UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA

ESCUELA UNIVERSITARIA DE INGENIERIA TECNICA DE TELECOMUNICACION

TRABAJO FIN DE CARRERA

TITULO: INTRODUCCION A NLM. DESARROLLO DE APLICACION DE IMPRESION Y BÚSQUEDA REMOTA

ESPECIALIDAD: TELEFONIA Y TRANSMISION DE DATOS

AUTOR: LUIS JUAN SANTACREU RIOS

TUTOR: DOMINGO MARRERO MARRERO

MES/AÑO: JULIO/95

UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA

ESCUELA UNIVERSITARIA DE INGENIERIA TECNICA DE TELECOMUNICACION

TRABAJO FIN DE CARRERA

TITULO: INTRODUCCION A NLM. DESARROLLO DE APLICACION DE IMPRESIÓN Y BÚSQUEDA REMOTA

Fecha :

Fdo. Presidente **Fdo. Secretario** **Fdo. Vocal**

INDICE

TEMA 0
INTRODUCCIÓN

0.INTRODUCCIÓN

Este proyecto surge como la consecuencia más inmediata de un creciente interés por las redes locales, su entorno, posibilidades y programación.

El hecho de que fuera posible la programación bajo el entorno de Novell Netware induce hacia la búsqueda de información para el desarrollo de aplicaciones que utilizasen los medios y recursos que la red y en concreto su sistema operativo suministraba.

Básicamente hicieron falta como herramientas de programación:

- Compilador de C++ de la casa Watcom.
- Kit de desarrollo SDK de Novell.

Después de un estudio arduo y detallado de las herramientas de programación y de sus posibilidades se comenzó a pensar en el desarrollo de una aplicación cliente/servidor utilizando como interfaz el nivel de transporte.

Se comenzó desarrollando programas sencillos que únicamente mandaban y recibían mensajes entre cliente y servidor. El servidor con un Módulo Cargable Netware (NLM) y el cliente con un ejecutable.

El enfoque fué variando a medida que el desarrollo avanzaba pero siempre con la base de una comunicación utilizando TLI.

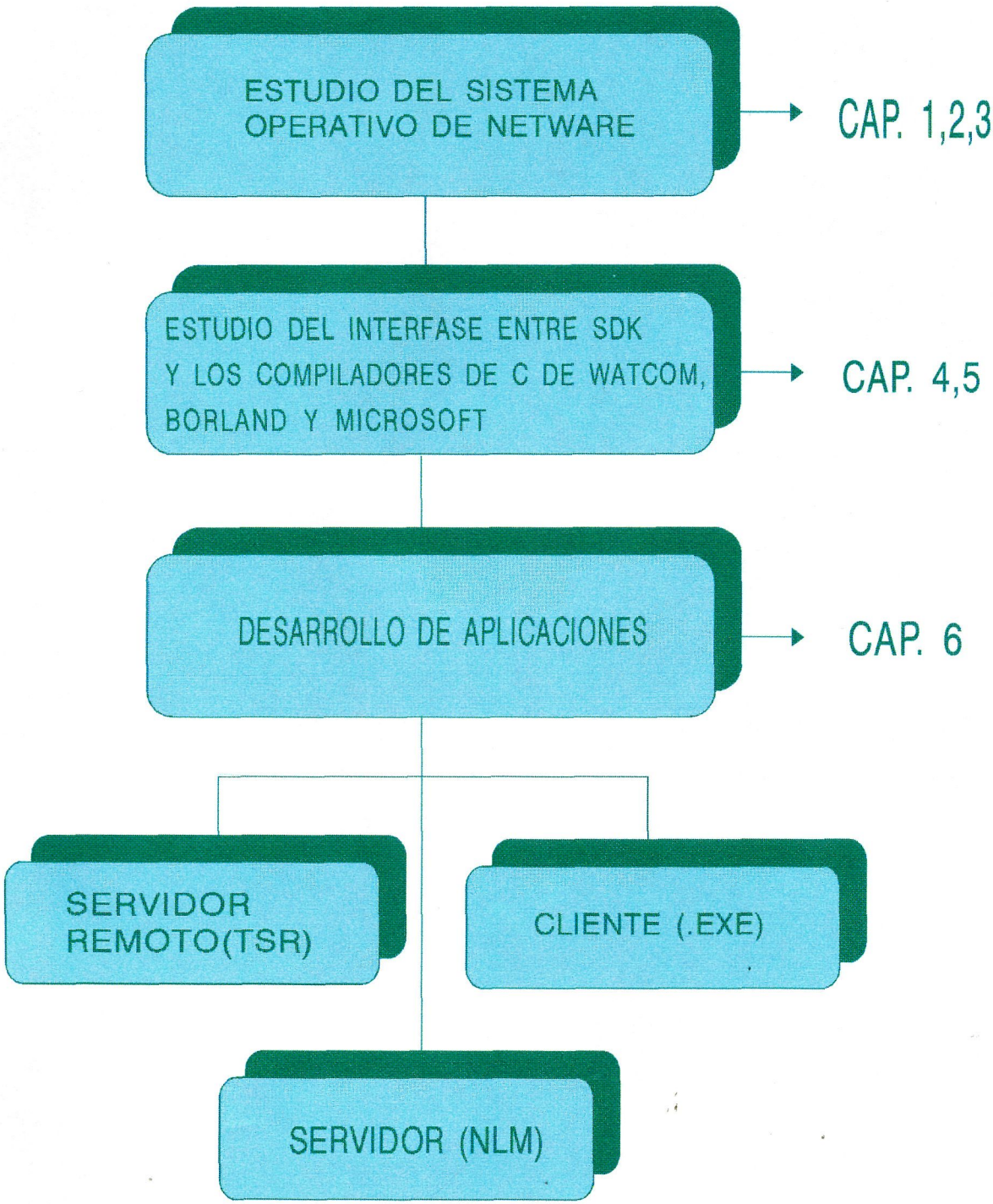
Los programas finales desarrollados fueron:

- Un programa para el servidor encargado de controlar una comunicación efectiva entre los programas que a continuación se describen:

➪ Un programa residente para una estación de trabajo que se encarga de poner a disposición de posibles clientes, servicios tales como la impresión de ficheros remota y la búsqueda de ficheros en las unidades locales de esa misma estación de trabajo.

➪ Un programa ejecutable que se encarga de utilizar los servicios suministrados por las estaciones de trabajo y antes reseñados.

A continuación se esquematiza cómo se desgloban los diferentes apartados del proyecto:



TEMA 1
CONCEPTOS BÁSICOS

1.CONCEPTOS BÁSICOS

Este capítulo pretende ser un primer contacto con lo que es el entorno del sistema operativo Netware y sus posibilidades. No se pretende una explicación exhaustiva, sino una introducción semiformal al NOS, con el objeto de que la exposición sobre el desarrollo de módulos NLM sea los mas clara posible.

1.1.Entorno del sistema operativo Netware

El sistema operativo Netware (NOS) es un sistema operativo especializado. Muchos de los sistemas operativos que funcionan en los ordenadores personales son de propósito general.

Un sistema operativo de propósito general está diseñado para proporcionar servicios que cubran las necesidades de muchos tipos de aplicaciones, siendo, por tanto, muy tolerantes a las distintas características de éstas. Las aplicaciones pueden escribirse sin preocuparse apenas de la cooperación con otros programas, ni de la compartición de recursos del ordenador, como la memoria y la CPU.

Netware es un sistema operativo de propósito específico ya que desde el principio fué diseñado para optimizar los servicios de red.

Para ser capaces de realizar las tareas más eficientemente, Netware asume que las aplicaciones (los NLM) que se ejecutan junto al sistema operativo conocen el entorno especializado en el que se encuentran, y actúan en consecuencia.

El hecho de que Netware tenga un alto rendimiento es consecuencia del funcionamiento tipo ***non-preemptive***. Esto quiere decir que tanto el sistema operativo como

las aplicaciones se ejecutan en su mayor parte sin que puedan ser interrumpidas por otras aplicaciones. Esta característica se conoce a veces como ***entorno amable,*** ya que se espera que las aplicaciones se comporten de forma "educada" en su uso de los recursos del ordenador.

En definitiva, si un módulo NLM no cede el control de la CPU de forma periódica, los demás módulos NLM no podrán ejecutarse, ya que carecen de acceso a la CPU.

Es labor del programador darse cuenta de las consecuencias que puede acarrear un acaparamiento de la CPU.

En general, el NOS confía en que todos los NLM se comporten debidamente, compartiendo la CPU mediante la cesión de control periódica.

De manera general NOS se ocupa de realizar las siguientes funciones:

-Gestión de memoria.
-Planificación.
-Protocolos de comunicación.
-Procesamiento del protocolo del núcleo Netware (NCP).
-Funciones de encaminamiento.
-Gestión de los sistemas de memoria caché.
-Acceso al sistema de ficheros.
-Bloqueo de ficheros y registros.
-Seguimiento de transacciones.
-Interfases de controladores de dispositivos.

Netware funciona en el ***modo protegido*** de la CPU, y utiliza las facilidades de direccionamiento y multitarea que ofrecen los procesadores 386 y 486. En este modo, la

memoria se puede direccionar como un intervalo contínuo de direcciones, permitiendo que la asignación y gestión de memoria sea más rápida y flexible. No es necesario cambiar de un segmento a otro, ya que toda la memoria está en uno solo.

Otra ventaja de este tipo de modo de funcionamiento es la posibilidad de ejecutar programas de manera concurrente, concepto conocido comúnmente como ***multitarea.*** Dentro de la terminología NetWare, los procesos o tareas reciben el nombre de ***procesos ligeros*** (threads) .

Cuando la CPU funciona en ***modo real*** el tamaño máximo que puede asignarse como bloque es 64 Kb, que es el límite de tamaño de un segmento de memoria.

1.2.¿Qué es un NLM?

Un módulo cargarble NetWare (NLM) es un programa que se ejecuta en la misma máquina que un servidor NetWare o un encaminador (router), ejecutándose junto con el sistema operativo NetWare o el software del encaminador.

En su forma ejecutable, un NLM es un fichero que el sistema operativo NetWare puede cargar (orden LOAD o descargar orden UNLOAD) y ejecutar en un servidor NetWare.

Sus diferentes extensiones dan una idea general sobre el servicio que proporcionan. Así los NLM con la extensión .NLM proporcionan normalmente algún tipo de servicio o función de utilidad, siendo éste del que nos ocuparemos posteriormente en el desarrollo. Los NLM con la extensión .DSK son controladores de disco, aquéllos con extensión .NAM son módulos del espacio de nombres, y aquéllos con extensión .LAN son controladores de tarjetas de interfaz de red.

1.3.Kernel del sistema operativo Netware

El ***kernel***, como su nombre indica, es el corazón o núcleo del sistema operativo. Es la parte que realiza las operaciones fundamentales.

Existen algunas diferencias entre los ***kernels*** de las distintas versiones de NetWare, nuestro estudio se centrará sobre las versiones 3.x. aunque se comentarán las características y diferencias fundamentales con respecto a la versión 4.0 ya que la portabilidad es posible.

Las unidades de programa en ejecución se denominan ***procesos ligeros.***

El sistema operativo junto con los módulos NLM generan procesos ligeros para la realización de ciertas funciones y tareas específicas. Ya que multitud de procesos ligeros concurren simultáneamente, el kernel tiene diversas colas donde se ubican los procesos ligeros, mientras esperan su turno para el uso de la CPU. Este turno se calcula en base al tipo de proceso ligero y a la prioridad otorgada cuando éste fué creado.

Un proceso ligero se ejecuta hasta que éste renuncia al control de la CPU. Las únicas interrupciones provienen de los elementos hardware.

Estos conceptos, aunque breves, son importantes y han de tenerse siempre en cuenta a la hora de programar.

1.5.Contextos

Un contexto engloba elementos tales como variables, pantallas, punteros, contadores, referencias, información de control y parámetros.

Dentro de los NLM existen tres tipos de contextos diferentes:

-Global con respecto al NLM y a todos los procesos ligeros dentro de él.
-Con respecto a un grupo de procesos ligeros previamente definido.
-Con respecto a un proceso ligero individual.

El contexto inicial se establece cuando se escribe main() en el código fuente.Cada vez que se creen procesos ligeros y grupos de procesos ligeros se establecen contextos adicionales.Es importante tener claro qué información se va a guardar en cada nivel de contexto de un NLM, así cómo de cuales son los elementos que se verán afectados en cada contexto.

Elementos cómo los ***semáforos*** son parte importante en el desarrollo de aplicaciones multiproceso ya que facilitan los cambios de contexto entre procesos ligeros. Estos permiten la cooperación entre procesos a la vez que facilitan el uso eficiente del tiempo de proceso del servidor.

1.5.Multiproceso

A la vez que múltiples NLMs pueden estar ejecutándose de manera concurrente o simultánea sobre el servidor NetWare, múltiples procesos ligeros (threads), pueden estar ejecutándose dentro del mismo código de cada uno de los NLMs.

Un thread es un proceso o flujo de ejecución que se ejecuta en el NOS.

A nivel comparativo un NLM se puede comparar con un programa en C y un proceso ligero con una función dentro de ese mismo programa. El multiproceso permite que se ejecuten de manera simultánea tanto los programas en C como las funciones que dentro de dichos programas coexisten.

Cada proceso ligero tiene una o más rutinas que realizan tareas como mostrar

información en pantalla, o recibir información de un servidor o de una estación de trabajo cliente. El sistema operativo identifica y registra cada proceso ligero con su ***bloque de contol de proceso (PCB),*** estructura que tiene información especial de ese proceso.

En el entorno de los NLM la programación multiproceso es algo fundamental. El programador ha de tener en cuenta que su programa se ejecutará en un entorno en el que otros estarán ejecutándose al mismo tiempo.

En la programación multiproceso es labor del programador saber que recursos hay que compartir y cómo compartirlos.

Si uno solo de los procesos ligeros no tiene un mecanismo incorporado para ceder el control de la CPU de manera regular, pueden surgir serios problemas en cuanto al rendimiento del sistema se refiere. Las consecuencias más probables pueden ser la ralentización del sistema e incluso el bloqueo y caida del servidor.

El NOS posee varias colas donde residen los proceso ligeros mientras esperan su turno de CPU.

Para las versiones 3.0, 3.1 y 3.11 de Netware existe unicamente la cola ***Runlist (FIFO)***. El programador crea y coloca los procesos ligeros en esta cola llamando a funciones tales como ***BeginThread, BeginThreadGroup, ThreadSwitch,...***

Para la versión 3.12 además de la cola ***RunList*** existe la cola ***DelayedWorkToDoList (LIFO).*** Un proceso ligero se sitúa en esta cola después de llamar a la función ***ThredSwitchWithDelay.*** Los procesos ligeros abandonan esta cola y pasan a la cola ***RunList*** (último paso antes de pasar a ejecutarse en la CPU) tras esperar un cierto número de cambios de contextos. El número de cambios de contexto recibe el nombre de ***handicap.***

Para la versión 4.0 de NetWare además de las dos colas citadas anteriormente existe la cola ***WorkToDoList*** y la cola ***LowPriorityRunList (FIFO).***

Un trabajo es una rutina que el sistema operativo debe ejecutar. Este no puede ejecutarse si no forma parte de un proceso ligero. Puede utilizarse la cola ***WorkToDoList*** para planificar un trabajo sin haberle asignado antes un proceso ligero. El sistema operativo crea y reserva una serie de procesos ligeros, que recogen los trabajos de esta cola y los ejecutan. Estos son los ***procesos ligeros trabajadores*** y son una de las mejoras introducidas por la versión 4.0 de Netware.

Un proceso ligero se sitúa en la cola ***LowPriorityRunList*** después de llamar a la función ***ThreadSwitchLowPriority*** . Esta cola es de baja prioridad y un proceso ligero que esté en esta cola se ejecutará sólo cuando las demás colas estén vacías.

1.6.Bindery (v.3.x)

El ***Bindery*** es una base de datos de propósito especial donde NetWare/386 mantiene información sobre recursos de la red y usuarios. Contiene definiciones para entidades tales como usuarios, grupos, colas, servidores y algún otro nombre de entidad de la red conocido como ***objeto***. Todos los servidores de ficheros tienen su propio ***Bindery*** y de este modo su propio grupo de objetos conocidos .

Los ***objetos*** (object) representan alguna entidad física o lógica. Se incluyen dentro de este tipo a usuarios, grupos de usuarios, servidores de ficheros, servidores de impresión...

1.7. Servicios de directorio (DIB, v.4.0)

Los servicios de directorio se encargan de facilitar el acceso a la red. Como ejemplos

de servicios podemos nombrar la creación y eliminación de directorios de red, el control de acceso de usuarios a directorios, información del estado de directorios...

Junto con los servicios de directorio tenemos la DIB que es una base de datos que almacena información sobre servidores y servicios, usuarios, impresoras, etc.

Para que los servidores NetWare 4.0 funcionen correctamente, debe de existir en todos ellos los servicios de directorio.

1.8 Diferencias entre la DIB y el Bindery

La información contenida en la DIB en la versión 4.0 de NetWare es guardada en versiones anteriores (3.0, 3.1, 3.11, 3.12) en el Bindery. Teniendo el mismo propósito el cambio se justifica por las siguientes razones:

- La DIB es de carácter distribuido mientras que el Bindery es centralizado.
- El acceso al Bindery está orientado al servidor, mientras que el acceso a la DIB está orientado a la red.
- La información almacenada por los servidores que poseen un Bindery es en su mayoría para uso propio mientras que la almacenada en los servidores con directorio es y se trata de toda la red.

1.9.Emulación del Bindery (v.4.0)

La versión 4.0 de Netware permite que tanto servidores como aplicaciones basados en la tecnología del Bindery puedan comunicarse con los servidores basados en el directorio gracias a la capacidad de emulación del Bindery.

Esta capacidad está incorporada en los servicios de directorio y puede estar activada o desactivada.

TEMA 2
OPTIMIZACIÓN DEL SERVIDOR

2.OPTIMIZACIÓN DEL SERVIDOR
(para el desarrollo de módulos NLM)

Para un buen desarrollo conviene conocer (y en cada caso poner en práctica) las posibilidades en cuanto a optimización se refiere de las condiciones de trabajo del "SERVIDOR".

2.1.Desactivación del "Sistema de Seguimiento de Transacciones"

La utilidad de control de transacciones de Netware (TTS) se utiliza para evitar la falta de integridad en la bases de datos en el caso de que fallase el sistema mientras se estuviera escribiendo en el disco.

Cuando TTS está activo, una secuencia de transacciones se ve como un solo evento que debe completarse o anularse.Si los cambios no se han completado porque cayó el sistema (bloqueos,reseteos..), Netware devuelve la base de datos a su estadio previo antes de que comenzaran los cambios.Las transacciones perdidas deberán de ser reescritas, pero la base de datos permanecerá intacta.

Para el caso en el que un NLM determinado mantenga una base de datos sobre el servidor debería de ser desactivada la utilidad TTS durante el testeo (no sería necesario si el NLM no hace uso del TTS).Esto fuerza a que el NLM se encargue de la apertura y cierre de todos los registros y se pueda observar si el NLM en cuestión maneja correctamente las llamadas al TTS aún cuando esté desactivado.

Para desactivar el TTS ejecutamos FCONSOLE y seleccionamos la opción Status en el menú principal.

2.2.Protección y gestión de memoria

La memoria se trata como un único gran segmento que se divide en partes según se reciben las solicitudes de asignación por parte de los NLM.

La asignación de memoria puede hacerse por bloques, páginas, intervalos o desde la memoria caché. Existe la posibilidad de indicar que un proceso se suspenda hasta que la memoria solicitada esté disponible. Las funciones del lenguaje C para asignar, desasignar y liberar memoria se encuentran en la librería CLIB.

Para las versiones 3.x y 4.0 de sistemas operativos, las aplicaciones NLM pueden correr sin protección de memoria. Estas comparten entonces el mismo espacio para código y datos. Se optimiza la velocidad de las aplicaciones pero no se provee de ningún tipo de protección al sistema operativo.

La protección de memoria concierne más bien a aplicaciones con misiones críticas en las que un error puede hacer que caiga el servidor.

Para las versiones 3.11 y 3.12 no existe protección de memoria para usuarios finales. Sin embargo, existe una protección de memoria disponible durante el ciclo de desarrollo. Esta protección se provee a través del uso del modulo cargable PROTECT.NLM.

Para la versión 4.0 sí existe una protección de memoria para usuarios finales además de para el desarrollo de aplicaciones.

El NOS 4.0 provee la protección de memoria en forma de DOMAIN.NLM. Este contempla dos modos distintos: el modo de sistema operativo y el modo protegido orientado a dominio.

Para más referencias sobre protección de memoria en la versión 4.0 de NOS ver "Getting Started" en los manuales del SDK.

2.3. Parámetros del NOS

Netware 3.x tiene un juego de Parámetros que pueden ser modificados cambiando así las características del sistema operativo. Estos permiten modificar los valores dados para la caché de archivos, de directorios, el sistema de archivos, comunicaciones, la memoria, el bloqueo de archivos, el sistema de control de transacciones, la codificación y otras características.

Para visualizar y cambiar los Parámetros de configuración puede ejecutarse el comando SET en la consola del servidor. El sistema operativo Netware 386 puede modificar automáticamente muchas de sus especificaciones para adaptarse a las necesidades de un entorno de red cambiante.

Muchas órdenes SET pueden introducirse desde la consola, mientras que otras sólo pueden usarse introduciéndolas en el archivo STARTUP.NCF.

El archivo AUTOEXEC.NCF sirve también para introducir órdenes SET con el fin de que el sistema arranque con una configuración determinada.

Se recomiendan a continuación los valores que deberían de tener algunos Parámetros para una mejor optimización del servidor:

➨ 2.3.1. *Display Old API Names = valor* (Mostrar nombres antiguos de API):

Este parámetro controla los mensajes visualizados cuando se carga un módulo que utiliza interfases de programación de aplicaciones (API) antiguas.

Las API fueron actualizadas con la versión 3.1 de Netware 386.

El valor por defecto es "Off" y el recomendado para el desarrollo de aplicaciones es "On" para el caso de programadores que estén actualizando los NLM de la versión 3.0 a las nuevas API de la versión 3.1.

Puede introducirse en el archivo STARTUP.NCF.

➨ ***2.3.2.Display Relinquish Control Alerts = valor*** **(Mostrar avisos de devolución de control):**

Este parámetro es de vital importancia para los programadores de NLM. Si introducimos "On", visualizaremos un mensaje que nos indicará si el NLM utiliza el procesador durante más de 0.4 segundos sin devolver el control a los restantes procesos.
El valor por defecto es "Off" y el recomendado para el desarrollo de aplicaciones es "On".
Puede introducirse en el archivo STARTUP.NCF.

➨ ***2.3.3.Pseudo Preemption Time = número:***

Este parámetro indica la cantidad de tiempo (en incrementos de 0.84 microsegundos) que se debe ejecutar un NLM antes de forzarlo a abandonar el control.

➨ ***2.3.4.Console Display Watchdog Logouts = valor*** **(Desconexión por falta de respuesta):**

Verifica que conexiones de estaciones están activas. Si una estación no envía una petición al servidor dentro de un período específico de tiempo, la función Watchdog envía un paquete al shell de la estación, invitando al envío de una réplica si aún está activa. Si la estación no responde, la función Watchdog envía paquetes a intervalos específicos de

tiempo.Si la estación sigue sin dar respuesta el servidor interrumpirá la conexión.

Si un NLM específico se comunica con un programa cliente se debería activar esta opción y permitir que el programa cliente estuviera inactivo durante más de 20 minutos (valor por defecto de la función Watchdog para interrumpir la conexión). Esto verificará que no se interfiere con las respuestas del shell de la estación de trabajo al envío de paquetes por parte de la función.

El valor recomendado es "On" (activa), por defecto es "Off".

➨ ***2.3.5.Auto Register Memory Above 16 Megabytes = valor*** **(Registro automático de memoria por encima de 16 MB):**

El valor por defecto de este parámetro es "On", lo que implica el registro de la memoria por encima de los 16 megabytes en computadores EISA.
Cuando tengamos placas que utilicen DMA en línea o controladoras de disco con control de bus AT, desactivaremos la opción incluyendo la orden en el archivo STARTUP.NCF con el valor "Off".

Estas placas sólo pueden direccionar como máximo 16MB (24 bits) de memoria pudiendo corromper la memoria del servidor de archivos. Si la placa es ISA o MCA, tendrá que ser sustituida por una placa que no utilice DMA en línea ni control de bus AT si queremos tener instalados más de 16 MB.

Es aconsejable el testeo (de los NLM que se estén desarrollando) en máquinas con menos y más de 16 MB de RAM y el valor del parámetro en "On", para asegurarse de que el NLM se ejecuta correctamente.

TEMA 3
INSTALACIÓN DEL SOFTWARE Y PRIMERA COMPILACIÓN

3.INSTALACIÓN DEL SOFTWARE Y PRIMERA COMPILACIÓN

3.1.Instalación del software y consejos prácticos

La instalación del software se puede dividir claramente en dos etapas:

- Instalación del compilador de Watcom.
- Instalación del software para el desarrollo de módulos NLM SDK.

Antes de comenzar con la instalación del NLM SDK es conveniente instalar primero el compilador y herramientas de C/386 o C/C++[32] de la casa Watcom. Para más información remitir a los manuales de Watcom.

Una vez completada la instalación del compilador se procederá con la instalación del NLM SDK.

La instalación del kit de desarrollo crea dos nuevos ficheros en el directorio en el que se ha instalado el NLM SDK (por ejemplo C:\NLMSDK\ ) que son:

-MAKEINIT
-NLMDEMO.BAT

El fichero MAKEINIT es de vital importancia, por lo que siempre ha de ser fácilmente localizable por el sistema. Este contiene información sobre la localización de los diferentes ficheros del SDK y ha sido generado durante el proceso de instalación.

Si se alteraran los nombres de los directorios o éstos fueran copiados en otra unidad diferente a la original habría que sustituir la antigua por la nueva información dentro del fichero MAKEINIT (tarea sumamente fácil de realizar).

El fichero NLMDEMO.BAT es un BAT que ejemplifica de manera sencilla el modo de generar un primer módulo NLM a partir de un simple programa en C (HELLO.C).

El programa de instalación copia y crea una serie de ficheros y directorios de los cuales los más importantes a destacar para cualquiera que comience con el desarrollo de módulos NLM son:

- **README.NOV**: notas importantes sobre el NLM SDK.

- **NLMDEMO.BAT**: un fichero BAT que compila y linka HELLO.C
- **NOVBIN\QMK386.EXE**: una utilidad que crea makefiles.

- **NOVBIN\MAKEINIT**: ya explicado.

- **NOVBIN\NLMLINKx**: existen dos versiones del enlazador NLM, una de ellas funciona en modo protegido y otra en modo real. El uso de una de estas dos versiones es imprescindible si se desea que el NLM sea "capacitado". Esto significa que los mensajes de salida del NLM pueden aparecer en distintos idiomas.

- **NOVH\; NOVH\SYS\; NOVH\ARPA\; NOVH\NETINET**: librerías de interfaz CLIB, las cuales se comentarán más adelante en profundidad.

- **NOVM\3.X**: NLMs específicos para las versiones 3.x.

- **NOVM\4.X**: NLMs específicos para la versión 4.0.

- **NOVX\...**: gran cantidad de ejemplos de NLMs.

3.2.Compilación de un MLM

Un NLM se compila de manera similar a la de cualquier otro programa en C. Lo que vamos a tratar son los aspectos específicos de la construcción de un NLM.

El compilador de C y el enlazador que se utilizan normalmente para construir módulos NLM son los de la casa Watcom aunque cabe la posibilidad de usar los de Borland y Microsoft.

Antes de compilar cualquier programa en C deberíamos asegurarnos que el fichero MAKEINIT se encuentra en un directorio contemplado en los path del sistema.

Un ejecutable denominado QMK386 se encarga de crear makefiles para nuestros programas.

Una vez generado el MAKEFILE de nuestro programa con el QMK386 (necesario para el proceso de compilación y montaje de nuestro NLM) procederemos a ejecutar el fichero WMAKE.

Algunas de las variables en el fichero MAKEFILE son SET en el fichero MAKEINIT.

Si WMAKE no encuentra los ficheros MAKEINIT y MAKEFILE no generará el NLM.

Para aplicaciones cliente/servidor se ha de realizar en su manera más sencilla dos programas (pueden ser más):

⇨ Uno para el servidor, que será un modulo NLM, pudiendo ser generado (después de diseñado) siguiendo los pasos anteriores.

⇨ Uno para el cliente, que puede ser generado con compiladores de C como el de Borland o el de MicroSoft, realizando un proyecto en el que se incluirá el código fuente y las librerias utilizadas en el código fuente y que son suministradas en el kit de desarrollo SDK.

TEMA 4
CARACTERÍSTICAS DEL NLM SDK

4.CARACTERÍSTICAS DEL NLM SDK

El juego de programas y herramientas del NLM SDK permiten el desarrollo de módulos NLM para las versiones 3.x y 4.0 de Netware. En lo sucesivo nos centraremos en las versiones 3.x para el desarrollo de NLMs aunque se hagan algunas referencias a la versión 4.0. Para más información sobre la versión 4.0 de Netware remitirse a los manuales del SDK.

4.1.Características generales

-Compatibilidad con el compilador de WATCOM C/386 y C/C++[32], el cual genera código de 32 bits.

-Protección de memoria para aplicaciones NLM (detección de referencias inválidas a memoria) haciendo uso de PROTECT.NLM (para las versiones 3.x) y de DOMAIN.NLM (para la versión 4.0).

-Comunicaciones asíncronas de entrada/salida (AIO) con soporte de tarjetas inteligentes tales como ARTIC y WNIM y puertos de comunicaciones tales como COM1 y COM2.

-Soporte de interfaz de transporte para los protocolos TCP/IP y IPX/SPX/SPX II.

-Servicios de interfaz de la capa de transporte. TLI es un API de AT&T que ha adquirido un amplio uso en el mundo de las comunicaciones.

-Programas que demuestran el uso de conexiones, comunicaciones, multiproceso, y varios servicios Netware.

-Interfaz de programación de aplicaciones (API) para el lenguaje C.

4.2.Requerimientos hardware del NLM SDK

-Un mínimo de 640 KB de memoria deberían de estar disponibles en el sistema para el desarrollo y un efectivo uso del NLM SDK. Un mínimo de 12 MB de espacio en el disco duro.

-Un ordenador 386 o 486 para poder trabajar en modo protegido. Las demás herramientas corren sobre 80x86.

-Un mínimo de 4 MB de RAM en el servidor para trabajar con las versiones 3.x de NOS y un ordenador 386 o 486.

-Para desarrollo de aplicaciones sobre Netware 4.0 un mínimo de 7 MB de memoria RAM en el servidor y un ordenador 386 o 486.

4.3.Interfaz con C para módulos NLM

Las librerías que se pueden encontrar escritas a su vez en forma de NLM y que proporcionan el interfaz con C son:

- **CLIB.NLM**: probablemente la más util e importante.
- **AIO.NLM**: proporciona funciones de entrada/salida asíncronas.
- **DSAPI.NLM**: contiene las funciones necesarias para ver, modificar, y manipular la base de información de directorio Netware (DIB) para la versión 4.0.
- **MATHLIB.NLM**: contiene funciones matemáticas.
- **NWSNUT.NLM**: contiene funciones de pantalla para la consola del servidor.

4.4.Categorías generales de las funciones CLIB

A continuación se describen de manera general y brevemente las categorías en las que se clasifican las funciones CLIB:

➪ ***4.4.1.Número de conexión y servicios de gestión de tareas:*** modo de identificación, por parte del servidor Netware, de cada NLM y de cada tarea dentro de cada NLM. De esta manera cada NLM que solicita un servicio puede ser identificado de manera unívoca.

➪ ***4.4.2.Servicios API para librerías:*** permiten crear al programador librerías propias.

➪ ***4.4.3.Servicios avanzados:*** gestión de arrays dinámicos, de múltiples procesos ligeros, de recursos de clientes, de asignación de memoria, y de asociación de atributos a ficheros.

➪ ***4.4.4.Servicios de auditoría:*** las funciones englobadas en este grupo permiten la creación y análisis posterior de una secuencia de eventos que pueden suceder en un servidor Netware.

➪ ***4.4.5.Servicios del Bindery:*** servicios de enlace para acceder a la base de datos de identificación de objetos. Estos servicios son usados por las versiones anteriores a Netware 4.0.

➪ ***4.4.6.Servicios de capacitación:*** permite a los NLM mostrar sus mensajes de salida en distintos idiomas.

➪ ***4.4.7.Servicios de comunicación:*** funciones necesarias para el establecimiento y

mantenimiento de comunicaciones tanto de tipo cliente/servidor como las que se realizan entre módulos de la misma categoría.

⇨ ***4.4.8.Servicios de conexión:*** permiten obtener información sobre las conexiones estándar de los clientes.

⇨ ***4.4.9.Servicios de contabilidad:*** estas funciones permiten obtener información sobre la cantidad de recursos del servidor Netware (almacenamiento en disco, acceso a disco, controladores de LAN, CPU y memoria) que usan los NLM o los clientes en las estaciones de trabajo.

⇨ ***4.4.10.Servicios directos del sistema de ficheros:*** permiten el acceso directo al sistema de ficheros. De este modo pueden evitarse los sistemas de caché del disco y los sistemas de seguimiento de transacciones.

⇨ ***4.4.11.Servicios de entrada/salida del sistema operativo:*** realizan funciones de entrada/salida que no requieren buffer, cómo la apertura de ficheros o la colocación del puntero de un fichero. Se utilizan normalmente en combinación con funciones de entrada/salida tipo stream.

⇨ ***4.4.12.Servicios de entrada/salida de tipo stream:*** este término (stream) se utiliza para designar a un fichero que ha sido abierto con el propósito de enviarle información o de recibir información del mismo. Estas funciones se utilizan en combinación con los servicios de entrada/salida del sistema operativo. Nunca debe de confundirse la entrada/salida tipo stream con los STREAMS del sistema operativo UNIX.

⇨ ***4.4.13.Servicios del interfaz de usuario Netware con módulos NLM:*** proporcionan un interfaz de usuario para la salida de consola por pantalla, de formato similar al que utilizan las utilidades Netware.

➪ ***4.4.14.Servicios de mensajes:*** tareas de envío de mensajes de difusión (broadcast) y envío de mensajes de consola a la pantalla de un cliente Netware.

➪ ***4.4.15.Servicios de migración de datos:*** ofrecen la posibilidad de cambiar de sitio los datos de un fichero manteniendo al mismo tiempo intacta y activa la información de almacenaje e identificación del fichero.

➪ ***4.4.16.Servicios de la partición del DOS :*** el servidor Netware puede dedicar una parte del disco a una partición del sistema operativo DOS. A esta partición se puede acceder utilizando estas funciones. Estas funciones además pueden acceder a un disco del DOS propiamente dicho.

➪ ***4.4.17.Servicios de procesos ligeros:*** se utilizan para crear y utilizar procesos ligeros dentro de un NLM.

➪ ***4.4.18.Servicios del protocolo de anuncio de servicios (SAP):*** utilizadas para anunciar o hacer públicos los servicios proporcionados por los NLM de un servidor.

➪ ***4.4.19.Servicios del protocolo de gestión de ficheros AppleTalk (AFP):*** estas funciones hacen posible el acceso y manipulación de ficheros generados por un ordenador Apple. Para más información remitirse a los manuales de las funciones CLIB.

➪ ***4.4.20.Servicios de seguimiento del sistema de ficheros:*** permiten la vigilancia de la actividad del sistema de ficheros.

➪ ***4.4.21.Servicios de sincronización:*** dan la posibilidad de bloquear un fichero para el acceso exclusivo, así como la posibilidad de controlar el flujo de los accesos a ficheros mediante semáforos.

➪ ***4.4.22.Servicios del sistema de ficheros:*** realizan operaciones específicas con ficheros tales como recuperación y purgado.

➪ ***4.4.23.Servicios del sistema de gestión de colas:*** dan la posibilidad del uso de cualquier tipo de colas por parte del programador (p.e. colas de impresión).

➪ ***4.4.24.Servicios del sistema de seguimiento de transacciones:*** dan la posibilidad de seguir los pasos de las modificaciones de datos en el sistema de ficheros. Además, este sistema de seguimiento de transacciones anula automáticamente las transacciones que no se han realizado en su totalidad. Después de la anulación, el sistema queda en el mismo estado que antes del inicio de la transacción.

➪ ***4.4.25.Servicios de STREAMS de UNIX:*** Netware ofrece una implementación de los STREAMS de Unix. Estos servicios proporcionan las funciones necesarias para la transmisión de datos entre un controlador de dispositivo y una aplicación en el entorno de un servidor Netware o de un NLM. Estas funciones también permiten la colocación de módulos software en el camino de transferencia de datos, para la realización automática de operaciones específicas sobre los mismos.

