueva comunicación el IET recibido
      3.- Si no se cumple ninguna de las dos anteriores posibilidades, la
          función devuelve el valor NO_DISPONIBLE pués no existe ningún
          proceso comunicándose "a través" de ese IET ni tampoco se le puede
          asignar a otro pués están todos temporalmente ocupados.

      NOTA: La necesidad de dimensionar el vector vecMux a LONG_VEC_MUX = 12,
            se debe a la posibilidad de caída de un ET sin notificación al TR;
            en ese caso puede pasar (T203 + (3 * T200))sg antes de que
            el lado TR decida que la comunicación que emplea ese determinado
            IET ha dejado de existir y permitir que ese proceso TR pueda
            comunicarse utilizando otro IET.
              Para que una posible comunicación entre un TR y un ET pueda
            emplear un nuevo IET cuando por ejemplo, se enciende y apaga un
            ET, es necesario dimensionar el vecMux a un valor mayor que 8.
              Por otra parte podría ser que un determinado ET utilizase más de
            un IET comunicándose con varios procesos TR (cada uno con un IET
            diferente):
  -------------------------------------------------------------------------*/
  for( i=0; i<LONG_VEC_MUX; i++)
    if (vecMux[i].numIETasignado == numIETrecibido)
    {
      *indiceTRdestino = i;
      return(DISPONIBLE);
```

```
    }
  for( i=0; i<LONG_VEC_MUX; i++)
  {
    if ((vecMux[i].libre == FALSE) &&
      (vecMux[i].numIETasignado == IET_DE_GRUPO))
    {
      vecMux[i].numIETasignado = numIETrecibido;
      *indiceTRdestino = i;
      return(DISPONIBLE);
    }
  }
  *indiceTRdestino = NO_DISPONIBLE;
  return(NO_DISPONIBLE);
}


UWORD CheckIETrecibido( numIETrecibido, indiceTRdestino)
UBYTE numIETrecibido;
UWORD *indiceTRdestino;
{
  /*-------------------------------------------------------------------------
    Comprueba si alguno de los procesos de nivel 2 Terminal de Red, está
    comuncicándose con un proceso de nivel 2 remoto (Equipo Terminal)
    utilizando el IET recibido. Si es así, devuelve el índice de ese TR
    en la variable indiceTRdestino y la función toma el valor INCLUIDO.
    En caso contrario la función toma el valor NO_INCLUIDO.
  -------------------------------------------------------------------------*/
  UWORD i;
  for (i=0; i<LONG_VEC_MUX; i++)
    if (vecMux[i].numIETasignado == numIETrecibido)
    {
      *indiceTRdestino = i;
      return(INCLUIDO);
    }
  return(NO_INCLUIDO);
}


UWORD GuardarBtxMux( idProcesoTRemisor)
UWORD idProcesoTRemisor;
{
  /*-------------------------------------------------------------------------
     Inserta un elemento (idProcesoTRemisor) en la cola bufferTx
     y devuelve TRUE. Si la cola está llena, devuelve FALSE
  -------------------------------------------------------------------------*/
  /* Comprobamos si cola llena: */
  if ((frenteBtxMux == finalBtxMux+1) ||
    ((frenteBtxMux == 0) && (finalBtxMux == LONG_BUF_TX_MUX-1)))
    return(FALSE);
  else
  {
    bufferTxMux[finalBtxMux].idProcesoTRemisor = idProcesoTRemisor;
    if ((++finalBtxMux) == LONG_BUF_TX_MUX)
      finalBtxMux = 0;
    return(TRUE);
  }
} /*GuardaBtxMux*/


UWORD RecuperarBtxMux( void)
{
  /*-------------------------------------------------------------------------
     Por cada signal recibido FI_DATOS_ENVIADOS, se ejecutará esta función:
     Se envía al proceso TR emisor la señal MUX_TR_DATOS_ENVIADOS_IND para
     indicar que el nivel físico ya envío la trama correspondiente
  -------------------------------------------------------------------------*/
  /* Comprobación cola vacía: */
  if (frenteBtxMux == finalBtxMux)
    return(FALSE);
  else
  {
    KSignal( bufferTxMux[frenteBtxMux].idProcesoTRemisor,
```