➪ ***4.4.26.Servicios de utilización de pantalla:*** realizan las tareas relacionadas con la salida de consola de un NLM en la pantalla del servidor Netware.

➪ ***4.4.27.Servicios de transporte:*** estas funciones proporcionan los servicios del protocolo de transporte de Netware para la comunicación a través de la red. Estos servicios comprenden:

-Interfaz de la capa de transporte (TLI).
-Protocolo de intercambio de paquetes en redes múltiples (IPX).
-Protocolo de intercambio de paquetes en secuencia (SPX y SPX II).

-Protocolo de control de transmisión (TCP).

-Protocolo de internet (IP).

-Sockets de Unix.

TEMA 5
INTERFASES DE COMUNICACIONES

5.INTERFASES DE COMUNICACIONES

5.1.TLI

TLI es un API de AT&T que ha adquirido un gran uso en el mundo de las comunicaciones. Es sencillo tanto en cuanto a su aprendizaje como a su utilización si lo comparamos con la programación directa sobre IPX o SPX y tiene la ventaja de su gran facilidad de migración entre los diferentes protocolos.

Este interface porporciona un estandar por el cual podemos acceder directamente a los servicios de transporte.

Para situar TLI en perspectiva , una pequeña introducción sobre el modelo de referencia de Interconexión de Sistemas Abiertos (OSI), de ISO, será útil para una mejor comprensión del tema que nos abarca. Las divisiones del modelo de referecia de trabajo en red (networking) funciona en 7 niveles, descritos brevemente en la tabla que aparece a continuación:

NIVEL OSI	DESCRIPCIÓN
Nivel 7	El nivel de ***Aplicación*** trabaja como ventana entre los procesos de las aplicaciones correspondientes que están intercambiando información.

Nivel 6	El nivel de ***Presentación*** dirige la representación de la información que las entidades del nivel de aplicación hacen referencia en su comunicación.
Nivel 5	El nivel de ***Sesión*** proporciona los servicios que las entidades del nivel de presentación necesitan para organizar y sincronizar su diálogo y manejar su intercambio de datos.
Nivel 4	El nivel de ***Transporte*** porporciona servicios de transferencia de datos transparente entre las entidades del nivel de sesión.
Nivel 3	El nivel de ***Red*** dirige la operación de la red. En particular, es el responsable del routing (enrutar) y de dirigir el intercambio de datos entre las entidades del nivel de transporte y la red.
Nivel 2	El nivel de ***Enlace*** activa el intercambio de datos entre las entidades del nivel de red. Detecta y corrige cualquier error que pueda ocurrir en la transmisión a nivel físico.
Nivel 1	El nivel ***Físico*** es el responsable de la transmisión en bruto de los datos sobre el medio de comunicación.

Un principio básico del modelo de referencia OSI es que cada nivel proporciona servicios que necesita el nivel inmediatamente superior, liberando a éste de determinar como se proporcionan dichos servicios. Este enfoque simplifica el diseño de cada nivel.

El nivel de transporte es importante porque proporciona el servicio básico, seguro, de transferir los datos (punto a punto) que necesitan las aplicaciones de los protocolos de nivel superior. Haciendo ésto, este nivel oculta la topología y caraterísticas de la red, a sus usuarios.

El nivel de transporte define un conjunto de servicios, comunes para niveles de muchas series de protocolos, incluyendo los siguientes:

- Protocolos ISO
- Protocolo de Control de Transmisiones/Protocolo de trabajo en red (TCP/IP)
- Xerox* Network Systemns* (XNS*)
- Systems Network Architecture* (SNA*)
- Netware's Internetwork Packet Exchange(tm)/Sequenced Packet Exchange (IPX(tm)/SPX(tm))

Uno de los objetivos del TLI es conseguir una independencia del medio para las aplicaciones de red y los protocolos de niveles superiores.

Netware TLI se rige por la especificación 1988XTI con dos mínimas excepciones: t_alloc y la estructura de nombres TLI de UNIX(R).

TLI se implementa como una librería de usuario usando el STREAM(tm), mecanismo de entrada/salida. Por tanto muchos servicios disponibles en las aplicaciones de STREAM(tm) están disponibles para usuarios de TLI.

El proveedor de transporte es la entidad que provee los servicios de TLI, y el usuario de transporte es la entidad que requiere estos servicios. Un ejemplo de un proveedor de transporte es el protocolo de tranporte ISO, mientras que un usuario de transporte puede ser una aplicación NLM,o una aplicación de red.

El usuario de transporte accede al servicio del proveedor de transporte por medio de la apropiada petición de servicios. Un ejemplo podría ser una petición para transferir datos sobre una conexión. De forma similar, el proveedor de transporte notifica al usuario de algunos eventos , como puede ser la llegada de datos en una conexión .

Las funciones de TLI soportan los servicios de TLI para procesos de usuario. Estas funciones permiten al usuario hacer peticiones al proveedor y procesar los eventos que van llegando.

El interfaz del protocole de transporte (TLI) proporciona dos modos o tipos de servicio:

- Modo de Conexión.
- Modo de No-Conexión.

El servicio en ***Modo de Conexión*** está orientado a circuito y permite la transmisión de datos sobre una conexión establecida. Este servicio está pensado para aplicaciones que requieran una relativa larga vida.

El servicio en ***Modo de No-Conexión***, en contraste, está orientado al mensaje y soporta transferencia de datos en unidades. Este servicio requiere sólo de la existencia anterior de una asociación entre los usuarios involucrados, lo cual determina la característica de los datos a transmitir.

Toda la información que hace falta para distribuir una unidad de datos (por ejemplo,la dirección de destino) se presenta al proveedor de transporte junto con los datos a ser transmitidos, en un acceso al servicio (el cual no se necesita relacionar con algún otro acceso al servicio).

⇨ Servicio en ***Modo de No-Conexión,*** recomendado para aplicaciones que :

- Involucran interacciones de terminos cortos de pedida/respuesta.
- Presentan un alto nivel de redundancia.
- Son reconfigurables dinámicamente.
- No requieren garantía en la distribución de los datos secuencialmente.

⇨ Servicio en ***Modo Conexión:***

-El servicio de transporte en modo conexión se caracteriza por 4 fases:

⇨ Gestión local.
⇨ Establecimiento de conexión.
⇨ Transferencia de datos.
⇨ Desconexión.

SERVICIO EN MODO CONEXIÓN

⧫ ***Gestión local:***

Esta fase define operaciones locales entre un usuario de transporte y un proveedor de transporte. Por ejemplo, un usuario debe establecer un canal de comunicación con el proveedor de transporte.Cada canal entre un usuario de transporte y un proveedor de

transporte es un único punto final (endpoint) de comunicación llamado el punto final de transporte o extremo de conexión de transporte.

La función t_open permite a un usuario elegir y establecer un proveedor de transporte en particular para suministrar el servicio en modo conexión y establecer el punto final de trasporte.

Otra labor local necesaria para cada usuario es establecer una identidad con el proveedor de transporte. Cada usuario se identifica por una dirección de transporte. Más exactamente una dirección de transporte se asocia con cada punto final de transporte. En el servicio Modo Conexión, un usuario pide una conexión a otro usuario mediante la especificación de la dirección de usuario.

La estructura de la dirección de transporte se define por el espacio de dirección del el proveedor de transporte. Una dirección puede ser tan simple como una cadena aleatoria de caracteres, o tan compleja como un patrón de bit codificados que especifican toda la información necesaria para enrutar los datos a través de la red.

Cada proveedor de transporte define su propio mecanismo para la identificación de usuarios. Las direcciones se pueden asignar a cada extremo de conexión de transporte mediante la funcion t_bind.

Además de las funciones t_open y t_bind, hay varias funciones disponibles para apoyar las operaciones locales, entre las que se encuentran : t_alloc,t_close,t_error,..etc.

- ***Establecimiento de conexión:***

La fase de establecimiento de una conexión permite a dos usuarios crear una conexión o un circuito virtual, entre ellos.

Esta fase se ilustra mediante un parentesco cliente-servidor entre dos usuarios de transporte. Un usuario, el servidor,típicamente anuncia algunos servicios a un grupo de usuarios y entonces queda a la espera de su petición por parte de ellos. Si un cliente requiere el servicio, intenta conectar por él mismo al servidor, usando la dirección de transporte anunciada del sevidor.

La funcion t_connect inicia la petición de conexión. Un argumento para t_connect, la dirección de transporte que identifica a que servidor quiere acceder el cliente. Al servidor se le notifica cada petición de conexión que le llega, usando t_listen, y éste puede llamar a t_accept para aceptar la petición del cliente de acceder al servicio. Si se accepta la petición, se establece la conexión de transporte.

- ***Transferencia de datos:***

La fase de transferencia de datos permite a los usuarios transferir datos en ambas direcciones sobre una conexión establecida. Dos funciones, t_snd y t_rcv,envían y reciben datos sobre esta conexión.

Se garantiza que todos los datos enviados por un usario se distribuirán al otro usuario en el otro punto de la conexión, en el orden en que fueron enviados.

En los diagramas que aparecen a continuación se puede tener una visión esquemática de las fases de gestión local y establecimiento de la conexión.

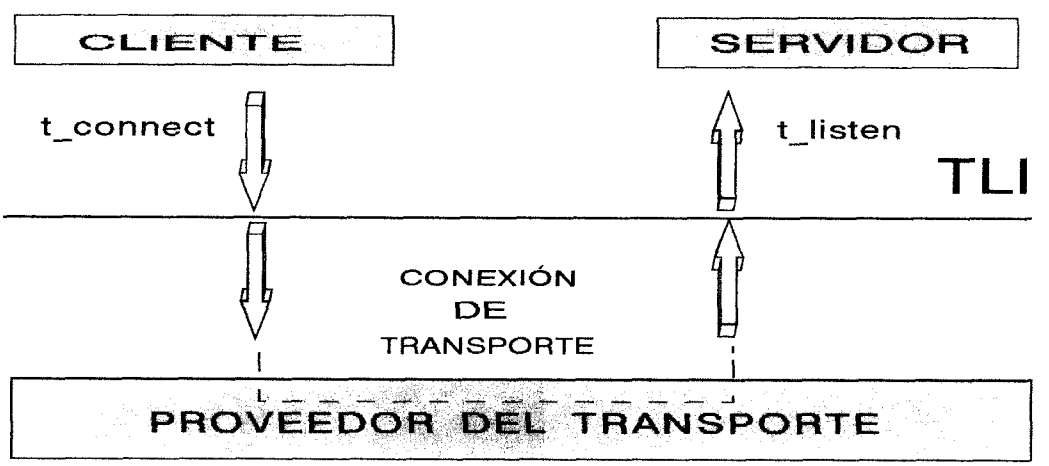
FASES PARA EL ESTABLECIMIENTO DE UNA CONEXIÓN DE TRANSPORTE (Modo Conexión)

⇨ Abrir y enlazar extremos de conexión

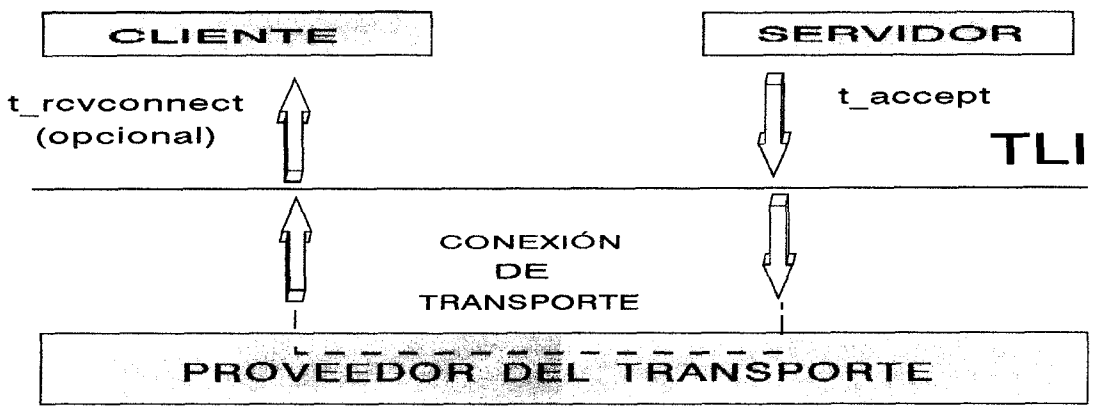
CLIENTE		SERVIDOR	
t_open	t_bind	t_open	t_bind

PROVEEDOR DEL TRANSPORTE

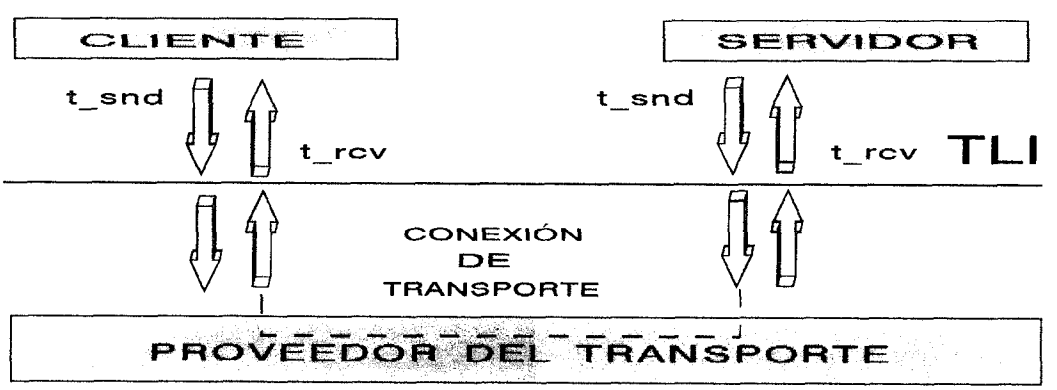
⇨ Establecimiento de la conexión



Establecimiento de la conexión



Transferencia de datos



Desconexión:

La fase de desconexión permite romper una conexión establecida. Cuando se decide que una conversación debería terminar, se puede pedir al proveedor que libere la conexión de transporte.

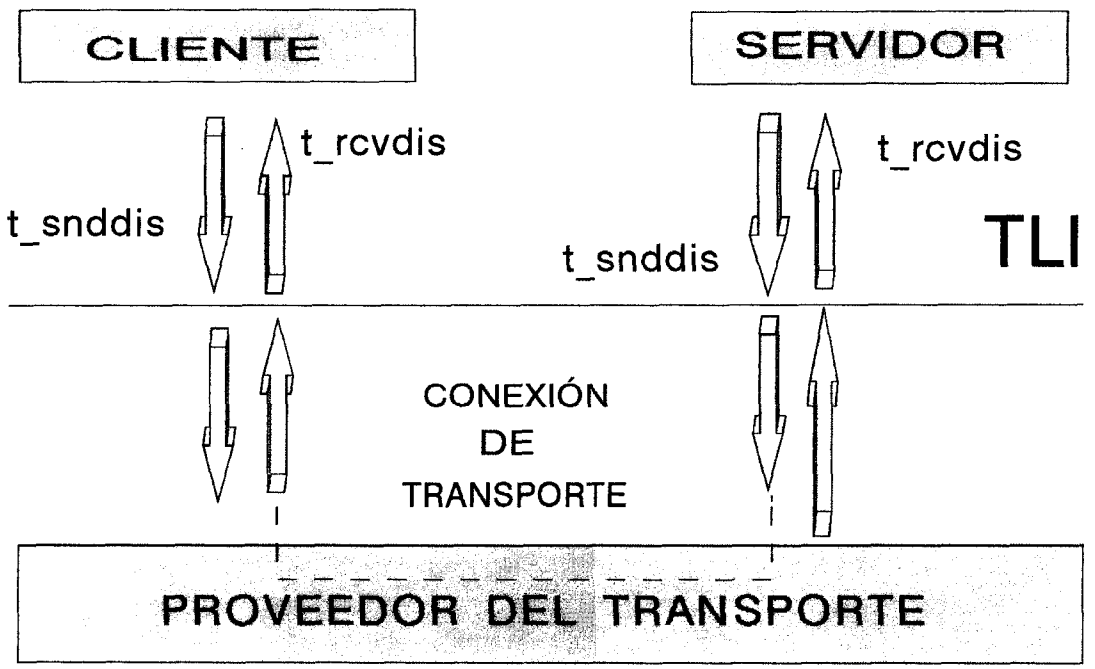
TLI soporta dos tipos de desconexión, abortiva y ordenada.

La desconexión abortiva o abrupta dirige al proveedor de transporte a liberar la conexión inmediatamente. Cualquier dato que todavía no haya llegado al otro usuario de transporte puede ser descartado por el proveedor de transporte. La función t_snddis inicia esta desconexión y t_rcvdis procesa la indicación de desconexión que nos llega.

Todos los proveedores de transporte deben soportar el procedimiento de liberación abortiva. Algunos proveedores de transporte podrían también soportar una liberación ordenada, facilitando así que los usuarios conectados terminen la comunicación elegantemente sin perdida de datos. Las funciones t_sndrel y t_rcvrel soportan esta habilidad.

A continuación puede observarse a modo esquemático esta fase.

Desconexión



SERVICIO EN MODO NO-CONEXIÓN

El servicio de transporte en modo de no conexión se caracteriza por dos fases:

- Mantenimiento o manejo local.
- Transferencia de datos.

La fase de mantenimiento local define las mismas operaciones locales descritas previamente para el servicio en modo conexión.

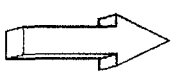
La fase de transferencia de datos permite a un usuario transferir unidades de datos (a veces llamados datagramas) a otro usuario especificado. A cada unidad de datos le debe acompañar la dirección de transporte del usuario destino.

Dos funciones, t_sndudata y t_rcvudata, soportan esta facilidad de transferencia de datos.

Las diferentes fases se reprentan en el esquema que a continuación se presenta.

FASES PARA EL ESTABLECIMIENTO DE UNA CONEXIÓN DE TRANSPORTE

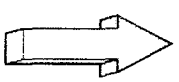
(Modo de No-Conexión)



Abrir y enlazar extremos de conexión

CLIENTE		SERVIDOR		
t_open	t_bind	t_open	t_bind	TLI

PROVEEDOR DEL TRANSPORTE



Transferencia de datos

CLIENTE		SERVIDOR		
t_snddata	t_rcvdata	t_snddata	t_rcvdata	TLI

PROVEEDOR DEL TRANSPORTE

5.1.1.Interfaz del protocolo de transporte.

La sencillez del interfaz del protocolo de transporte hace que el proceso de desarrollo de aplicaciones sea mucho más rápido, ventaja que supera bastante la pequeña ganancia de rendimiento que proporcionan IPX y SPX a costa de un mayor esfuerzo en programación.

El método a seguir para llevar a cabo una comunicación TLI es la siguiente:

1.Abrir un extremo de conexión TLI para IPX, SPX o TCP (apartado 5.3.1.4.1).
2.Asociar ese extremo a una conexión (apartado 5.3.1.4.3).
3.Establecer una conexión si se utiliza SPX o TCP 5.3.2.1.1 y 5.3.2.1.2).
4.Envío y recepción de datos (apartados 5.3.2.2.1 y 5.3.2.2.2).
5.Desconexión si se utiliza SPX o TCP (apartados 5.3.2.3.1, 5.3.3.2 y 5.3.3.2).

5.1.1.1.Introducción a las funciones, estructuras y datos de TLI.

Algunas de las funciones más importantes del interfaz de la capa de transporte se muestran a continuación (para un analisis más detallado ver "Novell Client Transport Protocol API for C" y "NLM Library Reference"):

t_open	⇨	Abre un extremo de conexión TLI.
t_bind	⇨	Asocia un extremo de conexión a una dirección.
t_sndudata	⇨	Envía un paquete de tipo datagrama.
t_rcvudata	⇨	Recibe un paquete de tipo datagrama.
t_connect	⇨	Inicia una solicitud de conexión.
t_listen	⇨	Espera una solicitud de conexión de t_connect.
t_accept	⇨	Envía notificación de aceptación al extremo que solicitó la conexión.
t_rcvconnect	⇨	Recibe la respuesta a una solicitud enviada con t_accept.

t_snd	⇨	Envía un paquete de información por una conexión.
t_rcv	⇨	Recibe un paquete de información por una conexión.
t_rcvdis	⇨	Desconecta el extremo receptor de una conexión.
t_snddis	⇨	Desconecta el extremo emisor de una conexión.
t_look	⇨	Devuelve el tipo de evento que generó un número de error TLI (t_errno).
t_error	⇨	Imprime un mensaje de error.
t_alloc	⇨	Guarda memoria para las estructuras de datos
t_close	⇨	Cierra un extremo de conexión abierto con t_open
t_free	⇨	Libera el espacio alojado por las estructuras con t_alloc
t_getinfo	⇨	Obtiene información sobre el proveedor de transporte
t_getstate	⇨	Devuelve el estado de un extremo de conexión
t_unbind	⇨	Desenlaza la dirección previamente enlazada con t_bind

Las estructuras de datos más relevantes están definidas en el fichero TIUSER.H y son: t_bind, t_call, t_optmgmt, t_discon, t_unitdata, t_uderr, t_info.

La estructura ***netbuf*** es un tipo genérico para los buffers del transporte de datos. Su formato es el siguiente:

struct netbuf {unsigned maxlen; unsigned len; char *buf;};

La variable ***buf*** es un puntero a un buffer que contiene datos.

La variable ***len*** indica cuantos bytes de datos son almacenados en el buffer.

La variable ***maxlen*** indica el máximo tamaño del buffer.

La estructura netbuf es una de las claves dentro del soporte del multiprotocolo por

parte del interfaz de la capa de transporte.

Algunas de estas funciones y estructuras se comentarán más adelante en profundidad.

5.1.1.1.1.Gestión de memoria

El interfaz de la capa de transporte incluye las funciones ***t_alloc*** y ***t_free*** para que dinámicamente se reparta y libere espacio de memoria para las variables de estructuras del TLI.

El TLI no requiere el uso de estas funciones, pero pueden ayudar a reducir la cantidad de trabajo.

5.1.1.2.Modos de operación (blocking y no-blocking)

La mayoría de las funciones pueden ejecutarse en ambos modos tambien llamados modo sincrono y modo asincrono.

En el modo blocking las operaciones son realizadas con control por parte del extremo de la conexión. En este modo una función no retorna hasta que se haya completado la operación.

En el modo no-blocking la función retorna tan rápidamente como le sea posible, pero en muchos casos continúa procesando la operación en el fondo.

El modo blocking se selecciona automáticamente en cuanto se abre un extremo de conexión. La función ***t_open*** abre un extremo de conexión.

El modo no-blocking se puede seleccionar por medio del flag ***O_NDELAY*** pasado en

la función ***t_open.***

Una vez abierto un extremo de conexión se puede usar la función ***t_getinfo*** para obtener información sobre el estado de la conexión.

La función ***t_rcv*** (recibe un paquete de información por una conexión) es un buen ejemplo de cómo una función opera de diferente manera bajo los dos modos. En el modo blocking, la función bloquea hasta que recibe datos del proveedor del transporte. Esto quiere decir que si se hace una llamada a la función ***t_rcv*** en este modo la aplicación perderá el control hasta que los datos lleguen al extremo de la conexión.

Para ejercer un mayor control sobre la operación es conveniente abrir el extremo de conexión en modo no-blocking. En este modo la función ***t_rcv*** recibe algún dato que está disponible y retorna. Si los datos no están disponibles la función falla y retorna ***TNODATA*** en ***t_errno.*** Para más detalles ver "Nerware Transport Protocol Reference".

5.1.1.3.Control de errores

Si durante un proceso determinado ocurre un error, una función TLI sitúa el valor de ese especifico error en una variable global, ***t_errno,*** y devuelve el valor -1 a la aplicación. Por lo tanto, para la evaluación de errores se debería testear el valor de ***t_errno.***

La variable global ***t_errno*** almacena el código de error correspondiente a la última función TLI que se ha ejecutado. Posibles valores de este código de error son ***TNODATA*** y ***TLOOK.***

TNODATA indica que a pesar de haberse ejecutado la función de recepción todavía no se han recibido datos. Se debería de volver a comprobar más adelante si los datos han

llegado.

TLOOK indica un tipo de error más grave. Que la variable global ***t_errno*** tome este valor, implica la existencia de más información sobre el evento que causó el error. Para poder disponer de esta información se utiliza la función ***t_look.*** Entre los posibles eventos (aunque hay unos cuantos más) que causan un código de error de este tipo están:

-***T_DISCONNECT:*** la conexión se ha interrumpido o bien ha sido rechazada
-***T_UDERR:*** error en datagrama enviado previamente.

-***T_LISTEN:*** el extremo de conexión recibe una petición para conexión por parte de otro extremo de conexión.
-***T_CONNECT:*** el extremo de conexión recibe una confirmación para nuestra petición de conexion.
-***T_DATA:*** el extremo de conexión recibe datos desde extremo de conexión.
-***T_EXDATA:*** el extremo de conexión recibe datos (acelerados, rápidos) desde otro extremo de conexión.
-***T_ORDREL:*** se recibe una petición para un abandono ordenado de la conexión (no es soportado por SPX).

5.1.1.4.Servicios de conexión

Con el fin de que cliente y servidor se comuniquen entre sí, éstos deben de ser capaces de averiguar, o bien conocer previamente , el número de SOCKET que utilizarán para comunicarse. El SOCKET viene a ser como un canal lógico que se establece entre cliente y servidor y sobre el cual se establecerán una o múltiples conexiones.

Los SOCKETS pueden ser de dos tipos:

-Estáticos: se describen o establecen cuando se escribe la aplicación y se incluyen en la misma. Siempre se utilizarán los mismos.
-Dinámicos: se asignan dinámicamente dependiendo de los que estén libres.

5.1.1.4.1.Abriendo un extremo de la conexión

Antes de comenzar con la transferencia de datos propiamente dicha las estaciones deberían de soportar la conexión.

La función ***t_open*** prepara para la conexión de transporte. Establece un extremo de conexión de transporte por apertura de un fichero que identifica a un particular proveedor del transporte (SPX, TCP/IP, etc) y configura flags. Si el extremo de conexión es abierto de manera satisfactoria la función retorna un file handle que identifica el extremo de la conexión.

La sintaxis de la función es la siguiente:

```
#include <tiuser.h>
#include <nwipxspx.h>
#define NWIN
int t_open (char *path, int oflag, struct t_info *info);
```

Los parámetros son:

path (entrada): puntero al pathname del fichero abrir.
oflag (entrada): especifica el juego de flags O_NDELAY y O_RDWR.
info (salida): recibe información fundamental sobre el protocolo de transporte.

La función puede devolver o retornar un "File handle", referencia inequívoca de que

se ha realizado de manera satisfactoria. Si por el contrario, devuelve el valor -1 es indicativo de que se ha producido un error. El juego de valores de ***t_errno*** nos dará información más precisa sobre el tipo de error.

Valores de t_errno:

TSYSERR: ha ocurrido un error de sistema durante la ejecución.
TBADFLAG: el flag especificado no es válido.
TBADNAME: no es válido el nombre de el proveedor de transporte especificado.

Ejemplo:
```
if ((fd=t_open("/dev/nspx",O_RDWR¦O_NDELAY, (struct t_info *)0))==-!)
{
        t_error("Error abriendo /dev/nspx");
        exit(2);
}
```

En el ejemplo anterior estamos abriendo un extremo de conexión para lectura/escritura (O_RDWR) y en modo de no bloqueo (O_NDELAY). Se trata de una conexión sobre SPX ("/dev/nspx") y no queremos recibir información sobre el protocolo de transporte (NULL).

Como la conexión es sobre SPX, se abre el fichero específico /dev/nspx, si fuera sobre IPX se abriría /dev/nipx y sobre TCP /dev/tcp.

Si el valor devuelto de la función (fd) es 0 ésta se ha realizado satisfactoriamente, si es -1 ha habido un error del que podremos obener más información leyendo el juego de valores t_errno.

La operación inversa a la función t_open la realiza la función ***t_close.***

Para más información sobre las funciones del interfaz del protocolo de transporte ver "Client Transport Protocol API for C" y "NLM Library Reference".

5.1.1.4.2.Información sobre los extremos de conexión

Se puede recibir información adicional una vez abierto el extremo de conexión mediante el uso de la función ***t_getinfo.***

Los datos referentes al extremo de conexión una vez abierto éste, están contenidos en la estructura ***t_info.***

Esta estructura tiene la forma siguiente:

> struct ***t_info*** {long addr; long options; long tsdu; long etsdu; long connect; long discon; long servtype;};

5.1.1.4.3.Asociar un extremo de conexión a una dirección

Después de abrir el extremo de conexión, hay que asignar o asociar una dirección del protocolo con ese extremo de conexión de transporte y activar la conexión de transporte. La función que se encarga de esto es ***t_bind.***

La sintaxis de esta función es la siguiente:

```
#include <tiuser.h>
#include <nwipxspx.h>
int t_bind (int fd, struct t_bind *req, struct t_bind *ret);
```

Las funciones por norma general suelen retornar el valor 0 si ésta se ha realizado de manera satisfactoria y valor -1 si ha habido algún tipo de error. Si se ha producido algún error el juego de valores de la variable ***t_errno*** suministra más información sobre las posibles causas del error.

La operación inversa es realizada por la función ***t_unbind.*** Esta función se encarga de desasignar el extremo de conexión con la dirección asignada previamente. Desactiva la conexión de transporte.

La sintaxis es:

```
#include <tiuser.h>
#include <nwipxspx.h>
int t_unbind (int fd);
```

Ejemplo:
```
if (t_bind (fd, (struct t_bind *)0, (struct t_bind *)0 ==-1)
{
        t_error("Error t_bind");
        exit(2);
}
```

```
struct t_bind {struct netbuf addr ; unsigned qlen; };
```

El valor de entrada fd identifica el extremo de conexión local del transporte a ser activada.

El valor de entrada req es usado para pedir que una dirección representada por la estructura netbuf sea enlazada para dar un extremo de conexión de transporte.

El valor de salida ret recibe la dirección que el proveedor del transporte ha enlazado, ésta puede ser diferente a la especificada por el usuario en req.

Si la petición de dirección no está disponible o si la dirección no es especificada en req (req.addr.len = 0), el proveedor del transporte puede asignar una dirección apropiada a ser enlazada y retornar esta dirección en ret.addr.

El usuario puede comparar la dirección en req y ret para determinar si el proveedor del transporte ha enlazado el extremo de conexión de transporte a una dirección diferente de la pedida. El valor de req puede ser NULL si el usuario no quiere especificar una dirección a enlazar. En este caso qlen asume que es cero y el proveedor del transporte debería de asignar una dirección al extremo de conexión de transporte.

5.1.2.Servicios del Modo de Conexión

Los servicios del modo de conexión se dividen en tres fases que no deberíamos de olvidar si queremos que la conexión se realice de manera satisfactoria.

5.1.2.1.Establecimiento de una conexión

Una vez abierto el extremo de conexión y enlazado con una dirección se puede proceder al establecimiento de la conexión. Los pasos a seguir para el establecimiento de dicha conexión requieren un modo de actuar diferente para cada uno de los extremos.

Un modo será el que llame y otro será el que escuche. Es caso típico en aplicaciones cliente/servidor que el servidor esté continuamente escuchando a clientes que intentan establecer una conexión. Sin embargo, una vez establecida la conexión, TLI no diferencia entre ninguno de los dos lados.

5.1.2.1.1.Escucha para conexión

El extremo que se encarga de escuchar usa la función ***t_listen,*** anticipándose a la petición de conexión por parte del otro extremo de conexión.

Esta función toma el extremo de conexión (abierto y enlazado con anterioridad) y la estructura ***t_call*** como entradas. Su sintaxis es:

```
int t_listen (int fh, struct t_call *call);
struct t_call { struct netbuf addr; struct netbuf opt; struct netbuf udata; int sequence;};
```

Ejemplo:
```
if (t_listen (fd, &call) == -1)
    {
        t_error("Error t_listen");
        exit(2);
    }
```

La función t_listen solamente recibe peticiones de conexión y no contesta a ellas. Se usa la función ***t_accept,*** por parte del extremo que escucha, para aceptar una petición de conexión y establecer una conexión.

La función t_listen retorna datos sobre la conexión en la estructura t_call.

5.1.2.1.2.Llamada para conexión

El extremo que llama usa la función ***t_connect*** para realizar la petición de conexión con el otro nodo. Esta función usa como entradas un extremo de conexión y un par de

estructuras t_call. Una de estas estructuras contiene datos para inicializar la conexión, la otra recibe cualquier dato relevante sobre la conexión. Si reciben el mismo nombre, una sobreescribe a la otra.

Sintaxis: int t_connect (int fd, struct t_call *sndcall, struct t_call *rcvcall);

Ejemplo:
```
if ( t_connect ( fd, &tcall, &tcall ) == -1)
    {
        t_error("t_connect ha fallado");
        exit(2);
    }
```

Como parte del establecimiento de la comunicación es necesario no sólo conocer el número de SOCKET sino también la dirección del nodo físico donde reside el programa con el que queremos comunicarnos. Algunas aplicaciones necesitan además el número de la red en el que reside el nodo destino. Si cliente y servidor residen en la misma LAN y pueden comunicarse entre sí sin la necesidad de enviar paquetes a través de un encaminador, no necesitan conocer el número de red.

La estructura sndcall contiene datos para inicializar la conexión. Entre esos datos está la dirección del destinatario. Por ejemplo, si SPX es el proveedor de transporte, entonces sndcall.addr.buf ha de apuntar a la dirección IPX del destino y sndcall.addr.len debería de dar la longitud de esa dirección.

Una dirección IPX está formada básicamente por tres campos que son:

-Número de red.
-Dirección del nodo.
-Número de Socket IPX.

Las direcciones IPX tienen 12 bytes : 4 para el número de red , 6 para el número de nodo y 2 para el número de socket. En el caso de TCP, la dirección se compone de una dirección de red, que ocupa 4 bytes en la notación estándar de la Internet, y el número de puerto , que ocupa 2 bytes.

Para cambiar de protocolo en una aplicación lo único que hay que modificar es la especificación de la longitud de la dirección y la obtención de la misma. El programa sigue siendo exactamente el mismo.

Ejemplo: formemos una dirección IPX con un Socket igual a 31 que será el que utilice la aplicación para la comunicación y que conocemos previamente.

```
.
#define  SPX_SOCKET    31
.
.
IPX_ADDR   spx_addr;
.
.
spx_addr.ipxa_socket[0] = 0;
spx_addr.ipxa_socket[1] = SPX_SOCKET;
sndcall.addr.buf = (char *)&spx_addr;
sndcall.addr.len = sizeof(spx_addr);
.
.
```

Una vez se han rellenado las estructuras convenientemente y la función t_connect se ha realizado de manera satisfactoria el otro extremo de la conexión contesta con la función ***t_accept*** .

5.1.2.2.Transferencia de datos

Una vez se ha establecido la conexión ya se pueden enviar y recibir datos. Las funciones utilizadas para esto son ***t_snd*** y ***t_rcv*** . Estas funciones pueden ejecutarse en modo sincrono o en modo asincrono (modo blocking o no-blocking) dependiendo del modo acordado cuando se estableció la conexión.

5.1.2.2.1.Recibiendo datos

Para recibir datos se usa la función ***t_rcv*** . Su sintaxis es la siguiente:

int t_rcv (int fd, char *buf, unsigned nbytes, int *flags)

El valor de entrada fd representa, al igual que en las demás funciones explicadas, el extremo de conexión local de transporte, a través del cual llegarán los datos.

La variable buf es un puntero al buffer de datos donde se situarán los datos a su llegada. El tamaño de este buffer viene determinado por la variable nbytes.

La variable flags especifica datos opcionales dentro de la transferencia. Asi si esta variable retorna (ya que es de salida, por lo tanto el valor de ésta viene determinado por la función t_snd del otro extremo) el valor T_MORE quiere decir que hay más datos, si retorna T_EXPEDITED los datos son acelerados, etc.

Si los datos han sido fragmentados la variable T_MORE debería de ser usada. De esta forma las llamadas a la función t_rcv deberían continuar hasta que el flag cambiara de valor.

5.1.2.2.2.Enviando datos

Para enviar datos se usa la función t_snd. Esta función tiene una sintaxis prácticamente igual t_rcv. Envía los datos a través de un buffer de datos apuntado por buf y cuyo tamaño viene determinado por la variable nbytes. Si los datos son fragmentados en paquetes se puede usar T_MORE en la variable flags para indicar al otro extremo de la conexión que hay más datos.

5.1.2.3.Abandono de conexión

Existen dos modos de abandono de una conexión: el modo de abandono abrupto y el abandono ordenado. Con el abandono abrupto se desecha la conexión sin preparar al otro extremo para un abandono de la conexión. En el abandono de manera ordenada cada uno de los extremos de la conexión señala al otro que está preparado para realizar el abandono de la conexión.

El abandono de manera ordenada asegura que todos los datos han sido procesados antes de que la conexión haya sido desechada. TLI requiere que todos los proveedores de transporte soporten el abandono de tipo abrupto pero no el ordenado, así SPX es un claro ejemplo de un proveedor que no soporta un abandono de tipo ordenado.

5.1.2.3.1.Abandono ordenado

Para iniciar un abandono ordenado se usa la función ***t_sndrel*** .Después de una llamada a esta función se ha de esperar el retorno de una indicación. Esta indicación retorna en la variable t_errno (vista anteriormante) como T_ORDREL.

Se pueden seguir recibiendo datos a través de la conexión hasta que la indicación de desconexión llega (T_ORDREL) entonces se usa ***t_rcvrel.*** Ya que SPX no soporta el modo de abandono ordenado, TLI no soporta estas funciones bajo estaciones DOS.

5.1.2.3.2.Abandono abrupto

El modo de abandono abrupto es solamente soportado por SPX. La función ***t_sndis*** es la encargada de realizar este tipo de abandono. Esta función rompe la conexión sin preparar al otro extremo que recibe solamente una indicación de que la conexión ha sido rota. Esta indicación se recibe en la variable t_errno como T_DISCONNECT y se puede usar la función t_look para confirmar el abandono, a continuación se llama a la función t_rcvdis para cerrar el lado de la conexión que ha quedado abierto.

Para mas información y ejemplos básicos ver "Programmer's Guide for C" y "Client Transport Protocol API for C"

5.1.3.Servicios del Modo de No-Conexión

La transferencia en este modo de conexión requiere unicamente que se abra y enlace un extremo de conexión.

5.1.3.1.Enviando datos

Para el envío de datos en este modo se utiliza la función ***t_sndudata*** que envía un datagrama. Esta función toma como entradas el extremo de conexión y una estructura del tipo ***t_unitdata.*** Esta estructura incluye la dirección de destino,opciones especificas sobre el protocolo y un mensaje.

Sintaxis: t_sndudata (int fh, struct t_unitdata *unitdata);

5.1.3.2.Recibiendo datos

Para la recepción de datos se utiliza la función ***t_rcvudata*** que recibe un datagrama. Como entradas esta función toma el extremo de conexión, una estructura ***t_unitdata*** y la variable flags.

El valor contenido por la variable flags debería de ser **T_MORE** si el buffer referenciado en la estructura ***t_unitdata*** no es demasiado grande para recibir el mensaje.En este caso se debería de seguir haciendo llamadas a la función hasta que el mensaje completo haya sido recibido.

Sintaxis: t_rcvudata (int fh, struct t_unitdata *unitdata, int *flags);

5.1.4.Transiciones de estado TLI

T_UNINIT ➧ Es el estado no inicializado de un programa TLI. En este estado el programa no ha comenzado o ha completado todas las operaciones TLI. Este es el estado inicial y final del programa.

T_UNBND ➧ En este estado el extremo de conexión está abierto pero aún no está enlazado con una dirección. La operación de apertura del extremo de conexión desplaza el programa del estado T_UNINIT al estado T_UNBND. La operación de cierre del extremo de conexión traslada el programa de este estado al estado anterior.

T_IDLE ➧ En este estado el programa ha enlazado el extremo de conexión a una dirección. Un programa utilizando el ***Modo de Conexión*** se puede mover o desplazar a los estados T_INCON o T_OUTCON. Por el contrario, en el ***Modo de No-Conexión*** puede moverse o desplazarse a T_DATAXFER, enviando o recibiendo datos. Desenlazando el extremo de conexión, el programa vuelve atrás, al estado T_UNBND.

T_OUTCON ⏵ Este es un estado del ***Modo de Conexión*** . En este estado el programa está atento al establecimiento de una conexión pero no ha recibido la confirmación del otro extremo. Desde este estado se puede desplazar al estado T_DATAXFER, al recibir y aceptar una petición de conexión. Si se recibe un rechazo del otro extremo de conexión se puede mover al estado T_IDLE.

T_INCON ⏵ Este es un estado de ***Modo de Conexión*** . En este estado el programa está escuchando para una conexión pero aún no se ha recibido una petición de conexión por parte del otro extremo. Desde este estado puede moverse al estado T_DATAXFER recibiendo y aceptando la petición de conexión. Por rechazo del otro extremo el programa puede volver al estado T_IDLE.

T_DATAXFER ⏵ En este estado el programa está recibiendo o enviando datos. Un programa en el ***Modo de Conexión*** se puede mover al estado T_OUTREL por petición de un abandono ordenado y a T_INREL por recepción de una petición para un abandono ordenado de la conexión. En el ***Modo de No-Conexión*** el programa vuelve al estado T_IDLE cuando se ha completado el envio o recepción de datos.

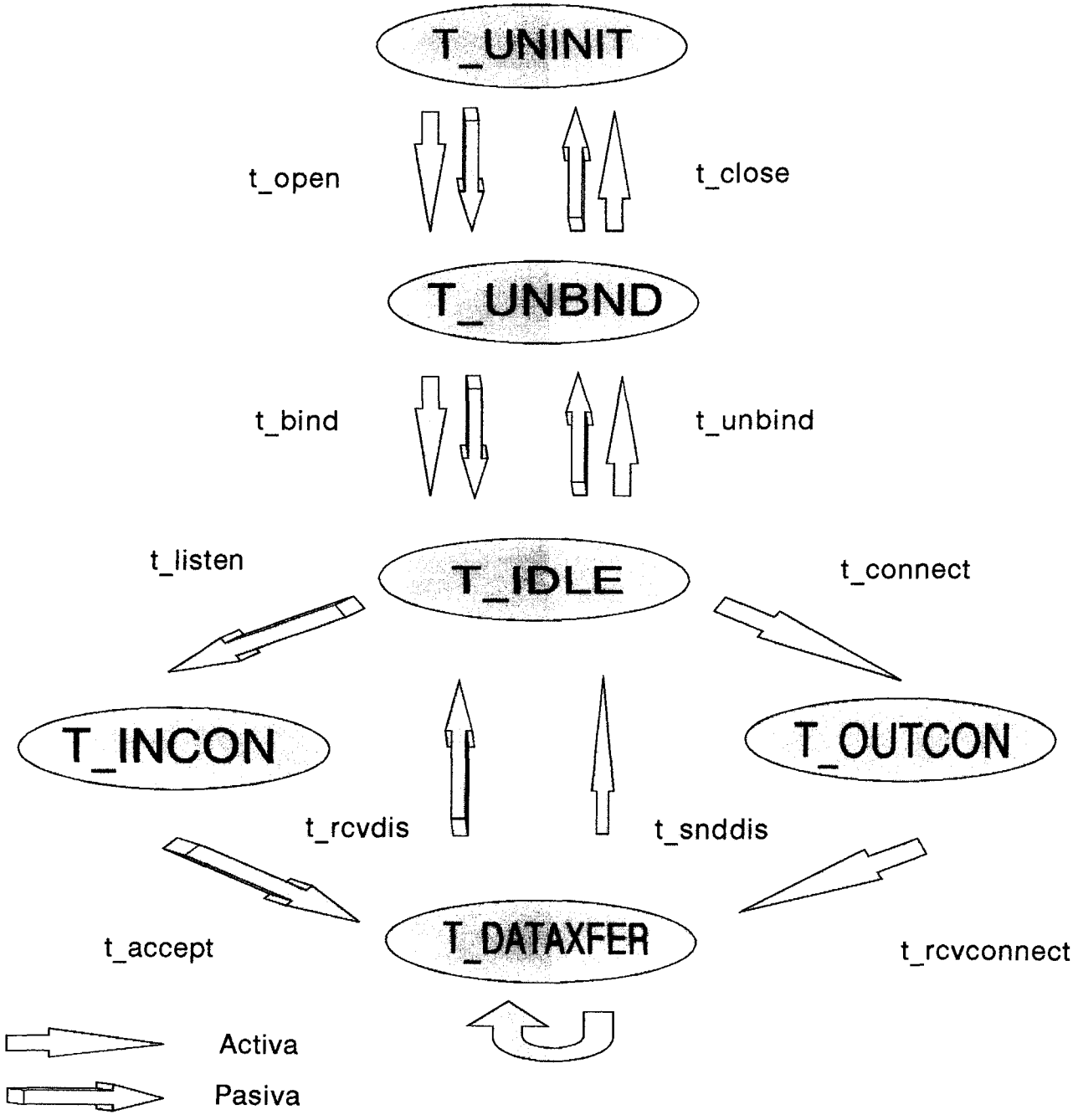
T_OUTREL ⏵ Este es un estado del ***Modo de Conexión.*** En este estado el programa ha pedido un abandono ordenado y está en espera de un respuesta. Después de recibir contestación el programa regresa al estado T_IDLE. Este estado no está soportado por el protocolo SPX.

T_INREL ⏵ Este es un estado del ***Modo de Conexión*** . En este estado el programa ha recibido por parte del otro extremo de conexión una petición para un abandono ordenado. Un vez procesada la petición de abandono ordenado, el programa regresa al estado T_IDLE. Al igual que en el estado anterior, este estado no está soportado por el protocolo SPX.

En los gráficos que aparecen a continuación se representan los diferentes estados TLI y las funciones causantes de los cambios entre estados. Se representa los dos modos de servicios que TLI suministra.

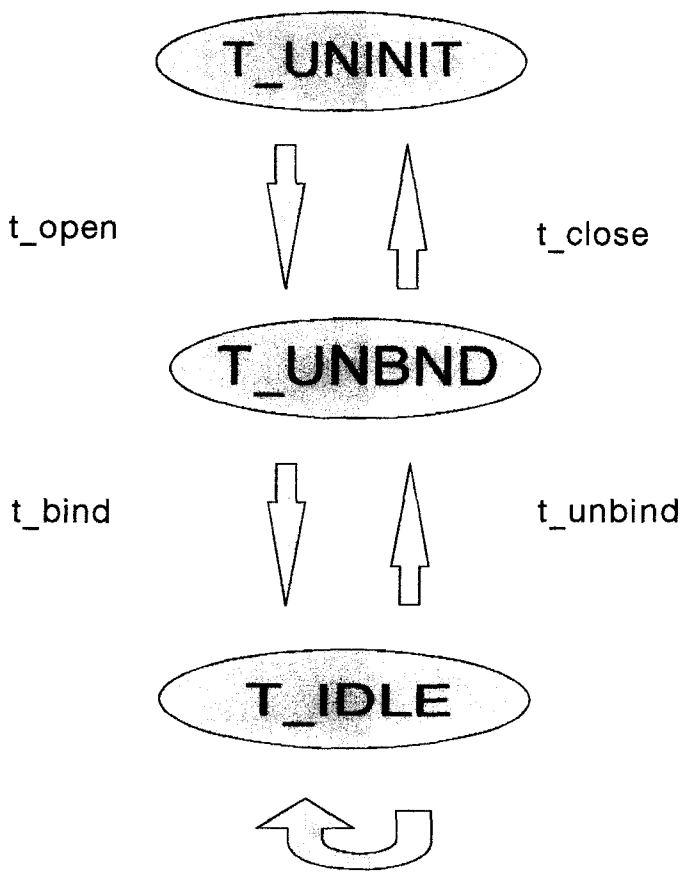
Transiciones de estado TLI

(Modo Conexión)



Transiciones de estado TLI

(Modo No-Conexión)



t_sndudata

t_rcvudata

t_rcvuderr

5.2.IPX Y SPX (NetWare)

IPX y SPX son servicios de comunicación nativos de NetWare.

IPX se usa por el software de las estaciones de trabajo NetWare para comunicarse con los servidores NetWare.

SPX añade el realce orientado a conexión para proporcionar a IPX un transporte seguro para las aplicaciones cliente. Para usar IPX o SPX , se debe cargar el software de comunicación IPX en la estación de trabajo local.

El protocolo IPX (Internetwork Packet Exchange(tm)) es un esfuerzo óptimo de servicio de distribución(datagrama). Los drivers de las tarjetas LAN intentan distribuir paquetes pero no garantizan dicha distribución.El protocolo SPX(Sequence Packet Exchange) es un servicio de distribución garantizada. En las aplicaciones que mandan paquetes SPX, formas de conexión SPX , SPX retransmite algún paquete de "desconocimiento" después de algunos intervalos apropiados de tiempo. Después de un cierto número de retransmisiones de desconocimiento,SPX asume que la aplicación de destino, no está "escuchando" y rompe la conexión.

Cada evento, como puede ser la recepción o la transmisión de un paquete, es controlado por una estructura llamada Bloque de Control de Eventos (ECB). Por ejemplo, antes que una aplicación pueda mandar un paquete, la aplicación debe preparar un ECB que (además de otras cosas) contenga la dirección en memoria del paquete. Típicamente, la aplicación llama a la función de envío de IPX o SPX con un puntero al ECB. Asímismo antes de que una aplicación pueda recibir un paquete, la aplicación debe preparar un ECB que contenga la dirección de un buffer en el cual almacenar el paquete que llega. Luego la aplicación llama a la función de recepción (escucha) de IPX o SPX con un puntero al ECB

Todos los IPX/SPX(tm) ECBs y cabeceras de paquete se deberían inicilizar a ceros binarios antes de pasárselos a IPX la primera vez, especialmente cuando se localiza memoria usando funciones que devuelven memoria no inicializada(por ejemplo, **malloc**). Pueden ocurrir resultados extraños , como conexiones SPX colgadas, si se le pasa a IPX , ECBs y/o cabeceras sin inicializar.

Para una descripción de estructuras y otras definiciones de datos mirar el Apéndice A "Servicios de datos IPX y SPX" del manual "Programmer's Guide for C".

Las funciones principales para los protocoles SPX e IPX son:

- **IpxCancelEvent/IpxCancelPacket:** cancela un ECB que está siendo usado por IpxReceive.
- **IpxCheckSocket:** determina si el socket especificado está abierto.
- **IpxCloseSocket:** cierra un Socket IPX.
- **IpxConnect:** Crea una conexión virtual para la dirección especificada.
- **IpxDisconnect:** destruye una conexión virtual para la dirección especificada.
- **IpxGetAndClearQ:** devuelve un puntero a la lista de los ECBs fuera de la cola de una cabeza de cola especificada y luego pone la cabeza de la cola a NULL.
- **IpxGetInternetworkAddress:** devuelve la dirección de la red y del nodo de la petición del NLM.
- **IpxGetLocalTarget:** rellena el campo immediateAddress en un ECB.
- **IpxGetStatistics:** devuelve estadísticas diagnósticas mantenidas por IPX (No implementadas;incluidas para compatibilidad)
- **IpxGetVersion:** devuelve la major y la menor versión y la revisión de IPX.
- **IpxOpenSocket:** abre un socket IPX.
- **IpxReceive:** inicia la recepción de un paquete IPX.
- **IpxResetStatistics:** resetea estadisticas diagnósticas mantenidas por IPX (no implementadas;incluidas por compatibilidad).

- **IpxSend:** inicia el envío de un paquete IPX.
- **SpxAbortConnection:** aborta una conexión SPX.
- **SpxCancelEvent/SpxCancelPacket:** cancela un evento SPX pendiente.
- **SpxCheckSocket:** chequea el estado de un número específico de Socket.
- **SpxCloseSocket:** cierra un Socket SPX.
- **SpxEstablishConnection:** intenta establecer una conexión SPX con un socket escuchando.
- **SpxGetConfiguration:** determina el número máximo de conexiones SPX y el número de conexiones SPX disponibles.
- **SpxGetConnectionStatus:** devuelve el estado de una conexión SPX
- **SpxGetTime:** toma un marcador de tiempo del SPX
- **SpxGetVersion:** devuelve la mayor y menor versión revisión y fecha de la revisión de SPX
- **SpxListeForConnection:** intenta recibir un paquete de Conexión Establecida y de ese modo establece una conexión SPX con un compañero remoto.
- **SpxListenForSequencdPacket:** pasa un ECB a SPX con el propósito de recibir un paquete secuencial.
- **SpxOpenSocket:** abre un socket SPX.
- **SpxSendSequencedPacket:** envía un paquete SPX.
- **SpxTerminateConnection:** termina una conexión SPX

5.2.1. Características de IPX

IPX trabaja con datagramas, paquetes independientes de datos que no requieren confirmación o secuenciamiento. Los datagramas reducen el tráfico de la red y dinamizan el funcionamiento de la red. En el caso de IPX , el tanto por ciento de los datagramas distribuidos con éxito ronda el 95%. Consecuentemente, si se utiliza IPX se necesitará desarrollar una estrategia que confirme la distribución de los datos y probablemente se tendrá que guardar los paquetes en orden.

Si se está usando IPX para accionar una simple transacción de petición/respuesta (parecido al agente cliente de NetWare) , la respuesta por sí misma puede servir como un reconocimiento. Si no se recibe respuesta en un cierto período de tiempo, se puede asumir que la distribución ha fallado y actuar de acorde con esto.

5.2.2.Características de SPX

Si se necesita saber con exactitud que los datos se están recibiendo en el orden en que se envían, se debe construir esas garantías en la aplicación o usar SPX.

SPX proporciona los paquetes secuenciados y en distribución garantizada asociado con un servicio en Modo Conexión. Esta utilidad añade un poco de sobrecarga a las comunicaciones de la red, pero viene listo para usar y no necesita de ningún trabajo adicional.

5.2.3.Información sobre el Paquete

IPX y SPX son adaptaciones de la arquitectura Xerox Network System (XNS).

IPX concuerda con el Xerox Internet work Datagram Protocol (IDP).

SPX concuerda con el Sequenced Packet Protocol (SPP).

Para usar estos protocolos se debe pensar en términos de paquetes. Un paquete es un dato formateado de una manera especial para ser transmitido de una estación de trabajo a otra. Un paquete contiene una cabecera y un mensaje:

- La cabecera contiene los datos necesarios para transmitir el paquete desde su origen hasta su destino.
- El mensaje son los datos de la aplicación.

IPX define el formato básico de la cabecera de paquete. SPX aumenta este formato con los datos que necesita para mantener la conexiones. Muchas de las cosas que a continuación se dicen para IPX también son verdad para SPX , a menos que se diga lo contrario.

Se puede mandar un mensaje usando IPX acumulándolo y poniendo el mensaje en la porción de datos de un paquete IPX, parecido a poner una carta en un sobre. La cabecera de paquete IPX debe incluir la red de destino, el nodo, y el número de puertos---la dirección a la cual se debe mandar el paquete.IPX transmite cada paquete individualmente a través de varias subredes (posiblemente por diferentes rutas para sacar provecho del tráfico más ligero de la red) hasta que el paquete llege a su destino. Así como cada paquete es una entidad individual, la ruta y la secuencia de cada uno de ellos puede variar.

Cuando llega el paquete, el emisario no recibe verificación de que el paquete ha sido distribuido con éxito. Sólamente en el caso de una verificación inherente puede estar seguro el emisario de que el paquete ha llegado.Esto es, el destino actúa sobre la información enviando una petición, por ejemplo, enviar una lista de servidores de internet, o una peti ción de imprimir algo.

La estructura de un paquete IPX es idéntica a la estructura de un paquete XNS (Xerox Netword Standard). El paquete consiste en una cabecera de 30 bytes seguida de 0 a 546 bytes de datos. El tamaño mínimo de paquete es 30 bytes (sólo la cabecera) y el máximo tamaño de paquete es 576 bytes. El contenido y estructura de la porción de datos es responsabilidad de la aplicación usando IPX y puede tomar cualquier formato.

El fichero include NWIPXSPX.H contiene definiciones para todas las estructuras C que se necesitan para cabeceras IPX

Antes de que se envíe un paquete SPX se debe establecer una conexión entre el emisor

y el receptor. SPX realiza la labor de garantizar la distribución, mediente el secuenciamiento de paquetes, detectando y corrigiendo errores y descartando los paquetes duplicados.

El paquete SPX es idéntico al IPX excepto en que tiene 12 bytes adicionales en su cabecera. Un paquete SPX consiste en una cabecera de 42 bytes seguida de 0 a 534 bytes de datos. El tamaño mínimo de paquete es 42 bytes(sólo la cabecera) y el tamaño máximo es 576 bytes. El contenido y la estructura de la parte de datos es responsabilidad absoluta de la aplicación mediante el uso de SPX, y puede tomar cualquier formato.

La cabecera de paquete SPX consiste en una cabecera de IPX (30 bytes) y 7 campos adicionales

Para una descripción de las partes definidas por los servicios IPX y SPX , mirar "IPX and SPX Services Data" en el Apéndice A de "Programmer's Guide for C".

5.2.3.1.Direcciones de Red.

Las direcciones de red de origen y destino toman la mayor parte del espacio en las cabeceras de IPX. Una dirección de red incluye lo siguiente:

- 4 Bytes de dirección de red.
- 6 Bytes de dirección de nodo.
- 2 Bytes de número de socket.

Los routers de la red cuentan con las direcciones de red para distribuir un paquete cuando el destino es una red distinta de la red de origen.

Cuando el paquete llega a la red de destino, el driver de LAN de la estación de trabajo reconoce el paquete por su dirección de nodo.

El driver de LAN maneja el paquete sobre IPX el cual usa el número se socket para distribuir los datos a un proceso en la estación de trabajo. Mediante la asiganación de un número de socket a la transmisión , IPX puede soportar multiples procesos en la estación de trabajo.

5.2.3.2.Cabecera IPX

El tamaño máximo de un paquete IPX es 576 bytes. La cabecera de un paquete IPX lo forman los primeros 30 bytes de cada paquete. (SPX requiere 42 bytes) . Lo que sobra lo puede usar la aplicación para transmitir datos. La cabecera incluye los siguientes campos:

- Checksum
- Longitud de paquete
- Control de transporte
- Tipo de paquete
- Dirección destino
- Dirección origen

La estructura IPXHeader contiene la cabecera IPX.

5.2.3.3.Cabecera SPX

La cabecera de un paquete SPX es idéntica a la cabecera de un IPX excepto que tiene 12 bytes adicionales de datos, tomando la cabecera un tamaño de 42 bytes y él reduciendo el mensaje máximo a 534 bytes. La cabecera SPX añade los siguientes campos al formato de cabecera IPX:

- Control de conexión.
- Tipo de corriente de datos.
- Identificación de conexión origen.
- Identificación de conexión destino.
- Número de secuencia.
- Número de confirmación.
- Número de localización.

Estos campos adicionales permiten a SPX el proporcionar el secuenciamiento de paquetes y la distribución garantizada. La estructura SPX contiene la cabecera SPX.

5.2.4.ECBs

Los Bloques de Control de Eventos (ECBs) son los enlaces entre las aplicaciones e IPX. Cada ECB contiene la información que IPX necesita para mandar o recibir una paquete a la medida de las necesidades de las aplicaciones. Poner a punto los ECB es la tarea más dura que se puede encuentrar cuando se programa sobre IPX o SPX.

Los ECB controlan los eventos relacionados con la transmisión y recepción de paquetes IPX y SPX. Los ECB también controlan el establecimento y la finalización de las sesiones SPX.

Los ECB pueden ser orientados a socket (socket-based) o del tipo no-socket (socketless).

Los ECB orientados a socket manejan los eventos IPX/SPX.

Los del tipo no-socket planifican los eventos si referencia a transmisiones de paquetes IPX/SPX.

Un ECB orientado a socket incluye los siguientes campos :

- Dirección de enlace.
- Dirección ESR
- Flag en uso
- Completion code
- Número de socket
- Espacio de trabajo IPX (IPX workspace)
- Espacio de trabajo de driver (driver workspace)
- Dirección inmediata
- Cuenta de fragmentos (fragment count)
- Lista de descriptor de fragmentos ECB

La estructura ECB contiene los datos de ECB.

Los ECB de no-conexión están (timed) por el planificador de eventos asíncronos (AES) y consecuentemente no requieren un socket. Los campos en el tipo de no-conexión son un subconjunto de los que se encontraron en los ECB orientados a socket:

- Dirección de enlace
- Dirección ESR
- Flag en uso
- Espacio de trabajo AES (AES workspace)

5.2.5.Gestión local

Los asuntos de gestión local de IPX y SPX incluyen: la inicialización de la libreria API (para programadores en C) , la apertura y el cierre de los sockets, el planificador de eventos asíncronos, el desarrollo de las rutinas de servicio de eventos , y el mantenimiento

de los fragmentos ECB.

5.2.5.1. Inicializando IPX y SPX

Antes de que se pueda llamar a las funciones IPX/SPX en la librería API, se tiene que inicializar medio IPX . Opcionalmente se puede inicializar SPX para asegurar la compatibilidad local con las funciones SPX.

- **Inicialización de IPX**

A las funciones IPX y SPX se accede haciendo una llamada far al interface IPX. Para obtener esta dirección se ejecuta la interrupción multiplex del DOS (DOS multiplex interrupt) . Por ejemplo , las siguientes instrucciones devolverán la dirección:

```
IPXLocation dd
mov AX,7A00h
int 2Fh
cmp AL 0FFh
jne NoIPX
mov AX,ES
mov IPXLocation,DI
mov IPXLocation + 2, AX
```

Si IPX está cargado, ésto pone el byte bajo de AX a FFh y devuelve la dirección de entrada de la llamada far en ES:DI. Si IPX no estaba cargado, la interrupción no modifica AL. Se puede hacer llamadas IPX y SPX preparando los registros apropiados y luego llamando al punto de entrada. El registro BP no se preserva en llamadas SPX.

Los programadores en C pueden llamar a IPXInitialize para chequear si IPX

está cargado . Si lo está, esta función almacena la dirección de entrada de IPX internamente para el uso de la libreria API.

➪ **Inicialización de SPX**

Antes de usar SPX, se debe chequear si se soporta SPX localmente. (algunas de las primeras versiones del software de las estraciones de trabajo NetWare, no soportaban SPX.) Para averiguar si SPX es soportado llamamos a SPXInitialize. Esta función nos devuelve el mayor y menor número de revisión y el número máximo de conexiones SPX disponibles. SPX es soportado por Netware 2.1 y versiones superiores.

5.2.5.2. Manejando Sockets.

Un socket es un valor de 2 bytes que asocia una aplicación o un proceso con la transmisión de datos. IPX y SPX requieren que se abra un socket en orden para recibir datos.

SPX también necesita un socket abierto para enviar datos. Si se están enviando datos con IPX no se necesita abrir un socket. IPX localizará un socket si no se especifica ninguno.De otra forma se puede utilizar un mismo socket para manejar las transmisiones de ambos , IPX y SPX.

➪ **Abriendo un socket**

Para abrir un socket, se llama a IPXOpenSocket. Esta función toma un número de socket y un tipo de socket como entrada. IPX seleccionará un valor de socket dinámico si se le pasa como número de socket 0000h. IPX elige sockets dinámicos entre 4000h y 4000h.

Los tipos de socket pueden ser de corta-vida o de larga-vida. Típicamente la

mayoría de las aplicaciones utilizan sockets de corta-vida.Un socket de larga vida permanece abierto después de que la aplicación haya terminado y puede ser ventajoso para programas residentes en memoria.

➪ **Cerrando un Socket**

Cuando se ha terminado con un Socket, se llama a IPXCloseSocket.

Esta función cancela los eventos asociados con el socket. Es especialmente importante que la aplicación cierre cualquier socket de larga vida que se haya abierto. Estos sockets persistirán incluso después de que la aplicación termine. IPX cierra los sockets de corta-vida con un programa de terminación.

5.2.5.3.Rutinas de Servicio de Eventos

El campo "ESR Addres" en el ECB puede contener la dirección de una rutina de eventos que una determinada aplicación desea ejecutar en respuesta a un evento IPX.El evento detonante de la rutina de servicio de eventos podria ser el envío de un paquete, la recepción de un paquete, la recurrencia de un evento IPX que se ha replanificado el mismo, o un evento planificado por determinada aplicación. Después de procesado el evento , IPX pone de nuevo el "flag en uso" del ECB a 0, entra un "completion code" quita al ECB de su lista interna y llama a la rutina de servicio de eventos. En este punto, IPX ya ha acabado y hay libertad para el manejo del ECB y cualquier dato asociado.

➪ **Interface de la Rutina de Servicio de Evento**

La rutina de servicio de evento deberia asumir las siguientes condiciones:

➪ El registro AL contiene la identidad del que llama al proceso : 00h para el AES y FFh para IPX.

- Hay un puntero al ECB asociado con la rutina de servicio de evento en el registro par ES:EI.
- El estado de los flags del procesador y todos los registros excepto SS y SP que se han guardado en la pila del usuario. La rutina de servicio de evento es la responsable de guardar SS y SP para que pueda retornar adecuadamente después de la ejecución.
- Las interrupciones están desactivadas.
- El registro DS necesita ser inicializado antes de que la rutina de servicio de eventos haga referencia a cualquier variable de aplicación.

La rutina de servicio de evento es llamada por un largo procedimiento de llamada. Consecuentemente esto debe retornar con una instrucción RETF y con las interrupciones desactivadas. No puede devolver un valor.

Para programas escritos en C, a menudo, la rutina de servicio de evento, se implementa como una funcion C llamada por un ensamblador "front end":

```
-ESRHandler proc far
mov ax,DGroup
mov ds,ax
push es
push si
call -EventServiceRoutine
retf -ESRHandler
enp
```

Si el manejador ESR no necesita pasar más que unos pocos bytes de datos a la rutina de servicio de evento, el manejador debería mantener una pila propia. En este caso,

debería restaurar el segmento de pila original y el puntero de pila antes de retornar.

⇨ Procesando Rutina de Servicio de Evento

Una rutina de servicio de evento se comporta como una rutina de servicio de interupción. IPX llama a la rutina de servicio de eventos con las interrupciones desactivadas. Consecuentemente, si se usa una rutina de servicio de Interrupción , se debería ejecutar tan rápido como fuera posible. Cuando la rutina acaba, IPX devuelve el control a la aplicación.

Una rutina de servicio de interrupción debería no hacer peticiones que el programa principal de cualquier aplicación pueda efectuar adecuadamente. Típicamente , una rutina de servicio de evento pone a sus ECB asociados en una cola y retorna.

El cuerpo principal de la aplicación controla la cola, procesa los datos, y maneja la vuelta de los ECB a IPX.

⇨ Activando Interrupciones

Una rutina de servico de interrupción puede ejecutar por largos periodos, un procedimiento que activa temporalmente, si activa las interrupciones (la rutina) o los "invokes". De cualquier forma, si IPX llama a la rutina de servicio de evento, pueden ocurrir problemas si la rutina no es reentrante y activa interrupciones.

Bajo estas circunstancias, la rutina de servicio de evento deberia tomar medidas para prevenir la recurrencia. Si, el AES llama a la rutina de servicio de interrupción , activando interrupciones prevendrá al AES de llamar de nuevo a la rutina de servicio de evento planificado durante el mismo intervalo de reloj. Estas rutinas serán retrasadas hasta el siguiente "click" de reloj.

- **Peticiones IPX y Rutinas de Servicio de Interrupción**

Una rutina de servicio de interrupción llamada por IPX puede libremente llamar a otra función IPX o AES excepto:

- IPXCloseSocket
- IPXDisconnectFromTarget

Estas funciones no deberían ser llamadas por una rutina de servicio de interrupción.

Una rutina de servicio de interrupción puede replanificarse a si misma para una ejecución retrasada, llamando a las funciones:

- IPXScheduleIPXEvent (si fue llamado por IPX)
- IPXScheduleSpecialEvent (si lo fue por AES)

En cualquier otro caso, la rutina de servicio de interrupción pasa la dirección de el procedimiento a su ECB asociado y luego ejecuta una instrucción RETF.

5.2.5.4.Fragmentos ECB

IPX ensambla un paquete del buffer o fragmentos referenciados por el ECB. El ECB almacena estos valores como un array de descriptores de fragmento junto con un contador de fragmento que indica la longitud del array.

Se puede usar tantos fragmentos como sean necesarios. El único requisito es que los primeros 30 bytes del primer buffer deben estar dedicados a la cabecera IPX (42 bytes para la cabecera SPX) y la longitud total del paquete no puede exceder de 576 bytes.

En la mayoria de las situaciones 2 buffers está bien para un mantenimiento de paquetes. El primer buffer , usalmente, contiene la cabecera IPX o SPX y el segundo buffer contiene los datos del mensaje.

5.2.6.Servicio en Modo No-Conexión : IPX

Una vez que se haya inicializado IPX localmente, se está preparado para mandar y recibir paquetes IPX.

➪ Manteniendo ECBs

El primer paso en el intercambio de paquetes es adecuar un "estanque" de ECBs para recibir mensajes, datos entrantes y otro "estanque" para enviar datos de salida. El número de ECBs que se puede señalar depende de las necesidades de la aplicación.

➪ Señalando ECBs

Hasta que se procese el ECB y se regrese a IPX , este no puede recibir datos entrantes. Cuantos más datos procese una aplicación concreta y más buffers de entrada use más ECBs se deberían señalar. Es mejor que sobren que no que falten.

➪ Colocando ECBs en cola

Se necesita idear alguna estrategia para colocar en cola y escanear los ECBs, como una lista enlazada. La rutina de servicio de eventos puede procesar los ECBs recibidos. Cuando IPX usa el ECB, invoca a la rutina de servicio de evento, la cual al tiempo puede situar al ECB dentro de una cola. La aplicación puede chequear la cola periódicamente y

procesar los ECB.

⇨ Inicializando ECBs

Para señalar y preparar los ECBs, varios campos necesitan inicializarse. Para recibir ECBs , se han de rellenar los siguientes campos:

- ⇨ Dirección de la rutina de servicio de evento.
- ⇨ Número de socket.
- ⇨ Campos de lista de fragmentos.

Para enviar ECBs se rellenan los siguientes campos:

- ⇨ Dirección de la rutina de servicio de evento.
- ⇨ Número de socket.
- ⇨ Dirección Inmediata.
- ⇨ Tipo de paquete.
- ⇨ Dirección de destino.
- ⇨ Campos de lista de fragmentos.

El campo de dirección inmediata puede ser puesto mediante la llamada a IPXGetLocalTarget. Si no se está usando una rutina de servicio de evento se debe de poner a NULL el campo de dirección de rutina de servicio de evento.

⇨ Manteniendo ECBs

Recibir y enviar ECBs se hace generalmente por separado.

La recepción de ECBs recae sobre IPX llamando a IPXListenForPacket. Esto

lo que hace es liberar a la aplicación de recibir ECBs hasta el momento en que se requiere su atención.

El envío de ECBs debe ser hecho por la aplicación. Una vez que IPX usa y envía un ECB, acaba con él, y es responsabilidad de la aplicación el guardar el ECB e n circulación. Se puede poner ECBs listos para enviar en una lista libre o se puede escanear el "estanque" entero de ECBs libres, cada vez que se necesite uno.

⇨ Escuchar por paquetes (esperar por paquetes)

Para escuchar por paquetes se llama a IPXListenForPacket. Esta función toma un ECB praparado para recibir paquetes como entrada.

Mientras IPX está usando un ECB , el flag en uso es de valor distinto de cero. Una vez que un ECB haya recibido un paquete, el flag en uso se pone a cero y el valor apropiado se mete en el campo "completion code". Cuando IPX acaba con la recepción de ECB , éste invoca a la rutina de servicio de evento, si se ha asignado alguna.

⇨ Enviando Paquetes

Para mandar un paquete,se realiza una llamada a IPXSendPacket. Como entrada, esta función toma un envío ECB cuya cabecera IPX especifica el nodo destino y el socket.

Para emitir a todas las estaciones , el campo destino de la cabecera se debe tener con el valor FFh. Si una aplicación específica está emitiendo a la red en la que reside, la dirección immediata del ECB debe estar también con el valor FFh.

Mientras IPX intenta mandar el paquete, pone el flag en uso del ECB a FFh.

Después, IPX pone este campo a cero y mete un valor apropiado en el campo "completion code" .Un valor de "completion code" igual a FEh indica que el paquete no es distribuible. Esto puede ser porque IPX no encuentre un brigde a la red de destino , o por que la dirección del nodo destino no existe.

⇨ Liberando recursos

Cuando se ha terminado la comunicación con un nodo en particular , se llama a IPXDisconnectFromTarget. Esta función permite a los drivers de la red, liberar recursos a nivel de protocolo de red.

⇨ Planificando Eventos Asíncronos:

El software IPX de la estación de trabajo, incluye un Planificador de Eventos Asíncronos (AES) . Este dispositivo permite planificar eventos IPX para que tengan lugar después de un intervalo de tiempo especificado. Para planificar un evento asíncrono, se llama a IPXScheduleIPXEvent. Como entrada, esta función toma, la cantidad de tiempo a retrasar el evento y un ECB que controlará la ejecución del evento.

El ECB debería incluir el número de socket donde ocurrirá el evento, y la rutina de servicio de evento que recibirá el control cuando el periodo de tiempo expire. La unidad de tiempo se expresa en "ticks" de reloj (de 0 a 65,353).

Se puede planificar un evento desde una hora en adelante. Para cancelar un evento programado, se llama a IPXCancelEvent.