```
      MUX_TR_DATOS_ENVIADOS_IND);
    if ((++frenteBtxMux) == LONG_BUF_TX_MUX)
      frenteBtxMux = 0;
    return(TRUE);
  }
} /*RecuperarBtxMux*/


UWORD GuardarBtramasAtx( idProcesoTRemisor, header, lengHeader,
  info, longInfo)
UBYTE idProcesoTRemisor;
UBYTE *header, *info;
UWORD lengHeader, longInfo;
{
  /*-------------------------------------------------------------------------
    Inserta un elemento en el buffer de tramas a transmitir:
      (identificador del proceso TR emisor,cabecero y su longitud,
      información y su longitud. Si la cola está llena, devuelve FALSE
  -------------------------------------------------------------------------*/
  /* Comprobamos si cola llena: */
  if ((frenteBtramasAtx == finalBtramasAtx+1) ||
    ((frenteBtramasAtx == 0) && (finalBtramasAtx == LONG_BUF_TRAMAS_A_TX-1)))
    return(FALSE);
  else
  {
    bufferTramasAtx[finalBtramasAtx].idProcesoTRemisor = idProcesoTRemisor;
    bufferTramasAtx[finalBtramasAtx].header            = header;
    bufferTramasAtx[finalBtramasAtx].lengHeader        = lengHeader;
    bufferTramasAtx[finalBtramasAtx].info              = info;
    bufferTramasAtx[finalBtramasAtx].longInfo          = longInfo;
    if ((++finalBtramasAtx) == LONG_BUF_TRAMAS_A_TX)
      finalBtramasAtx = 0;
    return(TRUE);
  }
} /*GuardarBtramasAtx*/


UWORD RecuperarBtramasAtx( idProcesoTRemisor, header, lengHeader,
  info, longInfo)
UBYTE *idProcesoTRemisor;
UBYTE **header, **info;
UWORD *lengHeader, *longInfo;
{
  /*-------------------------------------------------------------------------
    Cuando se ha recibido un FI_DATOS_ENVIADOS y se puede transmitir una trama
    en cola de espera, se recupera esa trama (cabecero e información con sus
    respectivas longitudes y el identificador de proceso TR que quiere
    enviarla. Si se logra enviar al nivel físico, se guardará el identificador
    de proceso, para que al recibir la señal FI_DATOS_ENVIADOS correspondiente
    a esa trama, se lo indiquemos (MUX_TR_DATOS_ENVIADOS_IND a ese proceso TR.
  -------------------------------------------------------------------------*/
  /* Comprobación cola vacía: */
  if (frenteBtramasAtx == finalBtramasAtx)
    return(FALSE);
  else
  {
    *idProcesoTRemisor = bufferTramasAtx[frenteBtramasAtx].idProcesoTRemisor;
    *header            = bufferTramasAtx[frenteBtramasAtx].header;
    *lengHeader        = bufferTramasAtx[frenteBtramasAtx].lengHeader;
    *info              = bufferTramasAtx[frenteBtramasAtx].info;
    *longInfo          = bufferTramasAtx[frenteBtramasAtx].longInfo;
    if ((++frenteBtramasAtx) == LONG_BUF_TRAMAS_A_TX)
      frenteBtramasAtx = 0;
    return(TRUE);
  }
} /*RecuperarBtramasAtx*/

/*------------------- Fin definiciones de funciones ----------------------*/
```

Bibliografía

Recomendaciones Q.920 y Q.921, Tomo VI - Fascículo VI.10, Libro Azul CCITT

Libro Curso "Red Digital Servicios Integrados"
Dr Juan Domingo Sandoval / Alfonso Medina Escuela
Escuela Técnica Superior de Ingenieros de Telecomunicación
Universidad de Las Palmas de Gran Canaria
I.S.B.N. 84-88412-14-2

MCC68K User's Guide (Manual del Compilador)
Microtec Research Inc.

ASM68K User's Guide (Manual Ensamblador - Linkador)
Microtec Research Inc.

XRAY 68K Debugger (Depurador)
Microtec Research Inc.

El lenguaje C
Kernighan & Ritchie
Ed. Prentice Hall

C: Manual de referencia
Herbert Schildt
Ed. McGraw-Hill

Entorno de Desarrollo

Compilador Borland C++ 3.1

MICE (Microtek In Circuit Emulation) para microprocesador M68000
(Emulador microprocesador M68000)