5.2.7.Servicio en Modo-Conexión: SPX:

Comunicarse con SPX requiere más o menos la misma preparación que con IPX. Se

tiene que abrir un socket, y preparar para enviar y recibir ECBs, tal y como se haría con IPX.

⇨ Estableciendo la conexion

La explicación de como preparar los ECBs para IPX es igual que para SPX . SPX necesita un ECB adicional para establecer la conexión.

Generalmente, se deberá tener un ECB separado del ECB inicial para este propósito. Este no requiere rutina de servicio de eventos y no referencia más datos que la cabecera SPX.

Para la prepararación de la conexión , un nodo actúa como parte activa y el otro como parte pasiva. Una vez que la conexión se ha establecido, SPX no hace distinción entre el que llama y el que escucha.

El nodo que llama utliza la función SPXEstablishConnection. Esta función toma el ECB inicial, una cuenta de reintento, y un flag "perro guardian" como entradas. La cuenta de reintento especifica cuantas veces SPX continúa mandando paquetes de reconocimiento.

El flag "perro guardian" permite al nodo que llama activar un proceso, "perro guardian", local para vigilar la conexión SPX después de que ésta se haya establecido. (Si se tiene activado el proceso "perro guardian" se debería señalar un ECB de recibo adicional). Si la petición es correcta, el que llamada recibe un Identificador de conexión SPX que identifica la conexión.

En el otro punto de la conexión , el nodo que escucha llama a la función SPXListenForConnection. Esta función toma parámetros similares a SPXEstablisConnection. SPXListenForConnection se puede cancelar con IPXCancelEvent.

➪ **Transmitiendo datos:**

Existen dos funciones usadas para intercambiar datos una vez que se haya establecido la conexión.:

➪ SPXListenForSequencedPacket.

➪ SPXSendSequencedPacket.

Como entrada, ambas funciones toman un ECB para gobernar el evento asociado. Una aplicación debería mantener los ECBs de una manera similar a la descrita en las operaciones para IPX.

Para recibir información de estado sobre la conexión SPX, se llama a la función SPXGetConnectionStatus.

➪ **Liberando una conexión SPX**

Existen dos funciones que cierran una conexión SPX:

➪ SPXTerminateConnection informa al nodo compañero que la conexión se está cerrando. Abandono abrupto.

➪ SPXAbortConnection no hace intento de alertar al compañero de que se está cerrando la conexión. Abandono ordenado.

El método preferible en la mayoría de situaciones es SPXTerminateConnection.

5.2.8. Checksums Paquetes

Las funciones de checksum de paquetes se han añadido recientemente al cliente API de NetWare. Estas funciones generan y verifican checksums para transmisiones de paquetes, proporcionando un alto nivel de integridad de los datos a través de la red. Los cheksum de paquete son una utilidad opcional que debe ser soportada por ambos puntos o extremos de la transmisión. La petición de checksum se implementa sólamente bajo el protocolo IPX de DOS (IPXODI).

➪ Negociando Checksums

Los nodos implicados en la comunicación deben negociar el cómo usar los checksums. Estas negociaciones se deberían llevar a cabo utilizando un protocolo orientado a conexión, como SPX o NetBIOS. Para determinar si se soporta el checksumming en la estación de trabajo local, se llama a la función GetIPXInformation.

➪ Transmisión de checksums y paquetes:

El IPX de DOS soporta dos funciones para enviar paquetes:

- ➪ IPXFastSend
- ➪ IPXSendPacket

IPXFastSend emite pasando el error normal de checking hecho por IPXSendPacket. IPXFastSend deja la cabecera de IPX sin modificar, asumiendo que el llamador ya ha incluido todos los datos necesarios.

En contraste, IPXSendPacket inicializa varios campos de cabeceras IPX antes de mandar el paquete, incluyendo el campo checksum, el cual está siempre puesto a 0FFFFh. En otras palabras, esta función siempre sobreescribe cualquier valor actual de checksum. Para permitir el no modificar los checksums, usa IPXSendWithChecksum.

⇨ Checksumming y Ethernet 802.3

Los valores de cheksums se almacenan en el campo checksum de la cabecera IPX. un valor de 0FFFFh en el campo checksum identifica los paquetes ETHERNET_802.3 a los drivers de LAN. En consecuencia, las aplicaciones no pueden usar checksums IPX cuando transmiten paquetes a través de una red que use ETHERNET_802.3. Específicamente, las aplicaciones no deberían usar checksumming si el IPX de dos está enlazado a una tarjeta de tipo ETHERNET_802.3 .

5.2.9.Sockets Look Ahead

La función IPXOpenLookAheadSocket se ha incluido recientemente al cliente API de NetWare. Esta función deja a una aplicación abrir un socket look ahead y registrar con IPX un ReceiveLookAheadHandler.

El "manejador" de recepción Look Ahead:

IPX llama a ReceiveLookAheadHandler cuando recibe paquetes en el puerto asociado así que el manejador puede prevenir la llegada inmediata de datos. Si la aplicación desea los paquetes, el manejador suple IPX con las direcciones de los buffers de la aplicación para recibir los datos. Usando esta ventaja , la aplicación puede estar dispuesta a recibir una ráfaga de paquetes diréctamente en sus buffers.

Normalmente el buffer de recepción de ECB sirve como almacén temporal para los datos que van llegando. Antes de que el ECB entre en juego otra vez de nuevo, la aplicación debe copiar los datos, fuera de estos buffers, en un area donde pueda operar con los datos. Mediante el uso del socket "look ahead" una aplicación ya no se tendrá que molestar más, en mantener estos "estanques" de buffers intermedios.

Para entender como trabaja ReceiveLookAheadHandler, considere como se manejan los paquetes que van llegando. Cuando una estación de trabajo recibe un paquete, el driver de LAN da una porción de él al Link Support Layer (LSL). A partir de este dato el LSL determina a que capa de red va destinado el paquete. Si el destino resulta ser IPX , LSL llama al look ahead handler interno de IPX.

El handler interno de IPX determina el estado de el socket de destino. Si el socket de destino es un socket normal , el handler pasa un ECB al LSL el cual a su turno, se lo pasa al driver de LAN. El driver de LAN rellena el ECB con referencia a los datos recibidos y luego llama a la rutina de servicio de evento asociada. En este punto, la aplicación debe recuperar los datos de los Buffers de ECB .

Si el socket de destino es un socket look ahead, el handler interno de IPX llama a la función ReceiveLookAheadHandler del socket. Este handler puede entonces determinar si la aplicación desea el paquete. Si es así debe construir un ECB del estilo-ODI para referenciar a los buffers de recepcion y a la rutina de servicio de evento de la aplicación.

ReceiveLookAheadHandler, retorna , a este ECB, pasándole un puntero a IPX. (para las especificaciones del interface de ReceiveLookAheadHandler mirar el Protocolo de Transporte Cliente API para Assemblyreference.)

Restricciones del socket Look Ahead:

Los sockets abiertos por IPXOpenLookAheadSocket se manejan diferente de los sockets abiertos por IPXOpenSocket. Las siguientes restricciones se aplican a los sockets look ahead:

⇨ Todos los sockets look ahead se abren en long term. Por esto, una aplicación que abre un socket look ahead debería siempre cerrarlo antes de que dicha aplicación termine.

- El recibir ECBs no se puede poner en un socket look ahead.

- Las funciones SPX no pueden ser usadas con un socket look ahead.

A pesar de estas diferencias, se cierra un socket look ahead como si fuera un socket normal , llamando a IPXCloseSocket. Además , como un socket normal, un socket look ahead se puede usar para enviar paquetes o planificar eventos asíncronos.

La estructura Look Ahead:

Cuando IPX llama al ReceiveLookAheadHandler de tu aplicación, éste le pasa una estructura LookAhead que referencia al paquete recibido. La estructura incluye los siguientes campos:

- Look Ahead address
- Look Ahead length

El campo Look Ahead addres es la dirección del buffer que posee los primeros n bytes del paquete recibido.

El campo Look Ahead length contiene la longitud del buffer look ahead.

El campo Look Ahead length es igual o mayor que el tamaño de el look ahead pedido por la aplicación cuando llamó a IPXOpenLookAheadSocket. De cualquier forma, si la longitud total del paquete es menor que el tamaño pedido, el campo look ahead length contiene la longitud del paquete. Si este valor es menor que el tamaño mínimo pedido, descarta el paquete.

TEMA 6
DESARROLLO DE UNA APLICACIÓN CLIENTE/SERVIDOR

6. DESARROLLO DE UNA APLICACIÓN CLIENTE/SERVIDOR.

6.1 Descripción del programa servidor. Características NLM.

El programa que corre en el servidor es un módulo cargable NetWare y se encarga de proporcionar una comunicación efectiva entre los clientes que piden servicios y los que los suministran.

Los clientes que suministran los servicios son los que ponen a disposición de posibles clientes las utilidades de impresión remota y búsqueda remota. También nos referiremos a ellos como servidores remotos.

Los clientes que piden servicios son aquellos que desean utilizar los servicios suministrados por los servidores remotos.

Al programa que corre en el servidor nos referiremos como ***programa servidor*** o servidor. Este es el encargado de sincronizar los procesos entre ambos tipos de clientes.

El programa servidor hace uso de un poderoso recurso como es la ***multitarea.*** Gracias al uso de ella se ha conseguido una perfecta sincronización entre aquellos clientes que piden servicios y aquellos que los suministran. De otra manera sería imposible que varios clientes realizaran peticiones de servicios de manera simultánea a la vez que también son atendidos de manera simultánea.

El método estándar de iniciar y llevar a cabo una comunicación TLI , es el siguiente:

1.- Abrir un extremo de conexión TLI.

2.- Asociar ese extremo a una dirección.

3.- Establecer una conexión.

4.- Enviar y recibir datos.

5.- Desconectar.

Se desarrolló en detalle cuando se explicó el interfase de nivel de transporte.

El programa comienza abriendo una serie de ***semáforos*** que se utilizarán posteriormente para tener un control efectivo de la multitarea. Los semáforos son una parte importante en la implementación de un NLM multiproceso. Estos permiten la cooperación entre procesos ligeros y el uso eficiente del tiempo de proceso de servidor.

```
SemEsc0 = OpenLocalSemaphore(0L);
SemEsc1 = OpenLocalSemaphore(0L);
          .
          .
```

A continuación se inicializan las diferentes estructuras que se utilizarán en las llamadas a las diferentes funciones TLI.

Se abren dos sockets:

- Uno para la comunicación con los servidores remotos (Socket 31).
- Uno para la comunicación con los diferentes clientes (Socket 41).

El fragmento de código siguiente muestra la menera en que se abren los sockets en el programa servidor.

.

.

```
/* Abrimos un extremo de conexión, no es necesaria la información */
if((*fd = t_open("/dev/nspx", O_RDWR , NULL)) = = -1)
  {
   t_error("t_open");
   exit(1);
  }

/* Inicializamos la estructura tbind y llamamos a t_bind para enlazar */
spx_addr.ipxa_socket[0] = 0;
spx_addr.ipxa_socket[1] = SPX_SOCKET;
tbind- > qlen = 5;
tbind- > addr.len = sizeof( spx_addr );
tbind- > addr.buf = (char *)&spx_addr;
tbind- > addr.maxlen = sizeof (spx_addr);
  if( t_bind(*fd, tbind, tbind) = = -1 )
        .
        .
```

Se registran con la función de C ***atexit*** las funciones que se ejecutarán cuando el módulo NLM intente ser descargado por algún usuario, desde la consola de servidor, con el comando ***unload*** o el programa aborte por falta de recursos que suministrar a los posibles clientes. (no hay servidores remotos activos)

Cuando se activa esta función, se realiza un llamada a otra función que se denomina ***DespertarProcesos***, y que se encarga de liberar todos aquellos recursos de la red que han sido tomados por el programa durante su ejecución.

```
atexit(DespertarProcesos);
        .
        .

void DespertarProcesos(void)
        {
                SignalLocalSemaphore(SemEsc0);
                SignalLocalSemaphore(SemEsc1);
                        .
                        .


                myunload();
                LiberarRecursos();
                return;
        }
void LiberarRecursos(void)
        {
                CloseLocalSemaphore(SemEsc0);
                CloseLocalSemaphore(SemEsc1);
                        .
                        .
```

Las funciones ***DespertarProcesos*** **y** ***LiberarRecursos*** son vitales para una terminación ordenada de los procesos que forman el NLM. Una mala gestión de los semáforos ocasiona con facilidad bloqueo del servidor.

La función ***myunload*** se encarga de devolver los recursos tomados por los controladores de pantalla. Restaura los manejadores de pantalla y destruye las pantallas que hayan sido abiertas.

A continuación se crean los procesos ligeros que se encargarán de ejecutar las diferentes tareas dentro del entorno de la multitarea.

El programa registra a los diferentes servidores remotos, y establece una conexión estable e independiente con cada uno de ellos.

Por cada servidor remoto se crea un proceso y por cada conexión entrante de cliente se crea otro proceso. De esta manera se puede controlar de una manera mucho más efectiva los diferentes accesos de servidores remotos y clientes.

El trozo de código que aparece a continuación muestra como se crean los procesos ligeros para los servidores remotos. Se crean tantos como servidores remotos hayan. Para más información sobre las funciones que aparecen remitir a "NLM Library Reference".

```
for (Num = 0;Num < NumServImp;Num + +)
  {
    if (t_listen(fh,&tcall31) = = -1)
      {
        t_error("t_listen");
        exit(1);
      }
switch (Num)          {
case 0:
```

```
if((fh_new = AceptarConeccion31Imp0())! = NULL)
  {
    if(BeginThread(ProcesarConeccion31Imp,NULL,NULL,
                                                (void *)fh_new) = = -1)
          {
            printf("BeginThread falló\n");
            exit(1);
          }
  }
break;
case 1:
.
.
```

Una vez han sido dados de alta los servidores remotos, el programa entra en un bucle en el cual se encuentra en constante escucha de posibles clientes. Por cada nuevo cliente se crea un nuevo proceso ligero.

Los procesos ligeros que se van creando se ejecutan dentro del entorno de la multitarea. Gracias a todo esto el programa puede atender peticiones de clientes que ya han establecido la conexión, mandar peticiones a los diferentes servidores remotos y seguir escuchando por nuevas conexiones, a la vez que presenta por pantalla información de los diferentes eventos.

El siguiente extracto de código muestra la parte fundamental del programa. En ella y después de haber realizado todas las tareas previas, entra en un bucle en el que constantemente está escuchando por la entrada de posibles clientes. Cada entrada de un nuevo

cliente generará un nuevo proceso.

La función ***ThreadSwitch*** realiza un cambio de contexto. Como ya se explicó, un cambio de contexto generado por esta función hace que el proceso que se está ejecutando en el momento de la llamada pase de ejecutarse en la CPU a esperar al final de la cola ***RunList.***

Se realiza una llamada a esta función para sincronizar en el programa la ejecución de los procesos y evitar que un proceso se adelante a otro cuando no debiera.

```
for (;;)
  {
    i (t_listen(fh2,&tcall41) = =-1)
     {
        t_error("t_listen 41");
        exit(1);
     }
     if((fh_new2 = AceptarConeccion41()) != NULL )
       {
       if(BeginThread(ProcesarConeccion41,NULL,NULL,
                                          (void *)fh_new2) = =-1)
     {
       printf("BeginThread falló\n");
       exit(1);
     }
  }
 ThreadSwitch();  /* Cambio de contexto */
} /* Final del for (;;) */
```

```
}       /* Final de main()    */
```

Una vez los servidores remotos han sido dados de alta, entran en un estado de espera. Las posibles peticiones de clientes se encargarán de sacarlos de este estado.

El estado de espera se consigue gracias a la poderosa herramienta que son los semáforos dentro de la multitarea. De tal manera , los procesos que rigen las tareas a ejecutar por los servidores remotos, quedan detenidos en un semáforo, esperando que la petición de servicio de un determinado cliente active el semáforo y haga que el flujo de ejecución del proceso continúe.

La entrada de los diferentes clientes hace que se creen nuevos procesos ligeros. La petición por parte de estos de una determinada tarea , como puede ser impresión o búsqueda remota, hace que los procesos ligeros de los servidores remotos detenidos arranquen.

Estos se encuentra en un bucle de trabajo, de tal manera que cuando han acabado de realizar la labor pedida vuelven a detenerse en el mismo semáforo en espera de una nueva petición.

El programa servidor atiende a los clientes por el socket 41 y a los servidores remotos por el 31. Cuando hay alguna petición de entrada de algún cliente, eéte comienza el proceso de establecimiento de la conexión. Una vez establecida la conexión el programa servidor pide al cliente que elija una determinada impresora dentro de las disponibles.

Cada impresora está relacionada con un servidor remoto de tal manera que al elegir una impresora, se está eligiendo indirectamente un determinado servidor remoto. La

posibilidad existente es de dos impresoras por servidor remoto.

Una vez finalizado el establecimiento de la conexión con el cliente, y se haya realizado la elección de la impresora, el programa servidor preguntará al cliente que tipo de trabajo quiere que se realice.

Si el cliente ha elegido la opción de ***"búsqueda"*** el programa servidor pedirá el nombre del fichero a buscar en el servidor remoto. Este servidor remoto es el que corresponde a la impresora elegida anteriormente por el cliente.

Ha llegado el momento de despertar al proceso ligero, que corresponde con el servidor remoto elegido, dormido en el semáforo en espera de alguna petición. Este proceso es el que se encarga de la comunicación con el servidor remoto.

El proceso ligero que se estaba ejecutando y que era el encargado de la gestión del cliente, quedará dormido en un semáforo y sólo será despertado cuando el servidor remoto dé contestación de las peticiones hechas.

Al servidor remoto se le mandará el nombre del fichero a buscar y éste al recibirlo ejecutará un programa encargado de la búsqueda exhaustiva a través de las diferentes unidades locales . Si el fichero es encontrado, en el paquete de regreso se enviará el lugar o los lugares donde haya sido encontrado, de lo contrario, éste irá vacío.

Una vez al programa servidor y más en concreto al proceso que se encarga de gestionar las tareas del servidor remoto, le llega contestación, éste despertará al proceso

dormido del cliente, al que se le asignará la tarea de remitir la información necesaria al destinatario sobre el resultado de la petición realizada.

De igual manera, si el trabajo eligido , en vez de la ***"búsqueda"*** ha sido ***"impresión"*** se le preguntará al cliente el nombre del fichero a imprimir , con su camino completo.

Si el fichero que el cliente quiere imprimir , está ubicado en alguna unidad de red, el programa servidor cogerá directamente el fichero de la red y lo mandará en paquetes de 132 caracteres al servidor remoto.

La tarea de despertar y dormir procesos se ejecuta de idéntica manera tanto para la opción de "búsqueda remota " como para la opción de "impresión remota".

Cuando el servidor remoto ha sido advertido del nuevo trabajo (impresión), comienza el envío del fichero en paquetes de 132 caracteres. Este lo recibirá y almacenará en un fichero local temporal, creado para tal propósito. Cuando la transferencia haya concluido, el servidor remoto mandará a imprimir el fichero, eliminando posteriormente el archivo temporal.

Para el caso en que el fichero a imprimir no esté ubicado en una unidad de red sino en una unidad local del cliente que hace la petición. El fichero será previamente transferido al programa servidor , y de éste al servidor remoto.

Cuando el trabajo ha concluido, el proceso ligero encargado de la gestión del servidor remoto, volverá a dormirse. El del cliente volverá a despertarse y un mensaje será enviado informando al cliente de lo acontecido. El número de posibles servidores remotos está limitado a tres, aunque puede ser fácilmente ampliado a cualquier número, dependiendo éste

únicamente de las licencias de la red en la que se esté ejecutando.

El programa servidor registra las desconexiones de los diferentes servidores remotos, comunicándole al cliente que ese servidor ha sido desconectado. Si llegara el caso de que todos los servidores se hubieran desconectado, el programa se abortará automáticamente ,mandando desconexión a todos aquellos posibles clientes que aún queden conectados.

Se muestra a continuación como se realiza la comprobación de una posible petición de desconexión por parte de un servidor remoto.

```
if( t_look(fh) = = T_DISCONNECT )
  {
        .
        .
   if  (t_rcvdis(fh, NULL) < 0 )
     t_close(fh);
        .
```

Es importante un control riguroso de los recursos del sistema que han sido usados por los programas y más en conccreto por el del programa servidor. Un olvido en la devolución de los recursos puede ser nefasta. Así, descargar el programa servidor sin cerrar los semáforos puede ocasionar la caída del sistema.

Los servidores remotos, pueden compartir hasta dos impresoras , y el posible cliente puede elegir entre esas dos impresoras.

6.1.1.Diagrama de flujo

INICIO

SE ABREN LOS SEMÁFOROS PARA EL CONTROL DE LOS PROCESOS.
- *SALIDAS POR PANTALLA*
- *CONTROL DE CLIENTES*
- *CONTROL DE SERVIDORES REMOTOS*

SE ABREN SOCKETS Y SE INICIALIZAN ESTRUCTURAS
- *SOCKET 31 PARA SERVIDORES REMOTOS*
- *SOCKET 41 PARA CLIENTES*

SE REGISTRAN LAS FUNCIONES A EJECUTAR PARA UNLOAD'

SE CREAN LOS PROCESOS LIGEROS
- *PANTALLA*
- *SERVIDORES REMOTOS*
- *CLIENTES*

SOCKET 31

ESTABLECIMIENTO DE CONEXIÓN

SERVIDOR REMOTO 1

SERVIDOR REMOTO 0

(2)

ESPERA EN UN SEMÁFORO A SER ACTIVADO

CUANDO LLEGUE UNA PETICIÓN DE CLIENTE SE DESPIERTA.

PETICIÓN CLIENTE

TRABAJO??

IMPRESIÓN

BÚSQUEDA

SI

DESCONEXIÓN?

NO

(1)

NO

ÚLTIMO?

SI

SE MANDA EL FICHERO

SE MANDA NOMBRE FICHERO

SE COMUNICA CLIENTE

SE RECIBE CONTESTACIÓN

SE RECIBE CONTESTACIÓN

FIN DE PROCESO LIGERO

DESPIERTA PROCESO LIGERO CLIENTE

DESPIERTA PROCESO LIGERO CLIENTE

(2)

SE ABORTAN TODAS LAS CONEXIONES CON CLIENTES

SE DEVUELVEN RECURSOS Y SE CIERRAN SOCKETS

FIN NLM

SE REPITE LA ESTRUCTURA DEL SERVIDOR REMOTO 0 PARA EL 1 Y 2

SOCKET 41

(4)

ESCUCHANDO POSIBLES CONEXIONES

NO COMIENZA HASTA QUE EL PROCESO DE ESTABLECIMIENTO DE CONEXIONES CON SERVIDORES HAYA ACABADO.

NO

CONEXIÓN??

SI

ESTABLECIMIENTO DE CONEXIÓN

(3)

ELECCIÓN DE IMPRESORA

SEGÚN IMPRESORA SERVIDOR REMOTO 0 1 o 2

TRABAJO

?

BUSQUEDA

IMPRESIÓN

NOMBRE FICHERO

LLEGA EL FICHERO

SE DESPIERTA EL SERVIDOR ELEGIDO

SE DESPIERTA EL SERVIDOR ELEGIDO

ESPERA EN SEMÁFORO HASTA QUE CONTESTA SERVIDOR REMOTO

ESPERA CONSTETACIÓN DEL SERVIDOR REMOTO

DESPIERTA

DESPIERTA

COMUNICADO AL CLIENTE

COMUNICADO AL CLIENTE

SI

QUIERE SEGUIR??

NO

(3)

(4)

6.2 Descripción del programa cliente.

Este programa se encarga de la comunicación con el servidor, con el único fin de la petición de servicios, como son el de ***"búsqueda" e "impresión"*** remota.

Se comunica con el programa servidor a través de socket 41.

Cada vez que se establece una comunicación con el servidor, se crea un nuevo proceso que se encarga de la gestión de sus peticiones.

Básicamente , este programa tiene dos posibilidades: la impresión y la búsqueda. Cuando se elige la impresión remota, se manda el fichero que se desea imprimir al servidor, y a través de éste se gestionará con destino hacia el servidor remoto que ha sido elegido previamente.

Si la opción de búsqueda es seleccionada, se mandará el nombre del fichero, del cual queremos solicitar la búsqueda, y que el programa servidor, de igual manera, gestionará hacia el servidor remoto que hemos elegido.

En ambos casos se recibe un mensaje del servidor remoto que gestiona el programa servidor y que nos comunica el resultado de la operación solicitada.

Si los posibles servidores remotos se desconectaran, la conexión del cliente con el programa servidor, sería automáticamente abortada por este último.

El programa comienza recibiendo una serie de parametros desde la línea de comandos al arrancar. Estos sirven para la localización del servidor y son:

- Nodo de red
- Dirección de servidor

Como parte del establecimiento de la comunicación es necesario conocer no sólo el número de socket sino también la dirección del nodo físico (o de la tarjeta de interfaz) donde reside el programa con el que deseamos comunicarnos. En este caso tanto el programa cliente cómo el programa servidor remoto se comunican entre ellos a través del programa servidor. Por lo tanto, el único programa con el que se comunican directamente es el que corre en el servidor NetWare y es la dirección del nodo físico que les interesa conocer.

Cada red posee un número que es la dirección de su sistema cableado.Este es el número de red.

Se ha de proporcionar una dirección para que el paquete que se envía por la red llegue a su destino. Esta dirección para una comunicación TLI sobre SPX es una dirección IPX.

Una dirección IPX esta formada por tres campos:

- NET_ADDR: número de red
- NODE_ADDR: dirección del nodo.
- SPX_SOCKET: socket IPX.

La función que aparece a continuación se encarga de colocar las direcciones de entrada en las variables correspondientes, con el fin de rellenar la dirección IPX. A esta función se la pasa como parametro **spx_addr** que en una **IPX_ADDR,** es decir una dirección IPX.

La función rellena la dirección del nodo y la dirección de red.

```
int PasaDireccion( char *direc, IPX_ADDR *destino )
 {
  if ( strlen( direc ) = = 21 && direc[ 8 ] = = '/' )
  if ( PonDireccion( direc, destino- > ipxa_net, 4 ) )
  if ( PonDireccion( &direc[ 9 ], destino- > ipxa_node, 6 ) )
      .
      .
 }
```

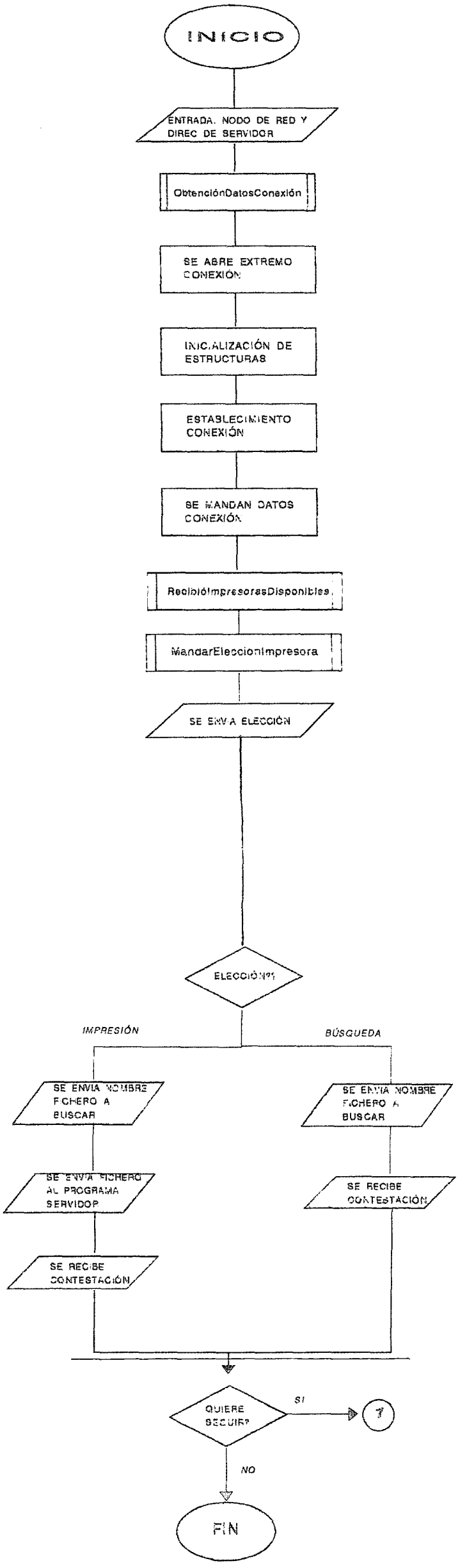
Nos falta comunicar el número de socket por el cual se realizará la comunicación.

```
#define   SPX_SOCKET      31

      .
      .
  spx_addr.ipxa_socket[ 0 ] = 0;
  spx_addr.ipxa_socket[ 1 ] = SPX_SOCKET;
      .
      .
```

Otra función a destacar dentro del código del programa sería la que se encarga de obtener los datos referentes a la conexión, tales como login, número de conexión...,etc.

Esta función utiliza llamadas a la **NWCALLS** .Estas llamadas hacen subir el programa un nivel dentro de la arquitectura de capas.

6.2.1.Dagrama de flujo



6.3 Descripición del programa servidor remoto

De este programa existen dos versiones, la versión residente y la versión dedicada.

- ⇨ En la **versión dedicada**, el servidor remoto queda sacrificado en pro de suministrar los servicios de impresión y búsqueda remota.
- ⇨ En la **versión residente**, el programa cohabita, con otros programas de usuario, evitando de esta manera que el PC quede sacrificado.

Básicamente, las dos versiones son iguales. La diferencia entre ambas , no está en la comunicación TLI , sino en el modo en que se ejecutan . Uno es residente y el otro no.

Para lograr que el programa fuera residente, se ensamblaron unas rutinas en lenguaje ensamblador con el programa en C . Esta versión , la primera vez que se ejecuta queda residente en memoria, y la segunda se descarga de memoria.

El programa una vez anclado en memoria se ejecuta repetidamente gracias a la redirección de la interrupción del temporizador (8253), a una rutina de interrupción de usuario, que se encarga de la ejecución del programa.

El programa comprueba en todo momento antes de ejecutarse, que no hay acceso a disco ni a disquetera, y que no existe ninguna función del sistema que se esté ejecutando y no pueda ser interrumpida. Cuando encuentra el hueco, pasa a primer plano de ejecución, y cuando concluye, devuelve el sistema al estado en el que estaba antes de su ejecución.

Una segunda llamada desde el prompt del dos, al programa, hace que éste se desinstale, liberando la memoria anteriormente ocupada y mandando al programa servidor un aviso de desconexión.

6.3.1.Modo residente

Se ofrece la posibilidad con este modo de instalar el programa servidor remoto de manera residente en memoria, eliminando de esta manera un equipo dedicado al uso exclusivo de la ejecución del programa.

Los programas residentes frecuentemente trabajan con la lógica de instalarse en la primera llamada y eliminarse en la siguiente.

Se utilizó un interfase de ensamblador a un programa C para lograr dejar anclado en memoria el programa. El interfase utiliza una serie de rutinas que se describen a continuación:

- **TsrInit:**
  Convierte el programa servidor remoto en un programa residente. Instala los controladores de interrupciones , lo acaba y lo ancla en memoria de manera residente.
- **TsrIsInst:**
  Se encarga de averiguar si existe una copia ya instalada del programa en memoria.
- **TsrCanUninst:**
  Averigua si la copia que está instalada en memoria puede ser eliminada de nuevo.
- **TsrUnInst:**
  Elimina una copia del programa instalada anteriormante en la memoria.
- **TsrSetptr:**
  Guarda la dirección de la rutina que se ha de llamar en la copia instalada del programa.

El programa al inicio realiza una llamada a la función ***TsrIsInst*** y si descubre que existe una copia instalada del programa intenta desinstalarla. Se llamará a la función ***TsrCanUnInst*** que se encarga de averiguar si la copia ya instalada puede ser eliminada, ya que ésto no es siempre posible.

El caso de no poder eliminar la copia del programa ya instalada, se suele dar cuando después de la instalación del programa, se instala otro que también redirecciona el vector de interrupción del temporizador. Por esta razón la función comprueba si todas las interrupciones redireccionadas siguen apuntando a la copia del programa residente.

La función ***TsrCanUnInst*** devuelve el valor TRUE si el programa se puede desinstalar. Entonces se llamará a la función ***TsrUnInst*** para eliminar la copia intalada en memoria.

Para todo ello se instalan de nuevo los antiguos controladores de las interrupciones 08h,13h,28h y 2Fh. A continuación se libera el programa de memoria.

A continuación se muestra un extracto de código que muestra la rutina del nuevo controlador de interrupciones 08h (temporizador).

```
.
.
; el nuevo controlador de interr. 08h (Timer)

int08    proc far
         dec  tsrnow
         cmp  tsrnow,0     ; ¿debe activarse el programa residente?
         jne  i8_end
```

```
;El programa debe activarse, pero ¿es posible?

        cmp  in_bios, 0   ; ¿Interrupción BIOS de disco activa?
        jne  i8_end
        call dosactive    ; ¿se puede interrumpir DOS?
        je   i8_tsr       ; Si, llamar entonces llamar al programa residente
i8_end: jmp  [int8_ptr]   ;saltar al controlador antiguo

        ; Activar el programa residente

i8_tsr:  mov  tsrnow,TIME_OUT      ;TSR ya no espera a la activación
         mov  tsractive,1          ;TSR está activo
         pushf                     ;simular llamada del controlador antiguo
         call [int8_ptr]           ;mediante el comando INT 8h
         call start_tsr            ;ejecutar el programa TSR
         iret                      ;volver al programa interrumpido
int08    endp
      .
      .
```

Esta es una de las partes más importantes del módulo de ensamblador, ya que en ella recae la activación del programa. Cada "tic" del temporizador provoca un salto a la dirección de la rutina de la interrupción. Como nosotros tenemos redireccionada la interrupción a nuestra rutina ésta será la que se ejecute.

Se puede observar en el código que aunque el temporizador haya provocado una interrupción, el programa no pasa a ejecutarse de manera inmediata.

Se ha de comprobar si la interrupción del sistema el posible, ya que el DOS puede estar realizando alguna tarea que no se puede interrumpir o se está realizando algún acceso a disco o disquetera (int 13h).

Se realizó el desarrollo bajo el modelo de memoria SMALL.

Para más información sobre redirección de interrupciones, controladores de interrupciones y en general desarrollo de programas residentes en lenguajes de alto nivel remitirse a "PC Interno" de la editorial Marcombo.

Una vez explicado de manera general la manera de hacer residente nuestro programa de alto nivel pasemos a la descripción de éste.

El programa una vez queda anclado en memoria está ejecutandose de manera permanente (siempre que sea posible interrumpir el sistema).

Cuando el programa pasa a ejecutarse a primer plano, lo primero que hace es comprobar el estado de la conexión. Si hay datos, continuará el flujo de ejecución normal del programa. En caso contrario , devolverá el control al DOS como se muestra en el fragmento de código que aparece a continuación.

```
.
.
int STATE;
.
.
t_nonblocking(fd);
STATE = t_getstate(fd);
```

```
if (STATE != T_DATAXFER)
  return;
        .
        .
t_blocking(fd);
        .
        .
t_rcv(fd,NombreFicheroBusca,sizeof(NombreFicheroBusca),&flags);
        .
        .
```

Se puede observar como después de la realización de las tareas previas de inicialización y establecimiento de la conexión con el programa servidor, el programa servidor remoto comprueba el estado del extremo de conexión. Si no hay datos devuelve el control al DOS, si hay, inicializa el modo bloqueo para que no exista la posibilidad de perdidas de datos, ya que no puede ser interrumpido. A continuación comienza la recepción.

Dependiendo de la tarea asignada (búsqueda o impresión) el programa se desviará hacia una parte del código u otra, mandando cuando haya finalizado , un mensaje referente a la ejecución de la tarea realizada. En este momento, el programa devolverá nuevamente el control al DOS y seguirá comprobando la llegada o no de nuevos datos.

En el modo de búsqueda se ejecuta una función que se encarga de la búsqueda de ficheros a través de los directorios y subdirectorios de las diferentes unidades locales.

A destacar ya por último una función que está presente en los tres programas y que se encarga de recibir y averiguar la razón de una posible desconexión.

Su código se muestra a continuación y se puede observar en él como la lectura de la variable discon, que es una estructura del tipo t_discon, nos puede suministrar más información sobre las posibles causas de la desconexión.

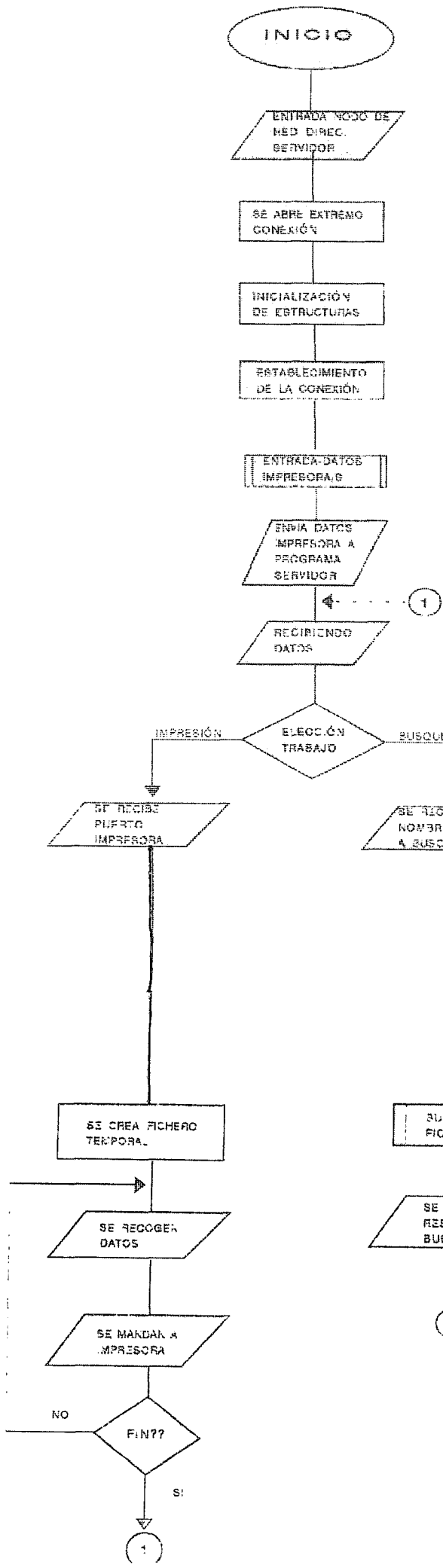
```
void RazonDesconeccionSPX( int fd )
{
struct t_discon    discon;
char          *msg;

if ( t_rcvdis( fd, &discon ) = = -1 )
        {
        t_error( "La función t_rcvdis ha fallado" );
        exit( 2 );
        }

switch( discon.reason )
{
        case TLI_SPX_CONNECTION_FAILED:
                msg = "La conexión ha fallado";
        break;

        case TLI_SPX_CONNECTION_TERMINATED:
                msg = "La conexión ha sido terminada por el cliente";
        break;

        case TLI_SPX_MALFORMED_PACKET:
                msg = "Error de interfase, malformación de paquete";
        break;
```

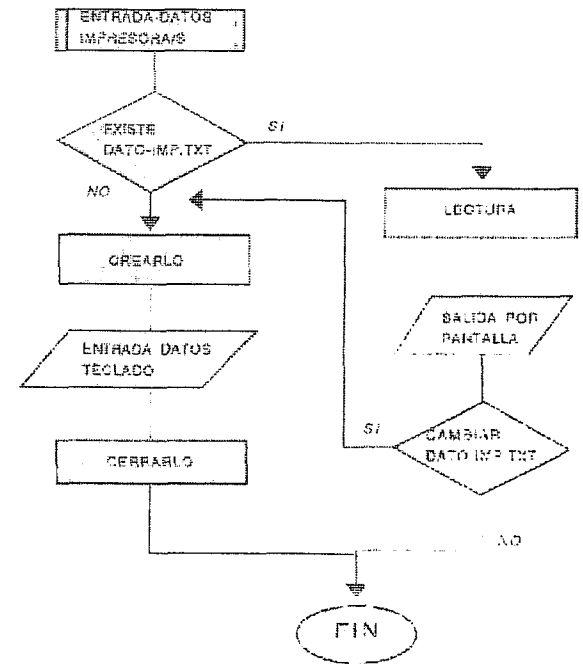
```
        default:
                msg = "Causa no recogida";
}
printf( "Conexión SPX finalizada: %s\n", msg );
}
```

6.3.2.Modo dedicado

Este programa es idéntico a la versión residente pero sin el módulo de ensamblador y las llamadas a éste. Quizás la única diferencia estriba en que el dedicado tiene un bucle permanente que lo hace estar constantemente escuchando o realizando alguna tarea encomendada. Este bucle es implementado con una estructura repetitiva que en el caso del residente, el encargado de esta labor es el temporizador.

6.3.3.Diagrama de flujo





TEMA 7
BIBLIOGRAFÍA

7.BIBLIOGRAFÍA

⇨ Novell NetWare 386. Manual de Referencia.
Tom Shekdon.
McGraw Hill.

⇨ NetWare 386. Programmer's Guide.
Ralph Davis.
Addison-Wesley.

⇨ Guía de programación Novell NetWare.
Michael Day, Michael Koontz y Daniel Marshall.
Anaya.

⇨ Manuales, guías y librerías del Kit de desarrollo SDK.

⇨ PC interno. Programación del sistema.
Michael Tischer.
Marcombo.

⇨ Instalación y Actualización.
Novell
NetWare 3.12

⇨ Instalación y Actualización.
Novell
NetWare 3.11

➪ Borland C++ Manual de Bolsillo
Luis Joyanes Aguilar.
McGraw Hill

➪ La biblia del TURBO C. Fundamentos y técnicas avanzadas de programación.
Scott Zimmerman / Beverly B. Zimmerman
Anaya

ANEXO A

El siguiente presupuesto ha sido realizado teniendo en cuenta que el precio en pesetas de la hora es de **3.000.**

PRESUPUESTO

DESCRIPCIÓN	HORAS EMPLEADAS	MESES EMPLEADOS	COSTE (pts)
Estudio del NOS	-	1	-
Estudio del interfaz con C para el desarrollo de aplicaciones cliente/servidor.	-	6	-
Desarrollo de aplicación cliente/servidor.	150	-	450.000
TOTAL			**450.000**

ANEXO B

LISTADO PROGRAMA SERVIDOR

```
/*************************************************************************/
/* Este módulo tiene como función atender las peticiones de impresión */
/* de diferentes clientes mandando sus ficheros a impresoras remotas. */
/* Además busca en esos "Servidores Remotos de Impresión", ficheros   */
/* cuyos nombres son suministrados por los clientes.                  */
/* Se abren dos canales lógicos para la comunicación, uno para los    */
/* diferentes "Servidores Remotos de Impresión" y otro para los       */
/* diferentes clientes, todo ello dentro del entorno de la multitarea.*/
/* Para cada nueva conexión se crea un nuevo proceso (Thread).       */
/*************************************************************************/

#include <stdio.h>
#include <stdlib.h>     /* malloc, free              */
#include <process.h>    /* exit, Thread..., delay    */
#include <string.h>     /* memcpy                    */
#include <fcntl.h>      /* O_RDWR flag               */
#include <tiuser.h>     /* Funciones y estructuras TLI */
#include <tispxipx.h>   /* IPX_ADDR                  */
#include <nwconn.h>     /* GetInternetAddress...     */
#include <nwmisc.h>     /* IntSwap                   */
#include <sap.h>        /* Advertiseservice          */
#include <nwsemaph.h>   /* OpenLocalSemaphore,.....  */
#include <conio.h>
#include <nwenvrn.h>    /* GetFileServerName,..      */
#include <time.h>       /* Para la fecha...          */
#include <io.h>         /* filelengh..               */
#include <dos.h>
#include <advanced.h>
#include <nwsnut.h>     /* Para la pantalla..        */
#include <malloc.h>

#define  TECLAF1   59   /* Para acabar desde el Servidor de ficheros con */
                        /* los Servidores Remotos. */


/************************/
/** Variables globales **/
/************************/

/* Semáforos para controlar los Servidores de Impresión 0, 1 y 2.*/
/* Se encargan de dormir los procesos de impresión hasta que    */
/* llegue alguna petición por parte de algún cliente.           */

LONG            SemEsc0,SemEsc1,SemEsc2;

/* Semáforo para el control de la búsqueda de ficheros en los   */
/* Servidores de Impresión Remotos.                             */

LONG            AcaboLaBusqueda;

/* Para éxito o fracaso en la gestión de los ficheros a imprimir */

LONG            SemMandarExito;

/* Para el menú de pantalla y entrada de datos desde teclado    */
```

```
LONG            SemPantalla,EntradaTeclado;

struct t_call   tcall31,tcall41;   /* Diferentes estructuras del */
struct t_bind   tbind;             /* interfaz de transporte.    */
int             fh,fh2;            /* Manejadores de conexión    */

int             NumServImp, /* Número de servidores de impresión */
                    NumeroServImp,      /* Para no perder NumServImp */
                    trabajo;    /* Número de trabajo global         */

char            FICHERO[50],   /* Nombre de fichero              */
                    Puerto[10];    /* Puerto de la impresora elegida */

/* Datos de impresoras que llegan de los servidores remotos      */

char            DirServ0Imp0[132],DirServ0Imp1[132], /* Servidor 0 */
                    DirServ1Imp0[132],DirServ1Imp1[132], /* Servidor 1 */
                    DirServ2Imp0[132],DirServ2Imp1[132], /* Servidor 2 */
                    DirServImp[132];

/* Variables para recoger la desconexión de los Servidores Remotos */

int             AbortarServImp0,Abortar=0,AbortarServImp;
int             AbortarServImp1;
int             AbortarServImp2;

char            datos[132];  /* Buffer de datos */

int             Local;      /* Fichero local a WS               */
int             Borrar;     /* Para borrar archivo temporal.    */
int             exito;      /* Para el éxito o fracaso del envío */
                                 /* del fichero a la impresora.      */

int             NumeroImp0,      /* Impresoras Servidor Remoto 0 */
                    NumeroImp1,      /* Impresoras Servidor Remoto 1 */
                    NumeroImp2;      /* Impresoras Servidor Remoto 2 */

int             NumeroTotalImp = 1;   /* Número total de impresoras */

char            iobufForCli[132], /* Comunica al cliente resultados */
                                      /* sobre la gestión de su fichero */

                    NombreServidorFicheros[70];  /* Nombre del Servidor */
                                                    /* de ficheros        */

/* Para recoger las elecciones de trabajo (búsqueda, impresión,..), */
/* el nombre del fichero a buscar y el lugar donde se encuentran.   */

char            Eleccion0[132],NombreFicheroBusca0[132],
                    LugarEncuentroFichero0[300];   /* Servidor Remoto 0 */

char            Eleccion1[132],NombreFicheroBusca1[132],
                    LugarEncuentroFichero1[300];    /* Servidor Remoto 1 */

char            Eleccion2[132],NombreFicheroBusca2[132],
                    LugarEncuentroFichero2[300];    /* Servidor Remoto 2 */
```

```
char            Eleccion[132],NombreFicheroBusca[132],
                    LugarEncuentroFichero[300];      /* Para uso global */

/* Variables para el menú de pantalla */

char            Pantalla[60], /* Para formar frases a presentar en pantalla */
                    Pantalla1[60];

int             Datos0 = 0; /* Hay o no hay datos destinados al Servidor Remoto 0 */

LONG            fila;       /* Para avanzar en filas en la pantalla */
NUTInfo         *handle;
LONG            NLMHandle;
LONG            allocTag;
int             CLIBScreenID;
PCB             *pPtr, *ipPtr;
BYTE            *ptr;

int             tecla;  /* Para recoger posible aborto desde teclado */

/* Para cerrar la ventana de información */

int             FinalSalidaPantalla;

int             cliente;   /* cliente = 1 nueva petición de cliente */

/*********************************************/
/** Prototipos de las funciones utilizadas **/
/*********************************************/

int  AceptarConeccion31Imp0(void);         /* Servidor de Impresión 0 */
void ProcesarConeccion31Imp0(void *parm);
int  AceptarConeccion31Imp1(void);         /* Servidor de Impresión 1 */
void ProcesarConeccion31Imp1(void *parm);
int  AceptarConeccion31Imp2(void);         /* Servidor de Impresión 2 */
void ProcesarConeccion31Imp2(void *parm);
int  AceptarConeccion41(void);             /* Posibles clientes       */
void ProcesarConeccion41(void *parm);
void AbrirSocket(int *fd,unsigned char SPX_SOCKET,struct t_bind *tbind);
void DespertarProcesos(void);     /* Para  cuando descarga con UNLOAD */
void LiberarRecursos(void);
void EntradaDatosDiarios(void);       /* Almacena los eventos diarios */
void WriteInstructions (BYTE *p, PCB *pcb);     /* Debajo de pantalla */
void myunload(void);    /* Para cuando descarga del módulo con UNLOAD */
void PantallaMenu();           /* Pantalla de presentación con NWSNUT */

/*************************************/
/** Comienzo de la función principal **/
/*************************************/

void main(void)
 {
   time_t          FechaActual;    /* Para la fecha */
   int             fh_new,fh_new2;
   int             Num;
   unsigned char   SPX_SOCKET_1=31; /* Socket Servidores de Impresión */
```

```
unsigned char    SPX_SOCKET_2=41; /* Socket para clientes */
struct t_bind    tbind1,tbind2;

/* Abrimos semáforos para los servidores de impresión, pantalla... */

SemEsc0 = OpenLocalSemaphore(0L);
SemEsc1 = OpenLocalSemaphore(0L);
SemEsc2 = OpenLocalSemaphore(0L);
SemMandarExito = OpenLocalSemaphore(0L);
AcaboLaBusqueda = OpenLocalSemaphore(0L);
SemPantalla = OpenLocalSemaphore(0L);
EntradaTeclado = OpenLocalSemaphore(0L);

/* Abrimos los Sockets para los canales lógicos de comunicación */

AbrirSocket(&fh2,SPX_SOCKET_2,&tbind2);
AbrirSocket(&fh,SPX_SOCKET_1,&tbind1);

/* Registramos la función a ejecutar cuando alguien desde la consola */
/* del servidor descarga el NLM con el comando UNLOAD o éste llama   */
/* a la función exit o ExitThread.                                   */

atexit(DespertarProcesos);

/* Obtiene el nombre del servidor para mandárselo a las estaciones */
/* que necesiten formar el camino completo del fichero que desean  */
/* mandar a imprimir.                                              */

GetFileServerName(0,NombreServidorFicheros);
printf("Nombre del Servidor: %s\n",NombreServidorFicheros);

/* Obtenemos la hora del Servidor de Ficheros y la almacenamos en */
/* el archivo de eventos diarios.                                 */

FechaActual=time(NULL);
sprintf(datos,"\n\n═══════════════════════════════════════════════════════\n");
EntradaDatosDiarios();
sprintf(datos,"Fecha del Servidor de Ficheros: ");
EntradaDatosDiarios();
strcpy(datos,ctime(&FechaActual));
EntradaDatosDiarios();

sprintf(datos,"\nNombre del Servidor de Ficheros: ");
EntradaDatosDiarios();
strcpy(datos,NombreServidorFicheros);
EntradaDatosDiarios();
sprintf(datos,"\n═══════════════════════════════════════════════════════\n\n");
EntradaDatosDiarios();

/* Entrada de los servidores de impresión remota */

if (BeginThread(PantallaMenu,NULL,NULL,0) == -1)
  {
    printf("BeginThread falló\n");
    exit(1);
  }
```

```
WaitOnLocalSemaphore(EntradaTeclado);
NumeroServImp = NumServImp;

for (Num=0;Num<NumServImp;Num++)
  {
    if (t_listen(fh,&tcall31)==-1) /* Escuchando candidatos de impresión */
        {
          t_error("t_listen");
          exit(1);
        }

    switch (Num)
        {
          case 0:

            if((fh_new=AceptarConeccion31Imp0())!=NULL) /* Proceso candidato impresión */
              {
                  if (BeginThread(ProcesarConeccion31Imp0,NULL,NULL,(void *)fh_new)==-1)
                    {
                      printf("BeginThread falló\n");
                      exit(1);
                    }
              }
            break;

          case 1:

            if((fh_new=AceptarConeccion31Imp1())!=NULL) /* Proceso candidato impresión */
              {
                  if (BeginThread(ProcesarConeccion31Imp1,NULL,NULL,(void *)fh_new)==-1)
                    {
                      printf("BeginThread falló\n");
                      exit(1);
                    }
              }
            break;

          case 2:

            if((fh_new=AceptarConeccion31Imp2())!=NULL) /* Proceso candidato impresión */
              {
                  if (BeginThread(ProcesarConeccion31Imp2,NULL,NULL,(void *)fh_new)==-1)
                    {
                      printf("BeginThread falló\n");
                      exit(1);
                    }
              }
            break;

        } /* Final de switch (Num) */

  } /* Final del for (Num=0;Num<NumServImp;Num++) */

cliente = 1;

for (;;)
```

```
    {

      if (t_listen(fh2,&tcall41)==-1) /* Escuchando peticiones de impresión */
          {
            t_error("t_listen 41");
            exit(1);
          }

      if((fh_new2 = AceptarConeccion41()) != NULL )
          {
            if (BeginThread(ProcesarConeccion41,NULL,NULL,(void *)fh_new2)==-1)
              {
                printf("BeginThread falló\n");
                exit(1);
              }
          }

      ThreadSwitch(); /* Cambio de contexto */

    } /* Final del for (;;) */
}     /* Final de main()    */

int AceptarConeccion31Imp0(void)
 {
   int fh_new;

   if((fh_new = t_open("/dev/nspx", O_RDWR , NULL)) == -1)
     {
        t_error("t_open 31");
        exit(1);
     }

   /* Enlazando, la dirección no es importante aquí */

   if( t_bind(fh_new, NULL, NULL) == -1 )
     {
        t_error("t_bind 31");
        t_close(fh_new);
        exit(1);
     }

   /* Aceptando la llamada desde un cliente */

   if( t_accept(fh, fh_new, &tcall31) == -1 )
     {
        if( t_errno == TLOOK )
          {
            printf("TLOOK 31\n");
            t_close(fh_new);
            return NULL;
          }
        else
          {
            t_error("t_accept 31");
            t_close(fh_new);
            exit(1);
```

```
            }
        }

    /* Chequeo para desconexión */

    ThreadSwitch();   /* Cambio de contexto */

    if( t_look(fh_new) == T_DISCONNECT )
      {
          printf("DISCONNECT  31\n");
          if( t_rcvdis(fh_new, NULL) < 0 )
            t_error("t_rcvdis 31");
          t_close(fh_new);
          return NULL;
      }

    return fh_new;

  }

void ProcesarConeccion31Imp0(void *parm)
 {

   FILE  *fp;
   char  iobuf[132];
   int   fh = (int) parm;
   int   AbrirFichero;   /* Un 0 si no se ha podido abrir el fichero */
   int   continua,handle;

   sprintf(datos,"Disponibles las siguientes impresoras\n");
   EntradaDatosDiarios();
   sprintf(datos,"=============================\n");
   EntradaDatosDiarios();
   sprintf(datos,"Servidor de impresión 1\n");
   EntradaDatosDiarios();
   sprintf(datos,"---------------------\n");
   EntradaDatosDiarios();

   continua=1;
   AbortarServImp0 = 0;

   /* Del Servidor de impresoras 0 se recibe un máximo de dos impresoras.*/

   while (continua == 1 )
     {
       if (t_rcv(fh,DirServ0Imp0,sizeof(DirServ0Imp0),NULL) != -1 )
           {
             if (strcmpi(DirServ0Imp0,"FIN") == 0)
               break;
             NumeroTotalImp++;
             ptr = DirServ0Imp0;
             SignalLocalSemaphore(SemPantalla);
             sprintf(datos,"%s",DirServ0Imp0);
             EntradaDatosDiarios();
             NumeroImp0=1;
           }
```

```
    if (t_rcv(fh,DirServ0Imp1,sizeof(DirServ0Imp1),NULL) != -1 )
        {
          if (strcmpi(DirServ0Imp1,"FIN") == 0)
            break;
          NumeroTotalImp++;
          ptr = DirServ0Imp1;
          SignalLocalSemaphore(SemPantalla);
          sprintf(datos,"%s",DirServ0Imp1);
          EntradaDatosDiarios();
          NumeroImp0=2;
        }

    if (t_rcv(fh,iobuf,sizeof(iobuf),NULL) != -1 )
        {
          if (strcmpi(iobuf,"FIN") == 0)
            break;
        }

  } /* Final del while (continua == 1 ) */

exito=0;
do
  {
    AbrirFichero=1;  /* Exito en la apertura del fichero */

    /* Dormimos el proceso hasta que llegue alguna petición de impresión */

    WaitOnLocalSemaphore(SemEsc0);

    /* Se manda la elección del trabajo a realizar por el Servidor Remoto */

    if (t_snd(fh,Eleccion0,sizeof(Eleccion0),NULL) == -1 )
        {
          ptr = "Error mandando elección de trabajo";
          SignalLocalSemaphore(SemPantalla);
        }

    if (strcmpi(Eleccion0,"BUSQUEDA") == 0)
        {
          /* Manda nombre de fichero y recibe mensaje */

          if( t_snd( fh, NombreFicheroBusca0, sizeof(NombreFicheroBusca0), NULL) == -1 )
            {
              ptr = "Se ha producido un error mandando el nombre del fichero a buscar";
              SignalLocalSemaphore(SemPantalla);
            }

          if (t_rcv(fh,LugarEncuentroFichero0,sizeof(LugarEncuentroFichero0),NULL) == -1 )
            {
              ptr = "Error recibiendo mensaje sobre búsqueda de fichero";
              SignalLocalSemaphore(SemPantalla);
            }
          strcpy(ptr,LugarEncuentroFichero0);
          SignalLocalSemaphore(SemPantalla);
          strcpy(LugarEncuentroFichero,LugarEncuentroFichero0);
```

```
        if( t_look(fh) == T_DISCONNECT )
          {
              ThreadSwitch();
              ptr = "Recibida petición de desconexión (Socket 31),";
              SignalLocalSemaphore(SemPantalla);
              ThreadSwitch();
              ptr = "por parte del Servidor de Impresión 0 EXIT.";
              SignalLocalSemaphore(SemPantalla);
              ThreadSwitch();
              strcpy(DirServ0Imp0,"Esta impresora ya no está disponible");
              strcpy(DirServ0Imp1,"Esta impresora ya no está disponible");
              AbortarServImp0=1;
              AbortarServImp = AbortarServImp0;
              if( t_rcvdis(fh, NULL) < 0 )
                {
                  ptr = "t_rcvdis 31";
                  SignalLocalSemaphore(SemPantalla);
                }
              t_close(fh);
          }

        SignalLocalSemaphore(AcaboLaBusqueda);

    }

if (strcmpi(Eleccion0,"IMPRESION") == 0)
    {

      /* Mandamos el Puerto correspondiente a la impresora elegida */

      if( t_snd( fh, Puerto, sizeof(Puerto), NULL) == -1 )
        {
          ptr = "Se ha producido un error mandando el puerto de la impresora";
          SignalLocalSemaphore(SemPantalla);
        }

      /* Comprobamos que el fichero no esté vacío */

      if (AbrirFichero == 1)
        {

          handle=open(FICHERO,O_RDONLY);

          if (filelength(handle) == 0)        /* El fichero está vacío */
              {
                ThreadSwitch();
                strcpy(Pantalla,"No se puede abrir el fichero ");
                strcat(Pantalla,FICHERO);
                strcat(Pantalla,", está vacío o no existe.");
                strcpy(ptr,Pantalla);
                strcpy(Pantalla,"    ");
                SignalLocalSemaphore(SemPantalla);
                delay(100);
                strcpy(iobufForCli,"No se puede abrir el fichero o no existe.");
                exito=0;
                Borrar=1;
```

```
            remove(FICHERO);
            SignalLocalSemaphore(SemMandarExito);
            AbrirFichero=0;
            strcpy(iobuf,"THE END");
            if( t_snd( fh, iobuf, sizeof(iobuf), NULL) == -1 )
              {
                ptr = "Se ha producido un error abortando";
                SignalLocalSemaphore(SemPantalla);
              }
          }

      close(handle);
    }

/* Ha llegado una petición de impresión por parte de un cliente */

if (AbrirFichero == 1)
  {
    if( (fp = fopen( FICHERO , "rt" )) == NULL )
        {
          ThreadSwitch();
          strcpy(Pantalla,"No se puede abrir el fichero ");
          strcat(Pantalla,FICHERO);
          strcat(Pantalla,", está vacío o no existe.");
          strcpy(ptr,Pantalla);
          strcpy(Pantalla,"   ");
          SignalLocalSemaphore(SemPantalla);
          delay(100);
          strcpy(iobufForCli,"No se puede abrir el fichero o no existe.");
          exito=0;
          SignalLocalSemaphore(SemMandarExito);
          AbrirFichero=0;
          strcpy(iobuf,"THE END");
          if( t_snd( fh, iobuf, sizeof(iobuf), NULL) == -1 )
            {
              ptr = "Se ha producido un error abortando";
              SignalLocalSemaphore(SemPantalla);
            }
        }
  }

/* Lee lineas del fichero y se las envía al servidor de impresión */

if (AbrirFichero == 1)
  {
    while( fgets( iobuf, sizeof(iobuf), fp ) != NULL )
        {
          delay(100);

          if( t_look(fh) == T_DISCONNECT )
            {
              ThreadSwitch();
              ptr = "Recibida petición de desconexión (Socket 31),";
              SignalLocalSemaphore(SemPantalla);
              ThreadSwitch();
              ptr = "por parte del Servidor de Impresión 0 EXIT.";
```

```
            SignalLocalSemaphore(SemPantalla);
            strcpy(DirServ0Imp0,"Esta impresora ya no está disponible");
            strcpy(DirServ0Imp1,"Esta impresora ya no está disponible");
            AbortarServImp0=1;
            AbortarServImp = AbortarServImp0;
            if( t_rcvdis(fh, NULL) < 0 )
               {
                 ptr = "t_rcvdis 31";
                 SignalLocalSemaphore(SemPantalla);
               }
            fclose(fp);
            t_close(fh);
            SignalLocalSemaphore(SemMandarExito);
            return;
          }

        if( t_snd( fh, iobuf, sizeof(iobuf), NULL) == -1 )
          {
            strcpy(iobufForCli,"No se ha podido mandar el fichero a la impresora");
            exito=0;
            SignalLocalSemaphore(SemMandarExito);
            ptr = "No se ha podido mandar el fichero a la impresora";
            SignalLocalSemaphore(SemPantalla);
            fclose(fp);
            t_close(fh);
            return;
          }
      } /* Final del while( fgets( DirServImp, sizeof(DirServImp), fp ) != NULL ) */

/* Le mandamos final de fichero para que haga un avance de página */

strcpy(iobuf,"THE END");

if( t_snd( fh, iobuf, sizeof(iobuf), NULL) == -1 )
   {
     ptr = "Se ha producido un error mandando final de fichero";
     SignalLocalSemaphore(SemPantalla);
   }
ThreadSwitch();
strcpy(Pantalla,"Se ha mandado a la impresora el fichero: ");
strcat(Pantalla,FICHERO);
strcpy(ptr,Pantalla);
SignalLocalSemaphore(SemPantalla);
ThreadSwitch();

exito=1;   /* Se ha realizado con éxito el envío */
             /* del fichero a la impresora.       */

SignalLocalSemaphore(SemMandarExito);
fclose(fp);

if (Borrar==1)  /* Borra el fichero temporal si éste fue creado */
   {
     remove(FICHERO);
     Borrar=0;
   }
```

```
            } /* Final de if (AbrirFichero = 1 ) */

          } /* Final de if (strcmpi(Eleccion,"IMPRESION") == 0) */

       if (kbhit())
           tecla = getch();

     } while (tecla != TECLAF1);

   /* Pulsando la tecla F1 en el servidor de ficheros, el servidor de */
   /* impresión remota concluirá */

   if( t_snddis(fh, NULL) == -1 )      /* Se envía la desconexión */
     {
       t_error("t_snddis 31");
       t_close(fh);
       return;
     }

   if( t_unbind(fh) == -1 )
     {
       t_error("t_unbind 31");
       t_close(fh);
       return;
     }

   t_close(fh);

  }


int AceptarConeccion31Impl(void)
  {

    int fh_new;

    if((fh_new = t_open("/dev/nspx", O_RDWR, NULL)) == -1)
      {
          t_error("t_open 31");
          exit(1);
      }

    /* Enlazando, la dirección no es importante aquí */

    if( t_bind(fh_new, NULL, NULL) == -1 )
      {
          t_error("t_bind 31");
          t_close(fh_new);
          exit(1);
      }

    /* Aceptando la llamada desde un cliente */

    if( t_accept(fh, fh_new, &tcall31) == -1 )
      {
          if( t_errno == TLOOK )
```

```
          {
            printf("TLOOK 31\n");
            t_close(fh_new);
            return NULL;
          }
        else
          {
            t_error("t_accept 31");
            t_close(fh_new);
            exit(1);
          }
      }

    /* Chequeo para desconexión */

    ThreadSwitch();   /* Cambio de contexto */

    if( t_look(fh_new) == T_DISCONNECT )
      {
         printf("DISCONNECT  31\n");
         if( t_rcvdis(fh_new, NULL) < 0 )
           t_error("t_rcvdis 31");
         t_close(fh_new);
         return NULL;
      }

    return fh_new;

  }

void ProcesarConeccion31Impl(void *parm)
 {

   FILE  *fp;
   char  iobuf[132];
   int   fh = (int) parm;
   int   AbrirFichero;   /* Un 0 si no se ha podido abrir el fichero */
   int   continua,handle;

   strcpy(datos,"Servidor de impresión 2");
   EntradaDatosDiarios();
   strcpy(datos,"————————————————————");
   EntradaDatosDiarios();
   continua=1;
   AbortarServImpl = 0;

   /* Del Servidor de impresoras 0 se recibe un máximo de dos impresoras.*/

   while (continua == 1 )
     {
       if (t_rcv(fh,DirServ1Impo,sizeof(DirServ1Impo),NULL) != -1 )
          {
            if (strcmpi(DirServ1Impo,"FIN") == 0)
              break;
            NumeroTotalImp++;
            ptr = DirServ1Impo;
```

```
          SignalLocalSemaphore(SemPantalla);
          sprintf(datos,"%s",DirServ1Imp0);
          EntradaDatosDiarios();
          NumeroImp1=1;
        }

    if (t_rcv(fh,DirServ1Imp1,sizeof(DirServ1Imp1),NULL) != -1 )
        {
          if (strcmpi(DirServ1Imp1,"FIN") == 0)
            break;
          NumeroTotalImp++;
          ptr = DirServ1Imp1;
          SignalLocalSemaphore(SemPantalla);
          sprintf(datos,"%s",DirServ1Imp1);
          EntradaDatosDiarios();
          NumeroImp1=2;
        }

    if (t_rcv(fh,iobuf,sizeof(iobuf),NULL) != -1 )
        {
          if (strcmpi(iobuf,"FIN") == 0)
            break;
        }

  }  /* Final del while (continua == 1 ) */

exito=0;
do
  {
    AbrirFichero=1;  /* Exito en la apertura del fichero */

    /* Dormimos el proceso hasta que llegue alguna petición de impresión */

    WaitOnLocalSemaphore(SemEsc1);

    /* Se manda la elección del trabajo a realizar por el Servidor Remoto */

    if (t_snd(fh,Eleccion1,sizeof(Eleccion1),NULL) == -1 )
        {
          ptr = "Error mandando elección de trabajo";
          SignalLocalSemaphore(SemPantalla);
        }

    if (strcmpi(Eleccion1,"BUSQUEDA") == 0)
        {

          /* Manda nombre de fichero y recibe mensaje */

          if( t_snd( fh, NombreFicheroBusca1, sizeof(NombreFicheroBusca1), NULL) == -1 )
            {
              ptr = "Se ha producido un error mandando el nombre del fichero a buscar";
              SignalLocalSemaphore(SemPantalla);
            }

          if (t_rcv(fh,LugarEncuentroFichero1,sizeof(LugarEncuentroFichero1),NULL) == -1 )
            {
```

```
            ptr = "Error recibiendo mensaje sobre búsqueda de fichero";
            SignalLocalSemaphore(SemPantalla);
          }
        strcpy(ptr,LugarEncuentroFichero1);
        SignalLocalSemaphore(SemPantalla);
        strcpy(LugarEncuentroFichero,LugarEncuentroFichero1);

        if( t_look(fh) == T_DISCONNECT )
          {
            ptr = "Recibida petición de desconexión (Socket 31),";
            SignalLocalSemaphore(SemPantalla);
            ptr = "por parte del Servidor de Impresión 1 EXIT.";
            SignalLocalSemaphore(SemPantalla);
            strcpy(DirServ1Imp0,"Esta impresora ya no está disponible");
            strcpy(DirServ1Imp1,"Esta impresora ya no está disponible");
            AbortarServImp1=1;
            AbortarServImp = AbortarServImp1;
            if( t_rcvdis(fh, NULL) < 0 )
              {
                ptr = "t_rcvdis 31";
                SignalLocalSemaphore(SemPantalla);
              }
            t_close(fh);
          }

        SignalLocalSemaphore(AcaboLaBusqueda);

    }

if (strcmpi(Eleccion1,"IMPRESION") == 0)
    {

      /* Mandamos el Puerto correspondiente a la impresora elegída */

      if( t_snd( fh, Puerto, sizeof(Puerto), NULL) == -1 )
        {
          ptr = "Se ha producido un error mandando el puerto de la impresora";
          SignalLocalSemaphore(SemPantalla);
        }

      /* Comprobamos que el fichero no esté vacío */

      if (AbrirFichero == 1)
        {

          handle=open(FICHERO,O_RDONLY);

          if (filelength(handle) == 0)        /* El fichero está vacío */
              {
                strcpy(Pantalla,"No se puede abrir el fichero ");
                strcat(Pantalla,FICHERO);
                strcat(Pantalla,", está vacío o no existe.");
                strcpy(ptr,Pantalla);
                SignalLocalSemaphore(SemPantalla);
                strcpy(iobufForCli,"No se puede abrir el fichero o no existe.");
                exito=0;
```

```
        Borrar=1;
        remove(FICHERO);
        SignalLocalSemaphore(SemMandarExito);
        AbrirFichero=0;
        strcpy(iobuf,"THE END");
        if( t_snd( fh, iobuf, sizeof(iobuf), NULL) == -1 )
          {
            ptr = "Se ha producido un error abortando";
            SignalLocalSemaphore(SemPantalla);
          }
      }

   close(handle);
  }

/* Ha llegado una petición de impresión por parte de un cliente */

if (AbrirFichero == 1)
  {
   if( (fp = fopen( FICHERO , "rt" )) == NULL )
       {

        strcpy(Pantalla,"No se puede abrir el fichero ");
        strcat(Pantalla,FICHERO);
        strcat(Pantalla," o no existe.");
        strcpy(ptr,Pantalla);
        SignalLocalSemaphore(SemPantalla);
        strcpy(iobufForCli,"No se puede abrir el fichero o no existe.");
        exito=0;
        SignalLocalSemaphore(SemMandarExito);
        AbrirFichero=0;
        strcpy(iobuf,"THE END");
        if( t_snd( fh, iobuf, sizeof(iobuf), NULL) == -1 )
          {
            ptr = "Se ha producido un error abortando";
            SignalLocalSemaphore(SemPantalla);
          }
      }
  }

/* Lee líneas del fichero y se las envía al servidor de impresión */

if (AbrirFichero == 1)
  {
   while( fgets( iobuf, sizeof(iobuf), fp ) != NULL )
       {
        delay(100);

        if( t_look(fh) == T_DISCONNECT )
          {
            ptr = "Recibida petición de desconexión (Socket 31),";
            SignalLocalSemaphore(SemPantalla);
            ptr = "por parte del Servidor de Impresión 1 EXIT.";
            SignalLocalSemaphore(SemPantalla);
            strcpy(DirServ1Imp0,"Esta impresora ya no está disponible");
            strcpy(DirServ1Imp1,"Esta impresora ya no está disponible");
```

```
                AbortarServImp1=1;
                AbortarServImp = AbortarServImp1;
                if( t_rcvdis(fh, NULL) < 0 )
                    {
                      ptr = "t_rcvdis 31";
                      SignalLocalSemaphore(SemPantalla);
                    }
                fclose(fp);
                t_close(fh);
                SignalLocalSemaphore(SemMandarExito);
                return;
              }

            if( t_snd( fh, iobuf, sizeof(iobuf), NULL) == -1 )
              {
                strcpy(iobufForCli,"No se ha podido mandar el fichero a la impresora");
                exito=0;
                SignalLocalSemaphore(SemMandarExito);
                ptr = "No se ha podido mandar el fichero a la impresora";
                SignalLocalSemaphore(SemPantalla);
                fclose(fp);
                t_close(fh);
                return;
              }
          } /* Final del while( fgets( DirServImp, sizeof(DirServImp), fp ) != NULL ) */

        /* Le mandamos final de fichero para que haga un avance de página */

        strcpy(iobuf,"THE END");

        if( t_snd( fh, iobuf, sizeof(iobuf), NULL) == -1 )
            {
              ptr = "Se ha producido un error mandando final de fichero";
              SignalLocalSemaphore(SemPantalla);
            }

        strcpy(Pantalla,"Se ha mandado a la impresora el fichero: ");
        strcat(Pantalla,FICHERO);
        strcpy(ptr,Pantalla);
        SignalLocalSemaphore(SemPantalla);

        exito=1;   /* Se ha realizado con éxito el envío */
                   /* del fichero a la impresora.       */

        SignalLocalSemaphore(SemMandarExito);
        fclose(fp);

        if (Borrar==1)  /* Borra el fichero temporal si éste fue creado */
            {
              remove(FICHERO);
              Borrar=0;
            }

      } /* Final de if (AbrirFichero == 1 ) */

    } /* Final de if (strcmpi(Eleccion,"IMPRESION") == 0) */
```

```
      if (kbhit())
          tecla = getch();

    } while (tecla != TECLAF1);

  /* Pulsando la tecla F1 en el servidor de ficheros, el servidor de */
  /* impresión remota concluirá */

  if( t_snddis(fh, NULL) == -1 )      /* Se envía la desconexión */
    {
      t_error("t_snddis 31");
      t_close(fh);
      return;
    }

  if( t_unbind(fh) == -1 )
    {
      t_error("t_unbind 31");
      t_close(fh);
      return;
    }

  t_close(fh);

 }

int AceptarConeccion31Imp2(void)
 {

   int fh_new;

   if((fh_new = t_open("/dev/nspx", O_RDWR, NULL)) == -1)
     {
        t_error("t_open 31");
        exit(1);
     }

   /* Enlazando, la dirección no es importante aquí */

   if( t_bind(fh_new, NULL, NULL) == -1 )
     {
        t_error("t_bind 31");
        t_close(fh_new);
        exit(1);
     }

   /* Aceptando la llamada desde un cliente */

   if( t_accept(fh, fh_new, &tcall31) == -1 )
     {
        if( t_errno == TLOOK )
          {
            printf("TLOOK 31\n");
            t_close(fh_new);
            return NULL;
          }
```

```
        else
          {
            t_error("t_accept 31");
            t_close(fh_new);
            exit(1);
          }
      }

    /* Chequeo para desconexión */

    ThreadSwitch();   /* Cambio de contexto */

    if( t_look(fh_new) == T_DISCONNECT )
      {
          printf("DISCONNECT  31\n");
          if( t_rcvdis(fh_new, NULL) < 0 )
            t_error("t_rcvdis 31");
          t_close(fh_new);
          return NULL;
      }

    return fh_new;

  }

void ProcesarConeccion31Imp2(void *parm)
 {

   FILE  *fp;
   char  iobuf[132];
   int   fh = (int) parm;
   int   AbrirFichero;   /* Un 0 si no se ha podido abrir el fichero */
   int   continua,handle;

   strcpy(datos,"Servidor de impresión 3");
   EntradaDatosDiarios();
   strcpy(datos,"----------------------");
   EntradaDatosDiarios();
   continua=1;
   AbortarServImp2 = 0;

   /* Del Servidor de impresoras 2 se recibe un máximo de dos impresoras.*/

   while (continua == 1 )
     {
       if (t_rcv(fh,DirServ2Imp0,sizeof(DirServ2Imp0),NULL) != -1 )
           {
             if (strcmpi(DirServ2Imp0,"FIN") == 0)
               break;
             NumeroTotalImp++;
             ptr = DirServ2Imp0;
             SignalLocalSemaphore(SemPantalla);
             sprintf(datos,"%s",DirServ2Imp0);
             EntradaDatosDiarios();
             NumeroImp2 = 1;
           }
```

```
    if (t_rcv(fh,DirServ2Imp1,sizeof(DirServ2Imp1),NULL) != -1 )
        {
          if (strcmpi(DirServ2Imp1,"FIN") == 0)
            break;
          NumeroTotalImp++;
          ptr = DirServ2Imp1;
          SignalLocalSemaphore(SemPantalla);
          sprintf(datos,"%s",DirServ2Imp1);
          EntradaDatosDiarios();
          NumeroImp2 = 2;
        }

    if (t_rcv(fh,iobuf,sizeof(iobuf),NULL) != -1 )
        {
          if (strcmpi(iobuf,"FIN") == 0)
            break;
        }

  }  /* Final del while (continua == 1 ) */

exito=0;
do
  {
    AbrirFichero=1;  /* Exito en la apertura del fichero */

    /* Dormimos el proceso hasta que llegue alguna petición de impresión */

    WaitOnLocalSemaphore(SemEsc2);

    /* Se recibe la elección del trabajo a realizar por el Servidor Remoto */

    if (t_snd(fh,Eleccion2,sizeof(Eleccion2),NULL) == -1 )
        {
          ptr = "Error mandando elección de trabajo";
          SignalLocalSemaphore(SemPantalla);
        }

    if (strcmpi(Eleccion2,"BUSQUEDA") == 0)
        {

          /* Manda nombre de fichero y recibe mensaje */

          if( t_snd( fh, NombreFicheroBusca2, sizeof(NombreFicheroBusca2), NULL) == -1 )
            {
                ptr = "Se ha producido un error mandando el nombre del fichero a buscar";
                SignalLocalSemaphore(SemPantalla);
            }

          if (t_rcv(fh,LugarEncuentroFichero2,sizeof(LugarEncuentroFichero2),NULL) == -1 )
            {
                ptr = "Error recibiendo mensaje sobre búsqueda de fichero";
                SignalLocalSemaphore(SemPantalla);
            }
          strcpy(ptr,LugarEncuentroFichero2);
          SignalLocalSemaphore(SemPantalla);
          strcpy(LugarEncuentroFichero,LugarEncuentroFichero2);
```

```
        if( t_look(fh) == T_DISCONNECT )
          {
              ptr = "Recibida petición de desconexión (Socket 31),";
              SignalLocalSemaphore(SemPantalla);
              ptr = "por parte del Servidor de Impresión 2 EXIT.";
              SignalLocalSemaphore(SemPantalla);
              strcpy(DirServ2Imp0,"Esta impresora ya no está disponible");
              strcpy(DirServ2Imp1,"Esta impresora ya no está disponible");
              AbortarServImp2=1;
              AbortarServImp = AbortarServImp2;
              if( t_rcvdis(fh, NULL) < 0 )
                {
                  ptr = "t_rcvdis 31";
                  SignalLocalSemaphore(SemPantalla);
                }
              t_close(fh);
          }

        SignalLocalSemaphore(AcaboLaBusqueda);

    }

if (strcmpi(Eleccion2,"IMPRESION") == 0)
    {

      /* Mandamos el Puerto correspondiente a la impresora elegída */

      if( t_snd( fh, Puerto, sizeof(Puerto), NULL) == -1 )
        {
          ptr = "Se ha producido un error mandando el puerto de la impresora";
          SignalLocalSemaphore(SemPantalla);
        }

      /* Comprobamos que el fichero no esté vacío */

      if (AbrirFichero == 1)
        {

          handle=open(FICHERO,O_RDONLY);

          if (filelength(handle) == 0)        /* El fichero está vacío */
              {
                strcpy(Pantalla,"No se puede abrir el fichero, ");
                strcat(Pantalla,FICHERO);
                strcat(Pantalla," está vacío o no existe.");
                strcpy(ptr,Pantalla);
                SignalLocalSemaphore(SemPantalla);
                strcpy(iobufForCli,"No se puede abrir el fichero o no existe.");
                exito=0;
                Borrar=1;
                remove(FICHERO);
                SignalLocalSemaphore(SemMandarExito);
                AbrirFichero = 0;
                strcpy(iobuf,"THE END");
                if( t_snd( fh, iobuf, sizeof(iobuf), NULL) == -1 )
                  {
```

```
            ptr = "Se ha producido un error abortando";
            SignalLocalSemaphore(SemPantalla);
          }
      }

    close(handle);
  }

/* Ha llegado una petición de impresión por parte de un cliente */

if (AbrirFichero == 1)
  {
    if( (fp = fopen( FICHERO , "rt" )) == NULL )
        {
          strcpy(Pantalla,"No se puede abrir el fichero ");
          strcat(Pantalla,FICHERO);
          strcat(Pantalla," o no existe.");
          strcpy(ptr,Pantalla);
          SignalLocalSemaphore(SemPantalla);
          strcpy(iobufForCli,"No se puede abrir el fichero o no existe.");
          exito=0;
          SignalLocalSemaphore(SemMandarExito);
          AbrirFichero = 0;
          strcpy(iobuf,"THE END");
          if( t_snd( fh, iobuf, sizeof(iobuf), NULL) == -1 )
            {
              ptr = "Se ha producido un error abortando";
              SignalLocalSemaphore(SemPantalla);
            }
        }
  }

/* Lee líneas del fichero y se las envía al servidor de impresión */

if (AbrirFichero == 1)
  {
    while( fgets( iobuf, sizeof(iobuf), fp ) != NULL )
        {
          delay(100);

          if( t_look(fh) == T_DISCONNECT )
            {
              ptr = "Recibida petición de desconexión (Socket 31),";
              SignalLocalSemaphore(SemPantalla);
              ptr = "por parte del Servidor de Impresión 2 EXIT.";
              SignalLocalSemaphore(SemPantalla);
              strcpy(DirServ2Imp0,"Esta impresora ya no está disponible");
              strcpy(DirServ2Imp1,"Esta impresora ya no está disponible");
              AbortarServImp2 = 1;
              AbortarServImp = AbortarServImp2;
              if( t_rcvdis(fh, NULL) < 0 )
                  {
                    ptr = "t_rcvdis 31";
                    SignalLocalSemaphore(SemPantalla);
                  }
              fclose(fp);
```

```
                    t_close(fh);
                    SignalLocalSemaphore(SemMandarExito);
                    return;
                  }

                if( t_snd( fh, iobuf, sizeof(iobuf), NULL) == -1 )
                  {
                    strcpy(iobufForCli,"No se ha podido mandar el fichero a la impresora");
                    exito=0;
                    SignalLocalSemaphore(SemMandarExito);
                    ptr = "No se ha podido mandar el fichero a la impresora";
                    SignalLocalSemaphore(SemPantalla);
                    fclose(fp);
                    t_close(fh);
                    return;
                  }
              }  /* Final del while( fgets( DirServImp, sizeof(DirServImp), fp ) != NULL ) */

            /* Le mandamos final de fichero para que haga un avance de página */

            strcpy(iobuf,"THE END");

            if( t_snd( fh, iobuf, sizeof(iobuf), NULL) == -1 )
                {
                  ptr = "Se ha producido un error mandando final de fichero";
                  SignalLocalSemaphore(SemPantalla);
                }

            strcpy(Pantalla,"Se ha mandado a la impresora el fichero: ");
            strcat(Pantalla,FICHERO);
            strcpy(ptr,Pantalla);
            SignalLocalSemaphore(SemPantalla);

            exito = 1;   /* Se ha realizado con éxito el envío */
                         /* del fichero a la impresora.       */

            SignalLocalSemaphore(SemMandarExito);
            fclose(fp);

            if (Borrar==1)  /* Borra el fichero temporal si éste fue creado */
                {
                  remove(FICHERO);
                  Borrar=0;
                }

          } /* Final de if (AbrirFichero == 1 ) */

        } /* Final de if (strcmpi(Eleccion,"IMPRESION") == 0) */

    if (kbhit())
        tecla = getch();

  } while (tecla != TECLAF1);

/* Pulsando cualquier tecla en el servidor de ficheros, el servidor de */
/* impresión remota concluirá */
```

```
   if( t_snddis(fh, NULL) == -1 )      /* Se envía la desconexión */
     {
       t_error("t_snddis 31");
       t_close(fh);
       return;
     }

   if( t_unbind(fh) == -1 )
     {
       t_error("t_unbind 31");
       t_close(fh);
       return;
     }

   t_close(fh);

  }

void AbrirSocket(int *fd,unsigned char SPX_SOCKET,struct t_bind *tbind)
  {

   IPX_ADDR        spx_addr;

   /* Abrimos un extremo de conexión, no es necesaria la información */

   if((*fd = t_open("/dev/nspx", O_RDWR , NULL)) == -1)
     {
       t_error("t_open");
       exit(1);
     }

   /* Inicializamos la estructura tbind y llamamos a t_bind para enlazar */

   spx_addr.ipxa_socket[0] = 0;
   spx_addr.ipxa_socket[1] = SPX_SOCKET;

   tbind->qlen = 5;
   tbind->addr.len = sizeof( spx_addr );
   tbind->addr.buf = (char *)&spx_addr;
   tbind->addr.maxlen = sizeof (spx_addr);

   if( t_bind(*fd, tbind, tbind) == -1 )
     {
       t_error("t_bind");
       exit(1);
     }

  }

int AceptarConeccion41(void)
  {

   int    fh_new2;

   if((fh_new2 = t_open("/dev/nspx", O_RDWR, NULL)) == -1)
     {
```

```
      t_error("t_open 41");
      exit(1);
    }

  /* Enlazamos, la dirección no es importante aquí */

  if( t_bind(fh_new2, NULL, NULL) == -1 )
    {
      t_error("t_bind 41");
      t_close(fh_new2);
      exit(1);
    }

  /* Aceptamos la llamada desde un cliente */

  if( t_accept(fh2, fh_new2, &tcall41) == -1 )
    {
      if( t_errno == TLOOK )
          {
            printf("TLOOK 41\n");
            t_close(fh_new2);
            return NULL;
          }
      else
          {
            t_error("t_accept 41");
            t_close(fh_new2);
            exit(1);
          }
    }

  /* Chequeamos para desconectar */

  ThreadSwitch();      /* Cambio de contexto */

  if( t_look(fh_new2) == T_DISCONNECT )
    {
      printf("DISCONNECT 41\n");
      if( t_rcvdis(fh_new2, NULL) < 0 )
          t_error("t_rcvdis");
      t_close(fh_new2);
      return NULL;
    }

  ptr = "------------------------------ CLIENTE ------------------------------";
  SignalLocalSemaphore(SemPantalla);
  sprintf(datos,"Cliente con petición de impresión\n");
  EntradaDatosDiarios();
  return fh_new2;

  }

/* Conexión con el cliente que desea imprimir */

void ProcesarConeccion41(void *parm)
  {
```

```
char   FICH[50],unidad[50];
char   NumeroImpresoras[70];
FILE   *fp1;
char   iobuf[132],DatosConeccion[132];
int    fh2 = (int) parm;
int    i;

i=NumServImp;   /* Número de Servidores de impresión */


if (NumServImp==1)
  strcpy(NumeroImpresoras,"1");
if (NumServImp==2)
  strcpy(NumeroImpresoras,"2");
if (NumServImp==3)
  strcpy(NumeroImpresoras,"3");

/* Se reciben los datos de la conexión */

if (t_rcv(fh2,DatosConeccion,sizeof(DatosConeccion),NULL) == -1 )
  {
     t_close(fh2);
     return;
  }

/* Se guarda información en un archivo */

sprintf(datos,"%s\n",DatosConeccion);
EntradaDatosDiarios();
ptr = DatosConeccion;
SignalLocalSemaphore(SemPantalla);

/* Se manda el número de impresoras disponibles */

if( t_snd(fh2,NumeroImpresoras,sizeof(NumeroImpresoras),NULL) == -1)
  {
     ptr = "Se ha producido un error enviando la cantidad de impresoras disponibles";
     SignalLocalSemaphore(SemPantalla);
     t_close(fh2);
     return;
  }

/* Se mandan las impresora disponibles */

if (i==1)  /* Servidor de impresoras 0 */
  {
     if (NumeroImp0 == 1)  /* Una sola impresora */
       {
         /* Se manda la dirección de la impresora 0 */

         if( t_snd( fh2, DirServ0Imp0 , sizeof(DirServ0Imp0), NULL) == -1 )
           {
              ptr = "t_snd 41";
              SignalLocalSemaphore(SemPantalla);
              t_close(fh2);
              return;
```

```
          }

        /* Se comunica que no hay más */

        strcpy(iobuf,"FIN");

        if( t_snd( fh2, iobuf , sizeof(iobuf), NULL) == -1 )
          {
              ptr = "t_snd 41";
              SignalLocalSemaphore(SemPantalla);
              t_close(fh2);
              return;
          }
      }

    if (NumeroImp0 == 2)  /* Dos impresoras */
      {
        /* Se mandan las direcciones de las dos impresoras */

        if( t_snd( fh2, DirServ0Imp0 , sizeof(DirServ0Imp0), NULL) == -1 )
          {
              ptr = "t_snd 41";
              SignalLocalSemaphore(SemPantalla);
              t_close(fh2);
              return;
          }

        if( t_snd( fh2, DirServ0Imp1 , sizeof(DirServ0Imp1), NULL) == -1 )
          {
              ptr = "t_snd 41";
              SignalLocalSemaphore(SemPantalla);
              t_close(fh2);
              return;
          }

        /* Se comunica que no hay más impresoras */

        strcpy(iobuf,"FIN");

        if( t_snd( fh2, iobuf , sizeof(iobuf), NULL) == -1 )
          {
              ptr = "t_snd 41";
              SignalLocalSemaphore(SemPantalla);
              t_close(fh2);
              return;
          }
      }

  }

if (i==2)   /* Servidor de impresoras 0 y 1 */
  {

    /* Servidor de impresoras 0 */

    if (NumeroImp0 == 1)  /* Una sola impresora */
```

```
  {
    /* Se manda la dirección de la impresora 0 */

    if( t_snd( fh2, DirServ0Imp0 , sizeof(DirServ0Imp0), NULL) == -1 )
      {
          ptr = "t_snd 41";
          SignalLocalSemaphore(SemPantalla);
          t_close(fh2);
          return;
      }

    /* Se comunica que no hay más */

    strcpy(iobuf,"FIN");

    if( t_snd( fh2, iobuf , sizeof(iobuf), NULL) == -1 )
      {
          ptr = "t_snd 41";
          SignalLocalSemaphore(SemPantalla);
          t_close(fh2);
          return;
      }
  }

if (NumeroImp0 == 2)  /* Dos impresoras */
  {
    /* Se mandan las direcciones de las dos impresoras */

    if( t_snd( fh2, DirServ0Imp0 , sizeof(DirServ0Imp0), NULL) == -1 )
      {
          ptr = "t_snd 41";
          SignalLocalSemaphore(SemPantalla);
          t_close(fh2);
          return;
      }

    if( t_snd( fh2, DirServ0Imp1 , sizeof(DirServ0Imp1), NULL) == -1 )
      {
          ptr = "t_snd 41";
          SignalLocalSemaphore(SemPantalla);
          t_close(fh2);
          return;
      }

    /* Se comunica que no hay más impresoras */

    strcpy(iobuf,"FIN");

    if( t_snd( fh2, iobuf , sizeof(iobuf), NULL) == -1 )
      {
          ptr = "t_snd 41";
          SignalLocalSemaphore(SemPantalla);
          t_close(fh2);
          return;
      }
  }
```

```
/* Servidor de impresoras 1 */

if (NumeroImp1 == 1)  /* Una sola impresora */
  {
    /* Se manda la dirección de la impresora 0 */

    if( t_snd( fh2, DirServ1Imp0 , sizeof(DirServ1Imp0), NULL) == -1 )
      {
          ptr = "t_snd 41";
          SignalLocalSemaphore(SemPantalla);
          t_close(fh2);
          return;
      }

    /* Se comunica que no hay más */

    strcpy(iobuf,"FIN");

    if( t_snd( fh2, iobuf , sizeof(iobuf), NULL) == -1 )
      {
          ptr = "t_snd 41";
          SignalLocalSemaphore(SemPantalla);
          t_close(fh2);
          return;
      }
  }

if (NumeroImp1 == 2)  /* Dos impresoras */
  {
    /* Se mandan las direcciones de las dos impresoras */

    if( t_snd( fh2, DirServ1Imp0 , sizeof(DirServ1Imp0), NULL) == -1 )
      {
          ptr = "t_snd 41";
          SignalLocalSemaphore(SemPantalla);
          t_close(fh2);
          return;
      }

    if( t_snd( fh2, DirServ1Imp1 , sizeof(DirServ1Imp1), NULL) == -1 )
      {
          ptr = "t_snd 41";
          SignalLocalSemaphore(SemPantalla);
          t_close(fh2);
          return;
      }

    /* Se comunica que no hay más impresoras */

    strcpy(iobuf,"FIN");

    if( t_snd( fh2, iobuf , sizeof(iobuf), NULL) == -1 )
      {
          ptr = "t_snd 41";
          SignalLocalSemaphore(SemPantalla);
          t_close(fh2);
```

```
            return;
          }
      }

  }

if (i==3)  /* Servidor de impresoras 0, 1 y 2 */
  {

     /* Servidor de impresoras 0 */

     if (NumeroImp0 == 1)  /* Una sola impresora */
       {
         /* Se manda la dirección de la impresora 0 */

         if( t_snd( fh2, DirServ0Imp0 , sizeof(DirServ0Imp0), NULL) == -1 )
           {
             ptr = "t_snd 41";
             SignalLocalSemaphore(SemPantalla);
             t_close(fh2);
             return;
           }

         /* Se comunica que no hay más */

         strcpy(iobuf,"FIN");

         if( t_snd( fh2, iobuf , sizeof(iobuf), NULL) == -1 )
           {
             ptr = "t_snd 41";
             SignalLocalSemaphore(SemPantalla);
             t_close(fh2);
             return;
           }
       }

     if (NumeroImp0 == 2)  /* Dos impresoras */
       {
         /* Se mandan las direcciones de las dos impresoras */

         if( t_snd( fh2, DirServ0Imp0 , sizeof(DirServ0Imp0), NULL) == -1 )
           {
             ptr = "t_snd 41";
             SignalLocalSemaphore(SemPantalla);
             t_close(fh2);
             return;
           }

         if( t_snd( fh2, DirServ0Imp1 , sizeof(DirServ0Imp1), NULL) == -1 )
           {
             ptr = "t_snd 41";
             SignalLocalSemaphore(SemPantalla);
             t_close(fh2);
             return;
           }
```

```
    /* Se comunica que no hay más impresoras */

    strcpy(iobuf,"FIN");

    if( t_snd( fh2, iobuf , sizeof(iobuf), NULL) == -1 )
      {
          ptr = "t_snd 41";
          SignalLocalSemaphore(SemPantalla);
          t_close(fh2);
          return;
      }
  }

/* Servidor de impresoras 1 */

if (NumeroImp1 == 1)  /* Una sola impresora */
  {
    /* Se manda la dirección de la impresora 0 */

    if( t_snd( fh2, DirServ1Imp0 , sizeof(DirServ1Imp0), NULL) == -1 )
      {
          ptr = "t_snd 41";
          SignalLocalSemaphore(SemPantalla);
          t_close(fh2);
          return;
      }

    /* Se comunica que no hay más */

    strcpy(iobuf,"FIN");

    if( t_snd( fh2, iobuf , sizeof(iobuf), NULL) == -1 )
      {
          ptr = "t_snd 41";
          SignalLocalSemaphore(SemPantalla);
          t_close(fh2);
          return;
      }
  }

if (NumeroImp1 == 2)  /* Dos impresoras */
  {
    /* Se mandan las direcciones de las dos impresoras */

    if( t_snd( fh2, DirServ1Imp0 , sizeof(DirServ1Imp0), NULL) == -1 )
      {
          ptr = "t_snd 41";
          SignalLocalSemaphore(SemPantalla);
          t_close(fh2);
          return;
      }

    if( t_snd( fh2, DirServ1Imp1 , sizeof(DirServ1Imp1), NULL) == -1 )
      {
          ptr = "t_snd 41";
          SignalLocalSemaphore(SemPantalla);
```

```
            t_close(fh2);
            return;
        }

    /* Se comunica que no hay más impresoras */

    strcpy(iobuf,"FIN");

    if( t_snd( fh2, iobuf , sizeof(iobuf), NULL) == -1 )
        {
            ptr = "t_snd 41";
            SignalLocalSemaphore(SemPantalla);
            t_close(fh2);
            return;
        }
  }

/* Servidor de impresoras 2 */

if (NumeroImp2 == 1)  /* Una sola impresora */
  {
    /* Se manda la dirección de la impresora 0 */

    if( t_snd( fh2, DirServ2Imp0 , sizeof(DirServ2Imp0), NULL) == -1 )
        {
            ptr = "t_snd 41";
            SignalLocalSemaphore(SemPantalla);
            t_close(fh2);
            return;
        }

    /* Se comunica que no hay más */

    strcpy(iobuf,"FIN");

    if( t_snd( fh2, iobuf , sizeof(iobuf), NULL) == -1 )
        {
            ptr = "t_snd 41";
            SignalLocalSemaphore(SemPantalla);
            t_close(fh2);
            return;
        }
  }

if (NumeroImp2 == 2)  /* Dos impresoras */
  {
    /* Se mandan las direcciones de las dos impresoras */

    if( t_snd( fh2, DirServ2Imp0 , sizeof(DirServ2Imp0), NULL) == -1 )
        {
            ptr = "t_snd 41";
            SignalLocalSemaphore(SemPantalla);
            t_close(fh2);
            return;
        }
```

```
        if( t_snd( fh2, DirServ2Impl , sizeof(DirServ2Impl), NULL) == -1 )
          {
              ptr = "t_snd 41";
              SignalLocalSemaphore(SemPantalla);
              t_close(fh2);
              return;
          }

        /* Se comunica que no hay más impresoras */

        strcpy(iobuf,"FIN");

        if( t_snd( fh2, iobuf , sizeof(iobuf), NULL) == -1 )
          {
              ptr = "t_snd 41";
              SignalLocalSemaphore(SemPantalla);
              t_close(fh2);
              return;
          }
      }

  }

/* Se manda el nombre de Servidor */

if (t_snd(fh2,NombreServidorFicheros,sizeof(NombreServidorFicheros),NULL)==-1)
  {
      ptr = "Se ha producido un error mandando el nombre del Servidor";
      SignalLocalSemaphore(SemPantalla);
      ThreadSwitch();
      t_close(fh2);
      return;
  }

/* Recibiendo el nombre de la impresora elegida */

if (t_rcv(fh2,DirServImp,sizeof(DirServImp),NULL) != -1 )
  {
      ThreadSwitch();
      strcpy(Pantalla,"Elección: ");
      strcat(Pantalla,DirServImp);
      strcpy(ptr,Pantalla);
      SignalLocalSemaphore(SemPantalla);
  }
else
  {
      ptr = "Error recibiendo el nombre de la impresora";
      SignalLocalSemaphore(SemPantalla);
  }

strcpy(iobuf,DirServImp);
strrev(iobuf);  /* Invertimos la cadena */

/* Obtenemos el puerto correspondiente a la impresora elegida */

if (iobuf[1] == '1')
```

```
  strcpy(Puerto,"LPT1");
else
  if (iobuf[1] == '2')
      strcpy(Puerto,"LPT2");
  else
      {
        ptr = "Se ha detectado un error en los puertos";
        SignalLocalSemaphore(SemPantalla);
      }

/* Se recibe del cliente la elección de la tarea: búsqueda o impresión */

if (t_rcv(fh2,Eleccion,sizeof(Eleccion),NULL) == -1 )
  {
      ptr = "Error recibiendo la elección de la tarea";
      SignalLocalSemaphore(SemPantalla);
  }


if (strcmpi(Eleccion,"BUSQUEDA") == 0)
  {
      /* Se recibe el nombre del fichero a buscar */

      ptr = "Se ha elegido la opción de BUSQUEDA REMOTA";
      ThreadSwitch();
      SignalLocalSemaphore(SemPantalla);


      if (t_rcv(fh2,NombreFicheroBusca,sizeof(NombreFicheroBusca),NULL) == -1 )
        {
          ptr = "Error recibiendo el nombre del fichero a buscar";
          SignalLocalSemaphore(SemPantalla);
        }

      /* Según el Servidor de Impresión elegido para la búsqueda */

      if (strcmpi(DirServImp,DirServ0Imp0)==0)
        {
          SignalLocalSemaphore(SemEsc0);
          strcpy(Eleccion0,Eleccion);
          strcpy(NombreFicheroBusca0,NombreFicheroBusca);
        }
      else
        if (strcmpi(DirServImp,DirServ0Imp1) == 0)
          {
            SignalLocalSemaphore(SemEsc0);
            strcpy(Eleccion0,Eleccion);
            strcpy(NombreFicheroBusca0,NombreFicheroBusca);
          }
        else
          if (strcmpi(DirServImp,DirServ1Imp0) == 0)
            {
                SignalLocalSemaphore(SemEsc1);
                strcpy(Eleccion1,Eleccion);
                strcpy(NombreFicheroBusca1,NombreFicheroBusca);
            }
```

```
            else
              if (strcmpi(DirServImp,DirServ1Imp1) == 0)
                  {
                    SignalLocalSemaphore(SemEsc1);
                    strcpy(Eleccion1,Eleccion);
                    strcpy(NombreFicheroBusca1,NombreFicheroBusca);
                  }
              else
                  if (strcmpi(DirServImp,DirServ2Imp0) == 0)
                    {
                      SignalLocalSemaphore(SemEsc2);
                      strcpy(Eleccion2,Eleccion);
                      strcpy(NombreFicheroBusca2,NombreFicheroBusca);
                    }
                  else
                    if (strcmpi(DirServImp,DirServ2Imp1) == 0)
                      {
                        SignalLocalSemaphore(SemEsc2);
                        strcpy(Eleccion2,Eleccion);
                        strcpy(NombreFicheroBusca2,NombreFicheroBusca);
                      }
                    else
                      {
                        ptr = "Se ha elegido un Servidor para búsqueda equivocada";
                        SignalLocalSemaphore(SemPantalla);
                      }

        /* Mandamos el mensaje sobre la búsqueda del fichero cuando ésta acabe */

        WaitOnLocalSemaphore(AcaboLaBusqueda);

        if (AbortarServImp == 1)
          {
            if (NumeroServImp == 3)   /* Servidor de Impresión 3 desconectado */
              {
                  strcpy(LugarEncuentroFichero,"Esta impresora y su Servidor  han  sido  desconectados. Inténtelo con otra
disponible.");
                  NumeroServImp--;
                  AbortarServImp = 0;
              }
            else
              {
                  if (NumeroServImp == 2)  /* Servidor de Impresión 2 desconectado */
                    {
                      strcpy(LugarEncuentroFichero,"Esta impresora y su Servidor  han  sido  desconectados. Inténtelo con
otra disponible.");
                      NumeroServImp--;
                      AbortarServImp = 0;
                    }
                  else
                    {
                      if (NumeroServImp == 1)  /* Servidor de Impresión 1 desconectado */
                        {
                            strcpy(LugarEncuentroFichero,"Todos los Servidores de Impresión han sido desconectados.");
                            Abortar = 1;
                        }
```

```
                }
            }
        }

    if (t_snd(fh2,LugarEncuentroFichero,sizeof(LugarEncuentroFichero),NULL)==-1)
      {
        ptr = "Error mandando mensaje sobre la búsqueda del fichero";
        SignalLocalSemaphore(SemPantalla);
      }

  }

if (strcmpi(Eleccion,"IMPRESION") == 0)
  {
    ptr = "Se ha elegido la opción de IMPRESION REMOTA";
    ThreadSwitch();
    SignalLocalSemaphore(SemPantalla);

    /* Recibiendo el nombre de la unidad */

    if (t_rcv(fh2,unidad,sizeof(unidad),NULL) == -1 )
      {
        ptr = "Error recibiendo el nombre de la unidad";
        SignalLocalSemaphore(SemPantalla);
      }

    /* Recibiendo el nombre del fichero a imprimir */

    if (t_rcv(fh2,FICH,sizeof(FICH),NULL) != -1 )
      {
        trabajo++;
        strcpy(Pantalla,"Trabajo de impresión número ");
        sprintf(Pantalla1,"%d",trabajo);
        strcat(Pantalla,Pantalla1);
        strcpy(ptr,Pantalla);
        SignalLocalSemaphore(SemPantalla);
        ThreadSwitch();
        strcpy(Pantalla,"Gestionando la impresión de ");
        strcat(Pantalla,FICH);
        strcpy(ptr,Pantalla);
        SignalLocalSemaphore(SemPantalla);
        ThreadSwitch();
      }
    else
      {
        ptr = "Error recibiendo el nombre del fichero";
        SignalLocalSemaphore(SemPantalla);
      }

    /* Si el fichero es local, abrimos un archivo temporal en F.    */
    /* Este archivo se irá llenando con paquetes de 132 caracteres */
    /* que irán llegando del cliente. Cuando el fichero  haya sido */
    /* traspasado completamente lo mandamos a imprimir.            */

    strcpy(iobuf,"            ");  /* Limpieza de buffer */
    Local=0;
```

```
if (strcmpi(unidad,"C:") == 0 )
  Local=1;
if (strcmpi(unidad,"D:") == 0 )
  Local=1;
if (strcmpi(unidad,"E:") == 0 )
  Local=1;
if (strcmpi(unidad,"A:") == 0 )
  Local=1;
if (strcmpi(unidad,"B:") == 0 )
  Local=1;

if (Local = 1)
  {

    fp1=fopen(FICH,"w"); /* Crea archivo de texto para escritura */

    if (fp1==NULL)
      {
          ptr = "Imposible crear archivo temporal";
          SignalLocalSemaphore(SemPantalla);
      }

    while ( strcmpi(iobuf,"FIN") != 0 )
      {
          if (t_rcv(fh2,iobuf,sizeof(iobuf),NULL) != -1)
            {
              if (strcmpi(iobuf,"FIN") != 0 )
                fprintf(fp1,"%s",iobuf);
            }
      }

    fclose(fp1);
    Borrar=1;  /* Para borrar el archivo temporal */

  } /* Final del if (strcmpi(unidad,"C") = 0 ) */

strcpy(iobuf,"            ");  /* Limpieza de buffer */

/* Si el fichero está en F sólo se manda el nombre del fichero */
/* con el path.                                                */

strcpy(FICHERO,FICH);

/* Despertamos el proceso de impresión dormido en el semáforo */
/* Según la impresora elegida despertamos uno u otro.         */

if (strcmpi(DirServImp,DirServ0Imp0)==0)
  {
    SignalLocalSemaphore(SemEsc0);
    strcpy(Eleccion0,Eleccion);
  }
else
  if (strcmpi(DirServImp,DirServ0Imp1)==0)
    {
      SignalLocalSemaphore(SemEsc0);
      strcpy(Eleccion0,Eleccion);
```

```
            }
          else
            if (strcmpi(DirServImp,DirServ1Imp0)==0)
              {
                  SignalLocalSemaphore(SemEsc1);
                  strcpy(Eleccion1,Eleccion);
              }
            else
              if (strcmpi(DirServImp,DirServ1Imp1)==0)
                  {
                    SignalLocalSemaphore(SemEsc1);
                    strcpy(Eleccion1,Eleccion);
                  }
              else
                  if (strcmpi(DirServImp,DirServ2Imp0)==0)
                    {
                      SignalLocalSemaphore(SemEsc2);
                      strcpy(Eleccion2,Eleccion);
                    }
                  else
                    if (strcmpi(DirServImp,DirServ2Imp1)==0)
                      {
                        SignalLocalSemaphore(SemEsc2);
                        strcpy(Eleccion2,Eleccion);
                      }
                    else
                      {
                        ptr = "Se ha elegido una opción de impresora equivocada";
                        SignalLocalSemaphore(SemPantalla);
                        strcpy(iobufForCli,"Se ha elegido una opción de impresora equivocada");
                        exito=0;
                        SignalLocalSemaphore(SemMandarExito);
                      }

        WaitOnLocalSemaphore(SemMandarExito);

        if (AbortarServImp == 1)
          {
            if (NumeroServImp == 3)   /* Servidor de Impresión 3 desconectado */
              {
                  exito = 0;
                  strcpy(iobufForCli,"Esta impresora y su Servidor  han  sido  desconectados. Inténtelo con otra
disponible.");
                  NumeroServImp--;
                  AbortarServImp = 0;
              }
            else
              {
                  if (NumeroServImp == 2)  /* Servidor de Impresión 2 desconectado */
                    {
                      exito = 0;
                      strcpy(iobufForCli,"Esta impresora y su Servidor  han  sido  desconectados. Inténtelo con otra
disponible.");
                      NumeroServImp--;
                      AbortarServImp = 0;
                    }
```

```
            else
              {
               if (NumeroServImp == 1)  /* Servidor de Impresión 1 desconectado */
                 {
                    exito = 0;
                    strcpy(iobufForCli,"Todos los Servidores de Impresión han sido desconectados.");
                    Abortar = 1;
                 }
              }
          }
      }

    /* Se envía mensaje al cliente sobre el resultado de su petición */

    if (exito==1)
      {
       strcpy(iobufForCli,"El fichero ha sido mandado a la impresora");
       ptr = "El fichero ha sido mandado a la impresora";
       SignalLocalSemaphore(SemPantalla);

       if( t_snd( fh2, iobufForCli , sizeof(iobufForCli), NULL) == -1 )
         {
            ptr = "No se ha podido mandar comunicado al cliente";
            SignalLocalSemaphore(SemPantalla);
            t_close(fh2);
            return;
         }
      }

    else
      {
       if( t_snd( fh2, iobufForCli , sizeof(iobufForCli), NULL) == -1 )
         {
            ptr = "Ha fallado la comunicación con el cliente";
            SignalLocalSemaphore(SemPantalla);
            t_close(fh2);
            return;
         }
      }

  }  /* Final de if (strcmpi(Eleccion,"IMPRESORA") == 0) */

if( t_look(fh2) == T_DISCONNECT )
  {
    ptr = "Recibida petición de desconexión 41, EXIT.";
    SignalLocalSemaphore(SemPantalla);
    if( t_rcvdis(fh2, NULL) < 0 )
      {
       ptr = "t_rcvdis 41";
       SignalLocalSemaphore(SemPantalla);
      }
    fclose(fp1);
    t_close(fh2);
    /*return;*/
  }
```

```
    /* Enviando desconexión al cliente */

    if( t_snddis(fh2, NULL) == -1 )
      {
         t_error("t_snddis 41");
         t_close(fh2);
         return;
      }

    if( t_unbind(fh2) == -1 )
      {
         t_error("t_unbind 41");
         t_close(fh2);
         return;
      }

    t_close(fh2);

    if (Abortar == 1)
      exit(2);

  }

/*************************************************************************/
/* Las funciones DespertarProcesos y LiberarRecursos son vitales    */
/* para una terminación ordenada de los procesos que forman el NLM. */
/* Una mala gestión de los semáforos ocasiona con facilidad bloqueo */
/* del servidor.                                                     */
/*************************************************************************/

void DespertarProcesos(void)
  {

    SignalLocalSemaphore(SemEsc0);
    SignalLocalSemaphore(SemEsc1);
    SignalLocalSemaphore(SemEsc2);
    SignalLocalSemaphore(SemMandarExito);
    SignalLocalSemaphore(AcaboLaBusqueda);
    SignalLocalSemaphore(SemPantalla);
    SignalLocalSemaphore(EntradaTeclado);
    myunload();   /* Para soltar los recursos utilizados de pantalla */
    LiberarRecursos();
    return;

  }

void LiberarRecursos(void)
  {

    CloseLocalSemaphore(SemEsc0);
    CloseLocalSemaphore(SemEsc1);
    CloseLocalSemaphore(SemEsc2);
    CloseLocalSemaphore(SemMandarExito);
    CloseLocalSemaphore(AcaboLaBusqueda);
    CloseLocalSemaphore(SemPantalla);
    CloseLocalSemaphore(EntradaTeclado);
```

```
    t_unbind(fh);
    t_close(fh);
    t_unbind(fh2);
    t_close(fh2);
    return;

  }

/*********************************************************************/
/* Esta función se encarga de guardar información diaria acerca de */
/* los sucesos acaecidos en el Servidor.                            */
/*********************************************************************/

void EntradaDatosDiarios(void)
  {

    FILE   *info;

    /* Abre el fichero si ya existe */

    if  ((info=fopen("DATO_SER.TXT","a")) != NULL)
      {
         /* Si existe añadimos datos al final */

         fprintf(info,"%s",datos);
      }
    else
      {
         /* Si no existe lo creamos */

         if ((info=fopen("DATO_SER.TXT","w")) == NULL)
           {
             ptr = "No se ha podido crear el fichero de datos de eventos";
             SignalLocalSemaphore(SemPantalla);
           }
         else
           fprintf(info,"%s",datos);

      }
    fclose(info);

  }

/* Se encarga de dibujar las ventanas de la pantalla de presentación */

void PantallaMenu()
  {
    int        a;
    LONG       ccode, type, portalNumber, infoPortalNumber;
    BYTE       value;
    LONG       portalTop = 0,            /* fila */
                   portalLeft = 0,           /* columna */
                   portalFrameHeight = 24,   /* alto */
                   portalFrameWidth = 80,    /* ancho */
                   portalVirtualHeight = 20, /* tamaño portal virtual 6 X 38 */
                   portalVirtualWidth = 78;
```

```
LONG        col,fil,alto,ancho,lineas;
WORD        screenHeight, screenWidth;

NLMHandle = GetNLMHandle();

/* Crea una pantalla para mostrar nuestra información */

CLIBScreenID=CreateScreen("Gestor de Impresión/Busqueda Remota",AUTO_DESTROY_SCREEN);
if (!CLIBScreenID)
  return;

allocTag=AllocateResourceTag(NLMHandle,"Gestor de Impresión/Busqueda Remota",
                                    AllocSignature);
if (!allocTag)
  {
      DestroyScreen(CLIBScreenID);
      return;
  }

/* Inicializa NWSNUT */

ccode=NWSInitializeNut(0,0,NO_HEADER,0,0,0,CLIBScreenID,allocTag,&handle);
if (ccode)
  {
      DestroyScreen(CLIBScreenID);
      return;
  }

/* Activa la pantalla creada */

DisplayScreen(CLIBScreenID);
GetSizeOfScreen (&screenHeight, &screenWidth);
infoPortalNumber = NWSCreatePortal((LONG) (screenHeight - 1), 0L,
                1L, (LONG)screenWidth, 1L, (LONG) screenWidth, SAVE,
                (BYTE *) 0, 0L, NOBORDER, 0, CURSOR_OFF, VIRTUAL, handle);

if (infoPortalNumber > MAXPORTALS)
  {
      printf ("No se puede crear información del portal\n");
      DestroyScreen(CLIBScreenID);
      return;
  }

/* Convierte el número de portal a puntero PCB */

 NWSGetPCB (&ipPtr, infoPortalNumber, handle);
 SetPositionOfInputCursor(3,22);
 printf("╔═══════════════════════════╗\n");
 SetPositionOfInputCursor(4,22);
 printf("║  PROGRAMA DE IMPRESION REMOTA  ║\n");
 SetPositionOfInputCursor(5,22);
 printf("║             Y             ║\n");
 SetPositionOfInputCursor(6,22);
 printf("║       BUSQUEDA  REMOTA        ║\n");
 SetPositionOfInputCursor(7,22);
 printf("╚═══════════════════════════╝\n");
```

```
WriteInstructions ("<Presione cualquier tecla para comenzar>", ipPtr);
NWSGetKey (&type, &value, handle);

SetPositionOfInputCursor(3,15);
printf("╔═══════════════════════════════════════╗\n");
SetPositionOfInputCursor(4,15);
printf("╟ ¿Cuantos servidores de impresión desea instalar? ╢\n");
SetPositionOfInputCursor(5,15);
printf("╟ 1, 2 o 3                             ╢\n");
SetPositionOfInputCursor(6,15);
printf("╟ Opción:                              ╢\n");
SetPositionOfInputCursor(7,15);
printf("╟                                      ╢\n");
SetPositionOfInputCursor(8,15);
printf("╚═══════════════════════════════════════╝\n");

WriteInstructions ("<Elija una opción>", ipPtr);

a=getch();
if (a=='1')
  NumServImp=1;
if (a=='2')
  NumServImp=2;
if (a=='3')
  NumServImp=3;

SignalLocalSemaphore(EntradaTeclado);

/* Crea un portal virtual */

portalNumber = NWSCreatePortal(portalTop, portalLeft,
              portalFrameHeight, portalFrameWidth, portalVirtualHeight,
              portalVirtualWidth, SAVE, "PROGRAMA DE IMPRESION REMOTA / BUSQUEDA REMOTA", 0,
              DOUBLE, 0, CURSOR_OFF, VIRTUAL, handle);

if (portalNumber > MAXPORTALS)
  {
     NWSDestroyPortal (infoPortalNumber, handle);
     printf ("No se puede crear portal virtual\n");
     DestroyScreen(CLIBScreenID);
     NWSRestoreNut(handle);
     return;
  }

NWSGetPCB (&pPtr, portalNumber, handle);
NWSClearPortal (pPtr);
NWSSelectPortal (portalNumber, handle);

WriteInstructions ("  <Información sobre Servidores Remotos disponibles y conexiones con clientes>", ipPtr);


NWSDisplayTextInPortal(0,1,"Esperando por la entrada de Servidores Remotos...",VBLINK,pPtr);
NWSUpdatePortal (pPtr);
NWSShowPortalLine (0, 1, ptr, strlen (ptr), pPtr);
NWSUpdatePortal (pPtr);*/
NWSShowPortalLine (6, 0, ptr, strlen (ptr), pPtr);
```

```
    NWSUpdatePortal (pPtr);*/

    /* Mostrar en pantalla las impresoras disponibles */

    fila = 1;
    while (cliente != 1)
      {
         NWSDisplayTextInPortal(0,0,"========== IMPRESORAS DISPONIBLES
==========",VINTENSE,pPtr);
         WaitOnLocalSemaphore(SemPantalla);
         NWSShowPortalLine (fila++, 1, ptr, strlen (ptr), pPtr);
         NWSUpdatePortal (pPtr);
      }
    NWSDisplayTextInPortal(fila++,0,"========== INFORMACION SOBRE CLIENTES
==========",VINTENSE,pPtr);
    NWSUpdatePortal (pPtr);

    NWSShowPortalLine (fila++, 0, ptr, strlen (ptr), pPtr);
    NWSUpdatePortal (pPtr);*/

    a = fila; /* Guardamos la fila donde empiezan los clientes */
    FinalSalidaPantalla=0;
    while (FinalSalidaPantalla == 0)
      {
              lineas = 1;
              NWSScrollPortalZone(fil,col,alto,ancho,VIBLINK,lineas,V_UP,pPtr);
              NWSUpdatePortal (pPtr);
              fila = 17;
           }
         else
           {
             WaitOnLocalSemaphore(SemPantalla);
             NWSShowPortalLine (fila++, 1, ptr, strlen (ptr), pPtr);
             NWSUpdatePortal (pPtr);

           }

         WaitOnLocalSemaphore(SemPantalla);
         NWSShowPortalLine (fila++, 1, ptr, strlen (ptr), pPtr);
         NWSDrawPortalBorder(pPtr);
         NWSUpdatePortal (pPtr);
         if (fila >= 19)
           {
             col = 1;fil = a;alto = 20 - fil;ancho = 78;
             lineas = 1;
             NWSScrollPortalZone(fil,col,alto,ancho,VNORMAL,lineas,V_UP,pPtr);
             NWSUpdatePortal (pPtr);
             fila = 18;
           }

      } /* Final del  while (FinalSalidaPantalla==0) */

    WriteInstructions ("<Presione ESC para cerrar esta ventana>", ipPtr);
    NWSWaitForEscape(handle);

   /* Limpia y descarta este portal */
```

```
    NWSDestroyPortal (portalNumber, handle);

    /* Destruye la pantalla creada */

    DestroyScreen(CLIBScreenID);
  }

void myunload(void)
  {
    NWSRestoreNut(handle);
    DestroyScreen(CLIBScreenID);
  }

void WriteInstructions (BYTE *p, PCB *pcb)
  {
    NWSClearPortal (pcb);
    NWSShowPortalLine (0, 0, p, strlen (p), pcb);
    NWSUpdatePortal (pcb);
  }
```

LISTADO PROGRAMA CLIENTE

```
#include <graphics.h>
#include <time.h>
#include <nwalias.h>
#include <nwcaldef.h>
#include <nwconnec.h>
#include <nwmisc.h>
#include <nwcalls.h>
#include <string.h>
#include <stdio.h>
#include <ctype.h>
#include <process.h>
#include <fcntl.h>
#include <tispxipx.h>
#include <tiuser.h>
#include <conio.h>
#include <bios.h>
#include <stdlib.h>
#include <dir.h>
#include <dos.h>                /* delay....*/

#define   SPX_SOCKET      41                        /* Socket estático */
#define   fa2             748             /* Tonos para señal de alarma */
#define   do1             396
#define   VISIBLE         0       /* Para que el cursor aparezca visible */
#define   INVISIBLE       1       /* o invisible.                        */

#define   NORMAL          0       /* Modo video */
#define   INVERSO         1
#define   TECLAESPECIAL   0       /* Movimiento */
#define   TECLADERECHA    77
#define   TECLAIZQUIERDA  75

#define   TECLAARRIBA     72
#define   TECLAABAJO      80
#define   TECLAINTRO      13      /* Elección */

/************************/
/** Variables Globales **/
/************************/

char      NomFichIn[50],NomFichOut[50];       /* Nombre del finchero */
char      NumeroImpresoras[70];   /* Número de impresoras disponbles */
char      Fin[132];                          /* Para final de fichero */
char      Letra[3];         /* Entrada de la elección de la impresora */

/* Nombres de impresoras y direcciones */

char      Direccion0[132],Direccion01[132], /* Dos impresoras por Servidor */
          Direccion1[132],Direccion11[132],
          Direccion2[132],Direccion21[132],
          Direccion[132];

char      DatosConeccion[132],    /* Información sobre la conexión */
          DatosPersonales[132];   /* Datos personales de usuario    */

int       continua;
```

```
int       NumeroVueltas;           /* Número de vueltas para ruleo      */
int       NumImp0,NumImp1,NumImp2; /* Número de impresoras en servicio  */
                                    /* por Servidor                      */
int       num;                     /* Número de Servidores de Impresión */

struct cargamenu {  unsigned x;
                    unsigned y;
                    char titulo[80];
                    char letra;       };

struct cargamenu menuprincipal[]={ 5,2," A. IMPRESION REMOTA ",'a',
                                   5,4," B. SALIR           ",'b',
                                   5,6," C. BUSQUEDA REMOTA  ",'c'   };


unsigned menu(struct cargamenu *menuprincipal,int nopciones);
void VentanaParaElegirOpciones(void);
void modovideo(int modo);
void LimpiarPantalla(void);
void marc(void);

int  PasaDireccion( char *direccion, IPX_ADDR *destino );
int  PonDireccion( char *cadena, char *buf, int hexBytes );
void RazonDesconeccionSPX( int fd );
int  MandoFicheroServidor(int *mfs);
void VentanasPantalla(void);
void FondoPantalla(void);
void CabeceraPantalla(void);
void Nota(unsigned int frec);
void Inipit(unsigned int frec);
void AbortarConexion(void);
void RestablecerPantalla(void);
void ObtencionDatosConeccion(void);
void ValidarEleccionImp();
void MandarEleccionImp(int *fd1);
void RecivirImpresorasDisponibles(int *fd2);
void ruleo(int posx,int posy,int time);
void songana();
void formacursor(int forma);
void sonpierde();
void DesapareceImpresora();

/**********************/
/** Comienzo de main **/
/**********************/

void main ( int argc, char *argv[] )

  {

    IPX_ADDR          spx_addr;
    SPX_OPTS          spx_options;
    char              buf[ 100 ], buf2[ 100 ],iobuf[132];
    int               fd, flags, i1,EI,aviso,sigue,i,Local,error,cont;
    struct t_call     tcall;
    char              unidad[MAXDRIVE],directorio[MAXDIR],
```

```
                archivo[MAXFILE],extension[MAXEXT],
                NomFichOutRed[50],NombreServidorFicheros[70],
                NombreFicheroBusca[13],LugarEncuentroFichero[300],
                Eleccion[132],EleccImp[3];


if ( argc != 2 || !PasaDireccion( argv[ 1 ], & spx_addr ) )
  {
    printf("El usuario ha de introducir en notación hexadecimal:\n" );
    printf("  -Nodo de red\n" );
    printf("  -Dirección del servidor\n" );
    printf("El formato de la orden será el siguiente:\n");
    printf("   cliente 00001234/00001b02362e\n");
    printf("Si estos datos son desconocidos para usted ejecute SYSCON\n");
    printf("y entre en 'File Server Information' \n");
    exit( 1 );
  }

ObtencionDatosConeccion();

/* Para el menú de pantalla */

continua=1;   /* Mientras continúa == 1 se repetirá este bucle */

do
  {
    window(1,1,80,25);
    clrscr();
    textbackground(1);
    FondoPantalla();
    VentanasPantalla();
    textbackground(3);
    CabeceraPantalla();
    textbackground(1);
    textcolor(14);

    /* Se abre un extremo de conexión */

    if ((fd = t_open( "/dev/nspx", O_RDWR, NULL) ) == -1 )
      {
        AbortarConexion();
        gotoxy(22,7);
        cprintf("Error abriendo fichero /dev/nspx\n" );
        RestablecerPantalla();
        exit( 2 );
      }

    /* Se enlaza el extremo de conexión una vez abierto con una dirección */

    if (t_bind(fd,NULL,NULL)== -1)
      {
        AbortarConexion();
        gotoxy(22,7);
        cprintf("El enlace ha fallado\n" );
        RestablecerPantalla();
        exit( 2 );
```

```
  }

/* Se inicializan las diferentes estructuras */

spx_addr.ipxa_socket[ 0 ] = 0;
spx_addr.ipxa_socket[ 1 ] = SPX_SOCKET;
tcall.addr.buf = ( char * )&spx_addr;
tcall.addr.len = sizeof( spx_addr );
tcall.addr.maxlen = sizeof( spx_addr );
spx_options.spx_connectionID[ 0 ] = 0;
spx_options.spx_connectionID[ 1 ] = 0;
spx_options.spx_allocationNumber[ 0 ] = 0;
spx_options.spx_allocationNumber[ 1 ] = 0;
tcall.opt.buf = ( char * )&spx_options;
tcall.opt.len = sizeof( spx_options );
tcall.opt.maxlen = sizeof( spx_options );
tcall.udata.buf = ( char * )0;
tcall.udata.len = 0;
tcall.udata.maxlen = 0;
flags=0;

if ( t_connect( fd, &tcall, &tcall ) == -1 )
  {
    AbortarConexion();
    gotoxy(22,7);
    cprintf("La función t_connect ha fallado");
    RestablecerPantalla();
    exit( 2 );
  }

textbackground(1);
textcolor(15);

/* Enviando el paquete de datos de la conexión con el Servidor */

if (t_snd(fd,DatosConeccion,sizeof(DatosConeccion),NULL)==-1)
  cprintf("Se ha producido un error enviando los datos de la conexión\r\n");

/* Número de impresoras disponibles */

if (t_rcv(fd,NumeroImpresoras,sizeof(NumeroImpresoras),&flags)==-1)
  cprintf("Se ha producido un error recibiendo el número de impresoras disponibles\r\n");

/* La función t_rcv se encarga de recibir información sobre las */
/* impresoras disponibles */

if (strcmp(NumeroImpresoras,"1") == 0)
  num=1;
if (strcmp(NumeroImpresoras,"2") == 0)
  num=2;
if (strcmp(NumeroImpresoras,"3") == 0)
  num=3;

/*continua=1;*/

/* De un Servidor de impresión se pueden recibir hasta dos impresoras */
```

```
/* Se reciben la impresoras disponibles en la red */

RecivirImpresorasDisponibles(&fd);

/* Se crea una ventana de diálogo con el usuario */

window(3,14,77,22);

/* Se recibe el nombre del Servidor para formar el camino */
/* completo de los ficheros de red.                       */

if (t_rcv(fd,NombreServidorFicheros,sizeof(NombreServidorFicheros),&flags)!=-1)
  cprintf("Esta usted conectado con el Servidor: %s\r\n",strupr(NombreServidorFicheros));

cprintf("Introduzca el número que corresponda a su elección de impresora.\r\n");
cprintf("Número:  ");
gets(Letra);
strupr(Letra);

/* Validación de la elección de la impresora */

ValidarEleccionImp();
DesapareceImpresora();
strcpy(EleccImp,Letra);

/* Mandamos la impresora elegida */

MandarEleccionImp(&fd);
clrscr();

/* Elegir entre la posibilidad de búsqueda o impresión remota */

window(1,1,80,24);

VentanaParaElegirOpciones();
_setcursortype(_NORMALCURSOR);
window(3,14,77,23);
clrscr();
textcolor(15);
textbackground(1);
gotoxy(1,1);
cprintf("\r");
cprintf("Está usted conectado con el Servidor: %s\r\n",strupr(NombreServidorFicheros));

if (strcmp(Letra,"C") == 0)
  {
    strcpy(Eleccion,"VOLVER");
  }

if (strcmp(Letra,"A") == 0)
  {
    strcpy(Eleccion,"BUSQUEDA");
    window(1,1,80,25);
    gotoxy(78,5);
    textcolor(14);
    cprintf("X");
```

```
            gotoxy(23,5);
            cprintf("%s",EleccImp);
            window(3,14,77,22);
            textcolor(15);
          }

        if (strcmp(Letra,"B") == 0)
          {
            strcpy(Eleccion,"IMPRESION");
            window(1,1,80,25);
            gotoxy(23,5);
            textcolor(14);
            cprintf("%s",EleccImp);
            window(3,14,77,22);
            textcolor(15);
          }

        if (t_snd(fd,Eleccion,sizeof(Eleccion),NULL) == -1)
          cprintf("\rError mandando la elección de trabajo\r\n");

        if (strcmp(Letra,"A") == 0)
          {
            cprintf("\r\nIntroduzca el nombre del fichero a buscar: ");
            gets(NombreFicheroBusca);

            if (t_snd(fd,NombreFicheroBusca,sizeof(NombreFicheroBusca),NULL) == -1)
              cprintf("\rError mandando el nombre de fichero a buscar\r\n");

            if (t_rcv(fd,LugarEncuentroFichero,sizeof(LugarEncuentroFichero),&flags)==-1)
              cprintf("\rError reciviendo mensaje sobre busqueda de fichero\r\n,");

            cprintf("\r%s\n\r",LugarEncuentroFichero);

            if (strcmpi(LugarEncuentroFichero,"Todos los Servidores de Impresión han sido
desconectados.") == 0)
              {
                sonpierde();
                delay(4000);
                AbortarConexion();
                gotoxy(20,7);
                cprintf("No hay Servidores Remotos disponibles\n" );
                RestablecerPantalla();
                exit( 2 );
              }

            if(strcmpi(LugarEncuentroFichero,"->") == 0)
              {
                cprintf("EL fichero no ha sido encontrado\r\n");
                sonpierde();
              }
            else
              {
                if (strcmpi(LugarEncuentroFichero,"Esta impresora y su Servidor han sido desconectados.
  Intentelo con otra disponible.") == 0)
                  sonpierde();
                else
```

```
            songana();
        }
    }

if (strcmp(Letra,"B") == 0)
  {

    cprintf("\r\n¿ Desea que la 1ª hoja contenga datos de usuario, S/N ?  ");
    gets(Letra);
    strupr(Letra);

    if (strcmp(Letra,"S") == 0)
      continua=1;
    else
      continua=0;

    if (continua == 1)
      {
        cprintf("\rIntroduzca sus datos personales: \r\n");
        gets(DatosPersonales);
      }

    cprintf("\r");
    cprintf("Introduzca el nombre y camino completo del fichero que desea imprimir. \r\n");
    cprintf("Nombre:  ");

    do              /* Validando las entradas */
      {

        gets(NomFichIn);
        strupr(NomFichIn);
        i=fnsplit(NomFichIn,unidad,directorio,archivo,extension);
        if (i & DRIVE)
          if (i & DIRECTORY)
            if (i & FILENAME)
              cont=1;
            else
              {
                cont=0;
                gotoxy(1,wherey()-1);
                cprintf("Nombre:  ");
                sonpierde();
              }
          else
            {
              cont=0;
              gotoxy(1,wherey()-1);
              cprintf("Nombre:  ");
              sonpierde();
            }
        else
          {
            cont=0;
            gotoxy(1,wherey()-1);
            cprintf("Nombre:  ");
            sonpierde();
```

```
        }

    } while (cont == 0);

/* Mandamos la unidad */

strcpy(NomFichOut,unidad);

if (t_snd(fd,NomFichOut,sizeof(NomFichOut),NULL) == -1)
  {
    cprintf("Se ha producido un error mandando la unidad\r\n");
    exit(2);
  }

/* Si el fichero es local mandamos el fichero en paquetes de 132 caracteres */

strcpy(NomFichOut,archivo);
strcat(NomFichOut,extension);

if (strcmp(unidad,"F:") == 0 )
  {

    /* Si el fichero está en F mandamos sólo el nombre del fichero */
    /* después de preparar la estructura para que sea reconocida   */
    /* por el software de red.                                     */

    strcat(NombreServidorFicheros,"/SYS:");
    strcpy(NomFichOutRed,NombreServidorFicheros);
    strcat(NomFichOutRed,directorio);
    strcat(NomFichOutRed,archivo);
    strcat(NomFichOutRed,extension);

    if (t_snd(fd,NomFichOutRed,sizeof(NomFichOutRed),NULL) == -1)
      {
        cprintf("Se ha producido un error mandando el nombre del fichero\r\n");
        exit(2);
      }
  }

strcpy(Fin,"FIN");
Local=0;  /* Si el fichero es local entonces Local=1 */

if (strcmp(unidad,"C:") == 0 )
  Local=1;
if (strcmp(unidad,"D:") == 0 )
  Local=1;
if (strcmp(unidad,"E:") == 0 )
  Local=1;
if (strcmp(unidad,"A:") == 0 )
  Local=1;
if (strcmp(unidad,"B:") == 0 )
  Local=1;

if (Local == 1)
  {
    /* Mandamos el nombre del fichero */
```

```
    if (t_snd(fd,NomFichOut,sizeof(NomFichOut),NULL) == -1)
      {
        cprintf("\r");
        cprintf("Se ha producido un error mandando el nombre del fichero\r\n");
        exit(2);
      }

    /* error = -1 entonces se ha producido un error */

    error=MandoFicheroServidor(&fd);

    /* Mandamos la señal de Fin de fichero */

    if ( t_snd( fd, Fin, sizeof( Fin ), NULL ) == -1 )
      {
        delay(2000);
        AbortarConexion();
        gotoxy(22,7);
        cprintf( "t_snd mandando final fichero falló\r\n" );
        RestablecerPantalla();
        exit( 2 );
      }

    if (error == 0)
      cprintf("\rFichero transferido al Servidor                          \r\n");

  }   /* Final de if (Local == 1) */

/* Esperando mensaje sobre la impresión del fichero */

if (error == 0)
  cprintf("\rEl fichero está siendo enviado a la impresora, espere...");

flags=0;

_setcursortype(_NOCURSOR);

do
  {
    if (error == 0)   /* Si el fichero existe */
      if (NumeroVueltas != 0)   /* Si número de líneas > 45 */
        {
          textcolor(14);
          ruleo(wherex()+1,wherey(),NumeroVueltas);
        }
    textcolor(15);

  } while (t_rcv(fd,iobuf,sizeof(iobuf),&flags)==-1);

_setcursortype(_NORMALCURSOR);

cprintf("\r\n%s\r\n",iobuf);

if (strcmpi(iobuf,"Todos los Servidores de Impresión han sido desconectados.") == 0)
  {
    sonpierde();
```

```
                delay(4000);
                AbortarConexion();
                gotoxy(20,7);
                cprintf("No hay Servidores Remotos disponibles\n" );
                RestablecerPantalla();
                exit( 2 );
              }

            if (strcmpi(iobuf,"No se puede abrir el fichero o no existe.") == 0)
              sonpierde();
            else
              {
                if (strcmpi(iobuf,"Esta impresora y su Servidor  han  sido  desconectados. Inténtelo
con otra disponible.") == 0)
                  sonpierde();
                else
                  songana();
              }

          }  /* Final de if (strcmp(Letra,"B") == 0) */

        cprintf("\r¿ Quiere seguir trabajando, S/N ?   ");
        gets(Letra);
        strupr(Letra);

        if (strcmp(Letra,"S") == 0)
          continua=1;
        else
          continua=0;

        cprintf("\r\n");

      }  while (continua == 1);

    _setcursortype(_NOCURSOR);
    RestablecerPantalla();
    return;

  }

/*******************/
/** Final de main **/
/*******************/

int PasaDireccion( char *direc, IPX_ADDR *destino )
  {
    if ( strlen( direc ) == 21 && direc[ 8 ] == '/' )
    if ( PonDireccion( direc, destino->ipxa_net, 4 ) )
    if ( PonDireccion( &direc[ 9 ], destino->ipxa_node, 6 ) )
      return 1;
    return 0;
  }

int PonDireccion( char *cadena, char *buf, int hexBytes )
  {
    int   i, j, valor;
```

```
    char  c;

    for ( i = 0; i < hexBytes; i++ )
      {
        valor = 0;
        for ( j = 0; j < 2; j++ )
          {
            valor <<= 4;
            if ((c = toupper(*cadena)) >= '0' && c <= '9')
              valor += c - '0';
            else
              if (c >= 'A' && c <= 'F')
                valor += c - 'A' + 10;
              else return 0;
                cadena++;
          }
        *buf++ = valor;
      }

    return 1;
  }

int MandoFicheroServidor(int *mfs)
  {

    FILE  *fp;          /* Puntero a fichero */
    char  iobuf[132];
    int   i,col;

    if ((fp=fopen(NomFichIn,"rt")) == NULL)
      return -1;           /* No se puede abrir el fichero o no existe */

    if (continua == 1)
      {
        sprintf(iobuf,"DATOS DE USUARIO: \n");
        if (t_snd(*mfs,iobuf,sizeof(iobuf),NULL)==-1)
            {
              cprintf("\r");
              cprintf("Se ha producido un error enviando los datos de usuario\r\n");
              fclose(fp);
              t_close(*mfs);
              return -1;
            }

        sprintf(iobuf,"%s\n",DatosConeccion);
        if (t_snd(*mfs,iobuf,sizeof(iobuf),NULL)==-1)
            {
              cprintf("\r");
              cprintf("Se ha producido un error enviando los datos de usuario\r\n");
              fclose(fp);
              t_close(*mfs);
              return -1;
            }

        sprintf(iobuf,"%s\n\f",DatosPersonales);
        if (t_snd(*mfs,iobuf,sizeof(iobuf),NULL)==-1)
```

```
            {
              cprintf("\r");
              cprintf("Se ha producido un error enviando los datos de usuario\r\n");
              fclose(fp);
              t_close(*mfs);
              return -1;
            }

      }

    /* Lee líneas del fichero y se las envía al cliente */

    cprintf("\rTransfiriendo al Servidor\r");
    i=0;
    col=27;
    NumeroVueltas=0;

    while (fgets(iobuf,sizeof(iobuf),fp)!=NULL)
      {
        if (t_look(*mfs)==T_DISCONNECT)
          {
            cprintf("Recivida desconección, EXIT \r\n");
            if (t_rcvdis(*mfs,NULL)<0)
              cprintf("Se ha producido un error t_rcvdis\r\n");
            fclose(fp);
            t_close(*mfs);
            return -1;
          }
        if (t_snd(*mfs,iobuf,sizeof(iobuf),NULL)==-1)
          {
            cprintf("\r");
            cprintf("Se ha producido un error enviando el fichero\r\n");
            fclose(fp);
            t_close(*mfs);
            return -1;
          }
        i++;
        if (i == 45)
          {
            gotoxy(col++,wherey());
            cprintf(".");
            i=0;
            NumeroVueltas=2+NumeroVueltas;  /* Para tiempo de espera */
          }

      }

    fclose(fp);
    return 0;       /* Ha leido el fichero con exito */
  }

/*************************************************************************/
/*    Se encarga de dibujar las ventanas de la pantalla de presentación    */
/*************************************************************************/

void VentanasPantalla(void)
```

```
{
  int col,fil;

  textcolor(11);
  textbackground(1);
  gotoxy(1,4); cprintf("╒");

  for (col=2;col<=79;col++)
    {
      gotoxy(col,4);
      cprintf("=");
    }

  gotoxy(80,4); cprintf("╕");
  gotoxy(21,4); cprintf("╤");
  gotoxy(26,4); cprintf("╤");
  gotoxy(52,4); cprintf("╤");
  gotoxy(76,4); cprintf("╤");
  gotoxy(1,5);  cprintf("│");
  gotoxy(3,5);  cprintf("IMPRESION REMOTA  │    │");
  gotoxy(52,5); cprintf("│ DIRECTORIO REMOTO     │ ");

  for (col=2;col<=79;col++)
    {
      gotoxy(col,6);
      cprintf("=");
    }

  gotoxy(21,6); cprintf("╧");
  gotoxy(26,6); cprintf("╧");
  gotoxy(52,6); cprintf("╧");
  gotoxy(76,6); cprintf("╧");
  gotoxy(1,6);  cprintf("╘");
  gotoxy(80,6); cprintf("╛");
  gotoxy(80,5); cprintf("│");
  gotoxy(80,6); cprintf("╛");

  textcolor(14);
  textbackground(1);

  for (col=2;col<=79;col++)
    {
      gotoxy(col,7);
      cprintf("=");
    }

  gotoxy(27,7); cprintf(" IMPRESORAS  DISPONIBLES ");

  for (fil=8;fil<=23;fil++)
    {
      gotoxy(1,fil);
      cprintf("│");
    }

  gotoxy(1,7);  cprintf("╒");
  gotoxy(80,7); cprintf("╕");
```

```
    gotoxy(1,13); cprintf("╞");

    for (col=2;col<=79;col++)
      {
        gotoxy(col,13);
        cprintf("=");
      }

    gotoxy(29,13); cprintf(" VENTANA DE DIALOGO ");

    for (fil=8;fil<=23;fil++)
      {
        gotoxy(80,fil);
        cprintf("│");
      }

    gotoxy(80,13); cprintf("╡");
    gotoxy(1,24);  cprintf("└");

    for (col=2;col<=79;col++)
      {
        gotoxy(col,24);
        cprintf("=");
      }

    gotoxy(80,24); cprintf("┘");
    return;

  }

void FondoPantalla(void)
  {
    int fil,col;

    for (fil=1;fil<=24;fil++)
      for (col=1;col<=80;col++)
        cprintf(" ");
  }

void CabeceraPantalla(void)
  {
    int fil,col;

    gotoxy(1,1);
    for (fil=1;fil<=3;fil++)
      for (col=1;col<=80;col++)
        cprintf(" ");

    textcolor(14);
    for (col=1;col<=80;col++)
      {
        gotoxy(col,1);
        cprintf("-");
      }

    gotoxy(1,1);  cprintf("┌");
```

```
    gotoxy(80,1); cprintf("┐");
    gotoxy(1,2);  cprintf("│");
    gotoxy(80,2); cprintf("│");

    for (col=1;col<=80;col++)
      {
        gotoxy(col,3);
        cprintf("─");
      }

    gotoxy(1,3);  cprintf("└");
    gotoxy(80,3); cprintf("┘");
    textcolor(14);
    gotoxy(11,2); cprintf(".....PROGRAMA DE IMPRESION REMOTA / BUSQUEDA  REMOTA.....");
  }

void Nota(unsigned int frec)
  {
    int x,y,i;

    Inipit(frec);
    y=inportb(0x61);
    x=y | 0x03;
    outport(0x61,x);
    y=inport(0x61);
    delay(400L);
    x=y & 0xFC;
    outport(0x61,x);
  }

void Inipit(unsigned int frec)
  {
    unsigned num=1193180L / frec;

    outportb(0x43,0xB6);
    outportb(0x42,num & 0x00FF);
    outportb(0x42,num >> 8);
  }

void AbortarConexion(void)
  {
    int aviso;

    _setcursortype(_NOCURSOR);
    for (aviso=1;aviso<=5;aviso++)
         {
           window(3,14,77,22);
           textbackground(3);
           textcolor(14);
           gotoxy(20,5); cprintf("                                  ");
           delay(250L);
           gotoxy(16,3); cprintf("╔═══════════════════════════════════════╗");
           gotoxy(16,4); cprintf("║                                       ║");
           gotoxy(16,5); cprintf("║      .......CONEXION ABORTADA.......  ║");
           gotoxy(16,6); cprintf("║                                       ║");
           gotoxy(16,7); cprintf("╚═══════════════════════════════════════╝");
```

```
            gotoxy(16,8); cprintf("╚═══════════════════════════════════════╝");
            delay(500L);
            Nota(fa2);
            Nota(do1);
          }
  }

void RestablecerPantalla(void)
  {
    textbackground(1);
    textcolor(14);
    gotoxy(15,10);
    cprintf("\r\n\r\n             <Pulse cualquier tecla para cerrar esta pantalla>");
    getch();
    window(1,1,80,25);
    textbackground(0);
    textcolor(15);
    clrscr();
    _setcursortype(_NORMALCURSOR);
  }

/*****************************************************************/
/* Esta función se encarga de obtener los datos referentes a la */
/* conexión, tales como login, número de conexión...          */
/*****************************************************************/

void ObtencionDatosConeccion(void)
  {
    WORD      nwrc,connNumber,connHandle,objtype;
    DWORD     objID;
    BYTE      logintime[7];
    char      objname[48],Moment[20];

    /* Obtención de los datos de la conexión con el Servidor.  */
    /* Estos datos se empaquetan y se mandan al Servidor.      */
    /* El paquete contiene número de conexión y login (nombre, */
    /* hora y fecha).                                          */

    nwrc = NWCallsInit(NULL, NULL);
    if (nwrc)
      {
        printf("\nNWCallsInit: %04x",nwrc);
        exit(1);
      }

    nwrc = NWGetDefaultConnectionID(&connHandle);
    if (nwrc)
      {
        printf("\nNWGetDefaultConnectionID: %04x",nwrc);
        exit(1);
      }

    nwrc = NWGetConnectionNumber(connHandle, &connNumber);
    if (nwrc)
      {
        printf("\nNWGetConnectionNumber: %04x",nwrc);
```

```
        exit(1);
      }

    sprintf(Moment,"%d",connNumber);
    strcpy(DatosConeccion,"Conección: ");
    strcat(DatosConeccion,Moment);

    nwrc = NWGetConnectionInformation(connHandle,connNumber,
    (char far *)objname,(WORD far *)&objtype,(DWORD far *)&objID,
    (BYTE far *)logintime);
    if (nwrc)
      {
        printf("\nNWGetConnectionInformation: %04x",nwrc);
        exit(1);
      }

    strcat(DatosConeccion," ⇔ Login: ");
    strcat(DatosConeccion,objname);
    sprintf(Moment,"%02d/%02d/%02d",logintime[2],logintime[1],logintime[0]);
    strcat(DatosConeccion," ⇔ Fecha login: ");
    strcat(DatosConeccion,Moment);
    strcat(DatosConeccion," ⇔ Hora: ");
    sprintf(Moment,"%02d:%02d:%02d",logintime[3],logintime[4],logintime[5]);
    strcat(DatosConeccion,Moment);

  }

void ruleo(int posx,int posy,int time)
  {
    int i;

    for(i=0;i<time;i++)
      {
        gotoxy(posx,posy);cprintf("|");
        delay(200);
        gotoxy(posx,posy);cprintf("/");
        delay(200);
        gotoxy(posx,posy);cprintf("-");
        delay(200);
        gotoxy(posx,posy);cprintf("|");
        delay(200);
        gotoxy(posx,posy);cprintf("-");
        delay(200);
        gotoxy(posx,posy);cprintf("\\");
        delay(200);
      }
  }

/*************************************************/
/* Señal sonora para el éxito de las operaciones */
/*************************************************/

void songana()
  {
    sound(1000);delay(75);
    sound(1250);delay(75);
```

```
    sound(1500);delay(75);
    sound(1750);delay(75);
    sound(2000);delay(75);
    nosound();
  }

/*************************************************/
/* Señal sonora para el fracaso en las operaciones */
/*************************************************/

void sonpierde()
  {
    sound(30);delay(1000);
    sound(15);delay(500);
    nosound();
  }

void DesapareceImpresora()
  {
    if (strcmp(Letra,"1A") == 0)
      {
        if (strcmp(Direccion0,"ESTA IMPRESORA YA NO ESTA DISPONIBLE") == 0)
          {
            gotoxy(1,3);
            cprintf("Número:  ");
            gets(Letra);
            strupr(Letra);
            DesapareceImpresora();
          }
      }
    if (strcmp(Letra,"1B") == 0)
      {
        if (strcmp(Direccion01,"ESTA IMPRESORA YA NO ESTA DISPONIBLE") == 0)
          {
            gotoxy(1,3);
            cprintf("Número:  ");
            gets(Letra);
            strupr(Letra);
            DesapareceImpresora();
          }
      }
    if (strcmp(Letra,"2A") == 0)
      {
        if (strcmp(Direccion1,"ESTA IMPRESORA YA NO ESTA DISPONIBLE") == 0)
          {
            gotoxy(1,3);
            cprintf("Número:  ");
            gets(Letra);
            strupr(Letra);
            DesapareceImpresora();
          }
      }
    if (strcmp(Letra,"2B") == 0)
      {
        if (strcmp(Direccion11,"ESTA IMPRESORA YA NO ESTA DISPONIBLE") == 0)
          {
```

```
            gotoxy(1,3);
            cprintf("Número:  ");
            gets(Letra);
            strupr(Letra);
            DesapareceImpresora();
          }
      }
    if (strcmp(Letra,"3A") == 0)
      {
        if (strcmp(Direccion2,"ESTA IMPRESORA YA NO ESTA DISPONIBLE") == 0)
          {
            gotoxy(1,3);
            cprintf("Número:  ");
            gets(Letra);
            strupr(Letra);
            DesapareceImpresora();
          }
      }
    if (strcmp(Letra,"3B") == 0)
      {
        if (strcmp(Direccion21,"ESTA IMPRESORA YA NO ESTA DISPONIBLE") == 0)
          {
            gotoxy(1,3);
            cprintf("Número:  ");
            gets(Letra);
            strupr(Letra);
            DesapareceImpresora();
          }
      }

  }

void ValidarEleccionImp()
  {
    int sigue;

    sigue=0;
    switch (num)
      {
        case 1: do
                  {
                    if (strcmp(Letra,"1A") == 0)
                      sigue=1;
                    else
                     {
                      if (strcmp(Letra,"1B") == 0)
                        {
                          if (NumImpO == 2)
                            sigue=1;
                          else
                            {
                              gotoxy(1,3);
                              cprintf("Número:  ");
                              gets(Letra);
                              strupr(Letra);
```

```
                    }
                  }
                 else
                  {
                    gotoxy(1,3);
                    cprintf("Número:  ");
                    gets(Letra);
                    strupr(Letra);
                  }
                }

            } while (sigue==0);
          break;

case 2: do
          {
            if (strcmp(Letra,"1A") == 0)
              sigue=1;
            else
             {
              if (strcmp(Letra,"1B") == 0)
                {
                  if (NumImp0 == 2)
                    sigue=1;
                  else
                    {
                      gotoxy(1,3);
                      cprintf("Número:  ");
                      gets(Letra);
                      strupr(Letra);
                    }
                }
              else
                {
                  if (strcmp(Letra,"2A") == 0)
                    sigue=1;
                  else
                   {
                    if (strcmp(Letra,"2B") == 0)
                      {
                        if (NumImp1 == 2)
                          sigue=1;
                        else
                          {
                            gotoxy(1,3);
                            cprintf("Número:  ");
                            gets(Letra);
                            strupr(Letra);
                          }
                      }
                    else
                     {
                       gotoxy(1,3);
                       cprintf("Número:  ");
                       gets(Letra);
                       strupr(Letra);
```

```
                    }
                  }

                }
              }

          } while (sigue==0);

        break;

case 3: do
          {
            if (strcmp(Letra,"1A") == 0)
              sigue=1;
            else
             {
              if (strcmp(Letra,"1B") == 0)
                {
                  if (NumImp0 == 2)
                    sigue=1;
                  else
                    {
                      gotoxy(1,3);
                      cprintf("Número:  ");
                      gets(Letra);
                      strupr(Letra);
                    }
                }
              else
                {
                  if (strcmp(Letra,"2A") == 0)
                    sigue=1;
                  else
                   {
                    if (strcmp(Letra,"2B") == 0)
                      {
                        if (NumImp1 == 2)
                          sigue=1;
                        else
                          {
                            gotoxy(1,3);
                            cprintf("Número:  ");
                            gets(Letra);
                            strupr(Letra);
                          }
                      }
                    else
                     {
                       if (strcmp(Letra,"3A") == 0)
                         sigue=1;
                       else
                         {
                          if (strcmp(Letra,"3B") == 0)
                            {
                              if (NumImp2 == 2)
                                sigue=1;
```

```
                                    else
                                      {
                                        gotoxy(1,3);
                                        cprintf("Número:  ");
                                        gets(Letra);
                                        strupr(Letra);
                                      }
                                   }
                                 else
                                  {
                                    gotoxy(1,3);
                                    cprintf("Número:  ");
                                    gets(Letra);
                                    strupr(Letra);
                                  }
                                }

                              }
                            }

                         }
                       }

                  } while (sigue==0);

                break;

      }
  }

/*********************************************************************/
/* Manda al Servidor de Ficheros los datos de la impresora elegida */
/*********************************************************************/

void MandarEleccionImp(int *fd1)
  {

    if (strcmp(Letra,"1A") == 0)
      {
        if (t_snd(*fd1,Direccion0,sizeof(Direccion0),NULL) == -1)
          {
            cprintf("Se ha producido un error mandando el nombre de la impresora\r\n");
            exit(2);
          }
      }

    if (strcmp(Letra,"1B") == 0)
      {
        if (t_snd(*fd1,Direccion01,sizeof(Direccion01),NULL) == -1)
          {
            cprintf("Se ha producido un error mandando el nombre de la impresora\r\n");
            exit(2);
          }
      }

    if (strcmp(Letra,"2A") == 0)
```

```
      {
        if (t_snd(*fd1,Direccion1,sizeof(Direccion1),NULL) == -1)
          {
            cprintf("Se ha producido un error mandando el nombre de la impresora\r\n");
            exit(2);
          }
      }

    if (strcmp(Letra,"2B") == 0)
      {
        if (t_snd(*fd1,Direccion11,sizeof(Direccion11),NULL) == -1)
          {
            cprintf("Se ha producido un error mandando el nombre de la impresora\r\n");
            exit(2);
          }
      }

    if (strcmp(Letra,"3A") == 0)
      {
        if (t_snd(*fd1,Direccion2,sizeof(Direccion2),NULL) == -1)
          {
            cprintf("Se ha producido un error mandando el nombre de la impresora\r\n");
            exit(2);
          }
      }

    if (strcmp(Letra,"3B") == 0)
      {
        if (t_snd(*fd1,Direccion21,sizeof(Direccion21),NULL) == -1)
          {
            cprintf("Se ha producido un error mandando el nombre de la impresora\r\n");
            exit(2);
          }
      }

  }

/************************************************************/
/* Se reciben las impresoras totales disponibles en la red */
/************************************************************/

void RecivirImpresorasDisponibles(int *fd2)
  {
    int   fila,flags;

    fila=8;
    flags=0;

    if (num >= 1)
          {
            while (continua == 1)
              {
                if (t_rcv(*fd2,Direccion0,sizeof(Direccion0),&flags)!=-1)
                  {
                    gotoxy(3,fila++);
                    cprintf("1A.%s\r",strupr(Direccion0));
```

```
            NumImp0=1;   /* Contabiliza el número de impresoras */
          }              /* de un sólo Servidor.                */

        if (t_rcv(*fd2,Direccion01,sizeof(Direccion01),&flags)!=-1)
          {
            if (strcmp(Direccion01,"FIN") == 0)
              break;
            gotoxy(3,fila++);
            cprintf("1B.%s\r",strupr(Direccion01));
            NumImp0=2;
          }

        if (t_rcv(*fd2,Direccion,sizeof(Direccion),&flags)!=-1)
          {
            if (strcmp(Direccion,"FIN") == 0)
              break;
          }

      } /* Fin del while (continua == 1) */

  }  /* Fin del if (num >= 1) */

if (num >= 2)
  {
    while (continua == 1)
      {
        if (t_rcv(*fd2,Direccion1,sizeof(Direccion1),&flags)!=-1)
          {
            gotoxy(3,fila++);
            cprintf("2A.%s\r",strupr(Direccion1));
            NumImp1=1;   /* Contabiliza el número de impresoras */
          }              /* de un sólo Servidor.                */

        if (t_rcv(*fd2,Direccion11,sizeof(Direccion11),&flags)!=-1)
          {
            if (strcmp(Direccion11,"FIN") == 0)
              break;
            gotoxy(3,fila++);
            cprintf("2B.%s\r",strupr(Direccion11));
            NumImp1=2;
          }

        if (t_rcv(*fd2,Direccion,sizeof(Direccion),&flags)!=-1)
          {
            if (strcmp(Direccion,"FIN") == 0)
              break;
          }

      } /* Fin del while (continua == 1) */
  }

if (num >= 3)
  {
    while (continua == 1)
      {
        if (t_rcv(*fd2,Direccion2,sizeof(Direccion2),&flags)!=-1)
```

```
                {
                  gotoxy(3,fila++);
                  cprintf("3A.%s\r",strupr(Direccion2));
                  NumImp2=1;   /* Contabiliza el número de impresoras */
                }              /* de un sólo Servidor.               */

              if (t_rcv(*fd2,Direccion21,sizeof(Direccion21),&flags)!=-1)
                {
                  if (strcmp(Direccion21,"FIN") == 0)
                    break;
                  gotoxy(3,fila++);
                  cprintf("3B.%s\r",strupr(Direccion21));
                  NumImp2=2;
                }

              if (t_rcv(*fd2,Direccion,sizeof(Direccion),&flags)!=-1)
                {
                  if (strcmp(Direccion,"FIN") == 0)
                    break;
                }

            } /* Fin del while (continua == 1) */
        }

  }

void VentanaParaElegirOpciones(void)
  {
    int opcion,nopciones=3;

    _setcursortype(_NOCURSOR);
    textcolor(15);
    textbackground(1);
    marc();
    window(24,16,54,22);

    for(;;)
      {
        unsigned menu(struct cargamenu *menuprincipal,int nopciones);
        opcion=menu(menuprincipal,nopciones);

        switch(opcion)
          {
             case 0:         clrscr();
                             textcolor(14);
                             cprintf("\r      Impresión  Remota    \n");
                             cprintf("\r      ─────────────────    \n");
                             textcolor(15);
                             cprintf("\r Esta opción se encargará de\n");
                             cprintf("\r imprimir su fichero en la  \n");
                             cprintf("\r impresora del Servidor Remoto.\n");
                             textcolor(14);
                             cprintf("\r    <Pulse cualquier tecla>  ");
                             textcolor(15);
                             getch();
                             strcpy(Letra,"B");
```

```
                        clrscr();
                        return;

            case  1:    clrscr();
                        cprintf("\r Esta opción le hará regresar\n");
                        cprintf("\r al comienzo del programa.  \n\n");
                        textcolor(14);
                        cprintf("\r    <Pulse cualquier tecla>  ");
                        textcolor(15);
                        getch();
                        strcpy(Letra,"C");
                        clrscr();
                        return;

            case  2:    clrscr();
                        textcolor(14);
                        cprintf("\r       Busqueda  Remota   \n");
                        cprintf("\r       ---------------    \n");
                        textcolor(15);
                        cprintf("\r Esta opción se encargará de\n");
                        cprintf("\r la busqueda de ficheros en\n");
                        cprintf("\r Servidores Remoto.\n");
                        textcolor(14);
                        cprintf("\r\n   <Pulse cualquier tecla>");
                        textcolor(15);
                        getch();
                        strcpy(Letra,"A");
                        clrscr();
                        return;

            default :   getch();
                        clrscr();
                        exit(0);

          }  /* Fin switch */

      }  /* Fin for(;;) */

  } /* Fin void */

void LimpiarPantalla(void)
  {
    int  fil;

    textbackground(0);
    textcolor(0);
    fil=1;
    while (fil != 50)
      {
        fil++;
        gotoxy(1,fil);
        cprintf("                                        ");
        cprintf("                                        ");
      }
  }
```

```
unsigned menu(struct cargamenu *menuprincipal,int nopciones)
  {
    static int opcion=0;
    int i;
    int tecla,col,fil;

    for(;;)
      {
        for (i=0;i<nopciones;i++)
          {
            if (opcion==i)
              modovideo(INVERSO);
            else
              modovideo(NORMAL);

            gotoxy(menuprincipal[i].x,menuprincipal[i].y);
            cprintf("%s",menuprincipal[i].titulo);
          }

        tecla=getch();
        switch(tecla)
          {
             case TECLAESPECIAL:  tecla=getch();
                                  switch(tecla)
                                    {
                                      case TECLAARRIBA:
                                      case TECLAIZQUIERDA: if(opcion==0)
                                                             opcion=nopciones-1;
                                                           else
                                                             opcion--;
                                                           break;
                                      case TECLADERECHA:
                                      case TECLAABAJO  :   if(opcion==nopciones-1)
                                                             opcion=0;
                                                           else
                                                             opcion++;
                                                           break;
                                      default          :  break;

                                    }   /* Fin switch(tecla) */

                                  break;

             case TECLAINTRO  :   modovideo(NORMAL);
                                  return(opcion);

             default :            for (i=0;i<nopciones;i++)
                                    {
                                      if (tecla==menuprincipal[i].letra)
                                        {
                                          modovideo(NORMAL);
                                          return(opcion=i);
                                        }
                                    }
          } /* Fin switch*/
```

```
    } /* Fin for(;;)*/

  } /* Fin menu()*/

void modovideo(int modo)
  {
    if (modo==NORMAL)
      {
        textcolor(15);
        textbackground(1);
      }
    else
      {
        textcolor(14);
        textbackground(1);
      }
  }

void marc(void)
  {
    textcolor(14);
    gotoxy(23,15); cprintf("╔═══════ OPCIONES ═══════╗");
    gotoxy(23,16); cprintf("║                        ║");
    gotoxy(23,17); cprintf("║                        ║");
    gotoxy(23,18); cprintf("║                        ║");
    gotoxy(23,19); cprintf("║                        ║");
    gotoxy(23,20); cprintf("║                        ║");
    gotoxy(23,21); cprintf("║                        ║");
    gotoxy(23,22); cprintf("║                        ║");
    gotoxy(23,23); cprintf("╚════════════════════════╝");
    textcolor(15);
  }
```

LISTADO SERVIDOR REMOTO

```
#include <stdlib.h>
#include <stdio.h>
#include <dos.h>
#include <conio.h>
#include <string.h>
#include <bios.h>
#include <alloc.h>
#include <direct.h>     /* _chdrive.. */

#include <ctype.h>
#include <fcntl.h>
#include <tispxipx.h>
#include <tiuser.h>
#include <dir.h>
#include <process.h>
#include <dirent.h>
#include <io.h>         /* filelength, close... */

/* Typedefs */

typedef unsigned char BITE;                    /* nos construimos un byte */
typedef unsigned int PALABRA;
typedef BITE BOOL;                              /* como BOOLEAN en Pascal */
typedef union vel far * VP;                  /* VP es un puntero FAR a la VRAM */
typedef void (*OAFP)(void);                 /* puntero a función sin argumentos */
typedef void (*SHKFP)( PALABRA KeyMask, BITE ScCode );     /* TsrSetHotkey */

/* Macros */

#ifndef MK_FP
#define MK_FP(seg, ofs) ((void far *) ((unsigned long) (seg)<<16|(ofs)))
#endif
#define VOFS(x,y) ( 80 * ( y ) + ( x ) )
#define VPOS(x,y) (VP) ( vptr + VOFS( x, y ) )

/* Estructuras y Unions */

struct velb {            /* describe la posición de pantalla como 2 bytes */
            BITE caractr,                            /* el código ASCII */
                 atributo;                       /* el atributo asociado */
           };

struct velw {              /* describe una posición de pantalla a 1 Word */
            PALABRA contdo;        /* acoge un carácter ASCII y atributo */
           };

union vel {                         /* describe una posición de pantalla */
          struct velb h;
          struct velw x;
         };

/* incluye las funciones del módulo ensamblador */

extern void TsrInit( BOOL tc, void (*fkt)(void), unsigned heap );
extern BOOL TsrIsInst( BITE i2F_fktnr );
extern void TsrUnInst( void );
```

```
extern OAFP TsrSetPtr( void far *fkt );
extern BOOL TsrCanUnInst( void );
extern void TsrCall( void );
extern void far TsrSetHotkey( PALABRA keymask, BITE sccode );

/* constantes y macros */

#define   TC           TRUE
#define   KeyAvail()   ( bioskey(1) != 0 )
#define   GetKey()     bioskey(0)

#define   I2F_CODE     0xC4                  /* número de función INT 2F */
#define   I2F_FKT_0    0xAA             /* Código para INT 2F, Función 0 */
#define   I2F_FKT_1    0xBB             /* Código para INT 2F, Función 1 */

#define   NOF          0x07                          /* color normal */
#define   INV          0x70                         /* color inverso */
#define   HNOF         0x0f                   /* color normal claro */
#define   HINV         0xf0                  /* color inverso claro */

#define   HEAP_FREE    1024        /* dejar 1 KByte de espacio en el Heap */

#define   TRUE         ( 0 == 0 )  /* Constantes para el trabajo con BOOL */
#define   FALSE        ( 0 == 1 )

#define   TECLAESC     27
#define   ESTADO       2
#define   SPX_SOCKET   31                          /* Socket estático */
#define   fa2          748
#define   do1          396

/* variables globales */

VP             vptr;       /* puntero al primer carácter de la RAM de vídeo */
unsigned       atimes = 0;       /* número de activaciones del programa TSR */
union vel      * scrbuf;/* puntero al buffer con el contenido de la pantalla */
char           * blankline;               /* puntero a una línea en blanco */

char           Puerto[10];                       /* Puerto de la impresora */
char           NomImp[16],Despacho[16],ModeloImp[16];
int            PUERTO;

char           NombreFicheroBusca[13];       /* Nombre del fichero a buscar */
int            UnidadBusquedaC,             /* Unidades locales de búsqueda */
               UnidadBusquedaD,
               UnidadBusquedaF,UnidadBusquedaE;

char           UnidadBusqueda[4];
char           LugarEncuentroFichero[300];
FILE           *fp,*fich;             /* Datos impresora y fichero temporal */
int            fd,flags,handle;
IPX_ADDR       spx_addr;                                    /* Para TLI */
SPX_OPTS       spx_options;
struct t_call  tcall;
char           buf[ 100 ], buf2[ 100 ],iobuf[132],           /* Buffers */
               Eleccion[132],Temporal[20];
```

```
int             i1,EI,aviso,CompruebaImpresora,
                temp=0,tecla;

int             unidad,encontrado,encontrado1 = 0;
char            path_original[MAXPATH];
char            path[80];
char            path2[80];
char            path3[80];
char            path4[80];
char            path5[80];

void    BuscaDirectorio2(void);
void    BuscaDirectorio3(void);
void    BuscaDirectorio4(void);
void    BuscaDirectorio5(void);
void    BuscaDirectorio(void);
void    Busca(void);
void    UnidadesLogicasDisponibles(void);
void    BuscarFichero(void);
int     PasaDireccion( char *direccion, IPX_ADDR *destino );
int     PonDireccion( char *cadena, char *buf, int hexBytes );
void    RazonDesconeccionSPX( int fd );
void    ImprimirFichero(char iobufI[132]);
void    VentanasPantalla(void);
int     EstadoImpresora(void);
void    FondoPantalla(void);
void    CabeceraPantalla(void);
void    Nota(unsigned int frec);
void    Inipit(unsigned int frec);
void    AbortarConexion(void);
void    Menu(void);
void    EntradaDatosImp(void); /* Para recoger los datos de la impresora */
void    tsr(void);

/* Obtiene la dirección base de la RAM de vídeo */

void disp_init(void)
  {
    union REGS regs;            /* reg. de procesador para llamada de int.*/

    regs.h.ah = 15;            /* número de función: obtener modo de vídeo*/
    int86(0x10, &regs, &regs);  /* llamar interrupción de vídeo de la BIOS*/

    /* calcular direcc. base de RAM de vídeo según modo de vídeo */

    vptr = (VP) MK_FP((regs.h.al == 7) ? 0xb000 : 0xb800, 0);
  }

/* Visualiza un String en la pantalla */

void disp_print(BITE column, BITE linea, BITE color, char * string)
  {
    register VP lptr;           /* puntero para acceder a la RAM de vídeo */

    lptr = VPOS(column, linea);         /* fijar puntero a la RAM de vídeo */
    for ( ; *string ; ++lptr)             /* recorrer el String hasta NUL */
```

```
      {
        lptr->h.caractr = *(string++); /* escribir carácter en RAM de vídeo */
        lptr->h.atributo = color;       /* fijar atributo para el carácter */
      }
  }

/* Guarda el contenido de la pantalla en un buffer  */

void save_screen( union vel * sptr )
  {
    register VP lptr;          /* puntero para acceder a la RAM de vídeo */
    unsigned i;                                    /* contador de bucle */

    lptr = VPOS(0, 0);                /* fijar puntero a la RAM de vídeo */

    for (i=0; i<2000; i++)    /* recorrer las 2000 posiciones de pantalla */
      (sptr++)->x.contdo = (lptr++)->x.contdo; /* guardar carácter y atributo */
  }

/* Copia el contenido de un buffer en la RAM de vídeo. */

void restore_screen( union vel * sptr )
  {
    register VP lptr;          /* puntero para acceder a la RAM de vídeo */
    unsigned i;                                    /* contador de bucle */

    lptr = VPOS(0, 0);                /* fijar puntero a la RAM de vídeo */

    for (i=0; i<2000; i++)   /* recorrer las 2000 posiciones de pantalla */
      (lptr++)->x.contdo = (sptr++)->x.contdo; /* recuperar car.y atributo */
  }

/* Se llama durante la reinstalación del programa   */

void far endfkt( void )
  {
    /* liberar de nuevo los buffers alojados */

    free( blankline );           /* liberar de nuevo los buffers alojados */
    free( (void *) scrbuf );     /* liberar de nuevo los buffers alojados */
  }

/* Obtiene el final actual del Heap en dependencia del compilador   */

void far *GetHeapEnd( void )
  {
    return (void far *) sbrk(0);
  }

/*************************************************************************/
/*                          PROGRAMA PRINCIPAL                           */
/*************************************************************************/

void main ( int argc, char *argv[] )
  {
    int fil,col;
```

```
if ( argc != 2 || !PasaDireccion( argv[ 1 ], & spx_addr ) )
  {
    printf("El usuario ha de introducir en notación hexadecimal:\n" );
    printf("  -Nodo de red\n" );
    printf("  -Dirección del servidor\n" );
    printf("El formato de la orden será el siguiente:\n");
    printf("   cliente 00001234/00001b02362e\n");
    printf("Si estos datos son desconocidos para usted ejecute SYSCON\n");
    printf("y entre en 'File Server Information'\n");
    exit( 1 );
  }

/* Para el menú de pantalla */

Menu();

/* Se abre un extremo de conexión */

if ((fd = t_open( "/dev/nspx", O_RDWR, NULL) ) == -1 )
  {
    textcolor(15);
    gotoxy(40,7);
    cprintf( "Error abriendo el fichero /dev/nspx\n" );
    AbortarConexion();
    gotoxy(1,24);
    printf("\n");
    exit( 2 );
  }

/* Enlaza el extremo de conexión una vez abierto con una dirección */

if (t_bind(fd,NULL,NULL)== -1)
  {
    textcolor(15);
    gotoxy(40,7);
    cprintf("El enlace ha fallado\n" );
    AbortarConexion();
    gotoxy(1,24);
    printf("\n");
    exit( 2 );
  }

/* Se inicializan las diferentes estructuras TLI */

spx_addr.ipxa_socket[ 0 ] = 0;
spx_addr.ipxa_socket[ 1 ] = SPX_SOCKET;
tcall.addr.buf = ( char * )&spx_addr;
tcall.addr.len = sizeof( spx_addr );
tcall.addr.maxlen = sizeof( spx_addr );
spx_options.spx_connectionID[ 0 ] = 0;
spx_options.spx_connectionID[ 1 ] = 0;
spx_options.spx_allocationNumber[ 0 ] = 0;
spx_options.spx_allocationNumber[ 1 ] = 0;
tcall.opt.buf = ( char * )&spx_options;
tcall.opt.len = sizeof( spx_options );
tcall.opt.maxlen = sizeof( spx_options );
```

```
tcall.udata.buf = ( char * )0;
tcall.udata.len = 0;
tcall.udata.maxlen = 0;
flags=0;

if ( t_connect( fd, &tcall, &tcall ) == -1 )
  {
    textcolor(15);
    gotoxy(40,7);
    cprintf( "La función t_connect ha fallado\n" );
    AbortarConexion();
    gotoxy(1,24);
    printf("\n");
    exit( 2 );
  }

textcolor(15);
gotoxy(40,7);
cprintf( "t_connect se ha realizado correctamente\n" );

EI=EstadoImpresora();
textbackground(1);
textcolor(15);

/* Un aviso sonoro por un código de error en la impresora */

if (EI==-1)
  for (aviso=1;aviso<=4;aviso++)
    {
      Nota(fa2);
      Nota(do1);
    }

/* Ejecuta el bucle hasta que el error de impresora desaparezca */

do
  {

    gotoxy(2,9);
    textcolor(13);
    cprintf("  ......REVISE LA IMPRESORA.......");
    delay(1000L);
    gotoxy(2,9);
    cprintf("                                   ");
    delay(500L);
    EI=EstadoImpresora();

  } while (EI==-1);

/* Se crea una ventana para visualizar la llegada del fichero */

window(3,11,77,22);

/* Entrada de datos de la impresora */

EntradaDatosImp();
```

```
    if ((fp=fopen("DATO_IMP.TXT","r")) == NULL)
      cprintf("El fichero no existe\r\n");
    else
      {
        while (fgets(iobuf,sizeof(iobuf),fp) != NULL )
          {
            if (t_snd(fd,iobuf,sizeof(iobuf),NULL) == -1)
              cprintf("Error mandando características de impresora\r\n");
          }
      }

    fclose(fp);
    strcpy(iobuf,"FIN");

    if (t_snd(fd,iobuf,sizeof(iobuf),NULL) == -1)
      cprintf("Error mandando FIN características de la impresora\r\n");

    textbackground(0);

    window(1,1,80,25);

    for (fil=1;fil<=25;fil++)
      for (col=1;col<=80;col++)
        cprintf(" ");

    clrscr();

    if ( !TsrIsInst( I2F_CODE ))          /*¿programa ya está instalado?*/
      {
        printf( "El programa se instaló residente.\n" );

        /* alojar buffer para gestión de pantalla */

        scrbuf = (union vel *) malloc(80 * 25 * sizeof(union vel));
        blankline = (char *) malloc( 80 + 1 );             /*alojar buffer*/
        *(blankline + 80 ) = '\0';              /*terminar el buffer con 0*/
        memset(blankline, ' ', 80);        /*rellenar el buffer con espacios*/
        TsrInit( TC, tsr, HEAP_FREE );              /*instalar el programa*/

      }
    else                                  /*programa ya estaba instalado*/
      {
        if ( TsrCanUnInst() )
          {
            (*(OAFP) TsrSetPtr(endfkt))();
            TsrUnInst();
            printf( "El programa se desinstaló con éxito.\n" );
          }
        else
          printf( "El programa no se puede desinstalar.\n" );

      }

  }

void RutaOriginal(void)
```

```
  {
    unidad = getdisk();
    getcwd(path_original,MAXPATH);
  }

void ReponerRutaOriginal(void)
  {
    setdisk(unidad);
    chdir(path_original);
  }

void Busca(void)
  {
    int          okf;
    struct       ffblk  f;

    okf = findfirst("*.*",&f,FA_ARCH);
    while (okf == 0)
      {
        if (strcmp(NombreFicheroBusca,f.ff_name) == 0)
          {
            encontrado = 1;
            encontrado1 = 1;
            return;
          }

        okf = findnext(&f);
      }
  }

void BuscaDirectorio(void)
  {
    int          okd = 0;
    struct       ffblk  d;

    /* Primero buscamos en el raiz */

    chdir("\\");
    encontrado1 = 0;
    Busca();
    if (encontrado == 1)
      {
        strcat(LugarEncuentroFichero,UnidadBusqueda);
        strcat(LugarEncuentroFichero,NombreFicheroBusca);
        strcat(LugarEncuentroFichero,";");
        encontrado1 = 1;
      }

    /* Ahora buscamos dentro de los directorios del raiz, haciendo una */
    /* primera bajada en el árbol de directorios. */

    okd = findfirst("*.*",&d,FA_DIREC);
    while (okd == 0)
      {
        if (d.ff_attrib == (char)16)
          {
```

```
            strcpy(path,UnidadBusqueda);
            strcat(path,d.ff_name);
            if (chdir(path))
              perror("Error cambiando directorio");

            encontrado = 0;
            Busca();
            if (encontrado == 1)
              {
                strcat(LugarEncuentroFichero,UnidadBusqueda);
                strcat(LugarEncuentroFichero,d.ff_name);
                strcat(LugarEncuentroFichero,"\\");
                strcat(LugarEncuentroFichero,NombreFicheroBusca);
                strcat(LugarEncuentroFichero,";");
                encontrado1 = 1;
              }

            /* Segunda bajada en el nivel de directorios */

            BuscaDirectorio2();

          }
        okd = findnext(&d);
      }
  }

void BuscaDirectorio2(void)
  {
    int           okd = 0;
    struct        ffblk  d;

    okd = findfirst("*.*",&d,FA_DIREC);
    while (okd == 0)
      {
        if (d.ff_attrib == (char)16)
          {
            if (strcmp(d.ff_name,".") == 0)
              {
                okd = findnext(&d);
                continue;
              }
            if (strcmp(d.ff_name,"..") == 0)
              {
                okd = findnext(&d);
                continue;
              }
            strcpy(path2,path);
            strcat(path2,"\\");
            strcat(path2,d.ff_name);
            if (chdir(path2))
              perror("Error cambiando directorio");
            encontrado = 0;
            Busca();
            if (encontrado == 1)
              {
                strcat(LugarEncuentroFichero,path2);
```

```
                strcat(LugarEncuentroFichero,"\\");
                strcat(LugarEncuentroFichero,NombreFicheroBusca);
                strcat(LugarEncuentroFichero,";");
                encontrado1 = 1;
              }
            strcpy(path3,path2);
            BuscaDirectorio3();
          }
        strcpy(path2,path);
        okd = findnext(&d);
      }
    chdir(path);
  }

void BuscaDirectorio3(void)
  {
    int           okd = 0;
    struct        ffblk  d;

    okd = findfirst("*.*",&d,FA_DIREC);
    while (okd == 0)
      {
        if (d.ff_attrib == (char)16)
          {
            if (strcmp(d.ff_name,".") == 0)
              {
                okd = findnext(&d);
                continue;
              }
            if (strcmp(d.ff_name,"..") == 0)
              {
                okd = findnext(&d);
                continue;
              }
            strcpy(path3,path2);
            strcat(path3,"\\");
            strcat(path3,d.ff_name);
            if (chdir(path3))
              perror("Error cambiando directorio");
            encontrado = 0;
            Busca();
            if (encontrado == 1)
              {
                strcpy(LugarEncuentroFichero,path3);
                strcat(LugarEncuentroFichero,"\\");
                strcat(LugarEncuentroFichero,NombreFicheroBusca);
                strcat(LugarEncuentroFichero,";");
                encontrado1 = 1;
              }
            strcpy(path4,path3);
            BuscaDirectorio4();
          }
        strcpy(path3,path2);
        okd = findnext(&d);
      }
    chdir(path2);
```

```
  }

void BuscaDirectorio4(void)
  {
    int           okd = 0;
    struct        ffblk  d;

    okd = findfirst("*.*",&d,FA_DIREC);
    while (okd == 0)
      {
        if (d.ff_attrib == (char)16)
          {
            if (strcmp(d.ff_name,".") == 0)
              {
                okd = findnext(&d);
                continue;
              }
            if (strcmp(d.ff_name,"..") == 0)
              {
                okd = findnext(&d);
                continue;
              }
            strcpy(path4,path3);
            strcat(path4,"\\");
            strcat(path4,d.ff_name);
            if (chdir(path4))
              perror("Error cambiando directorio");
            encontrado = 0;
            Busca();
            if (encontrado == 1)
              {
                strcpy(LugarEncuentroFichero,path4);
                strcat(LugarEncuentroFichero,"\\");
                strcat(LugarEncuentroFichero,NombreFicheroBusca);
                strcat(LugarEncuentroFichero,";");
                encontrado1 = 1;
              }
            BuscaDirectorio5();
          }
        strcpy(path4,path3);
        okd = findnext(&d);
      }
    chdir(path3);
  }

void BuscaDirectorio5(void)
  {
    int           okd = 0;
    struct        ffblk  d;

    okd = findfirst("*.*",&d,FA_DIREC);
    while (okd == 0)
      {
        if (d.ff_attrib == (char)16)
          {
            if (strcmp(d.ff_name,".") == 0)
```

```
            {
              okd = findnext(&d);
              continue;
            }
          if (strcmp(d.ff_name,"..") == 0)
            {
              okd = findnext(&d);
              continue;
            }
          strcpy(path5,path4);
          strcat(path5,"\\");
          strcat(path5,d.ff_name);
          if (chdir(path5))
            perror("Error cambiando directorio");
          encontrado = 0;
          Busca();
          if (encontrado == 1)
            {
              strcpy(LugarEncuentroFichero,path5);
              strcat(LugarEncuentroFichero,"\\");
              strcat(LugarEncuentroFichero,NombreFicheroBusca);
              strcat(LugarEncuentroFichero,";");
              encontrado1 = 1;
            }

        }
      strcpy(path5,path4);
      okd = findnext(&d);
    }
  chdir(path4);
}

void tsr(void)
  {

    int STATE;
    encontrado=0;

    t_nonblocking(fd);
    STATE = t_getstate(fd);
    if (STATE != T_DATAXFER)
      return;

    t_blocking(fd);
    if (strcmp(strupr(Eleccion),"BUSQUEDA") == 0)
      {

        t_rcv(fd,NombreFicheroBusca,sizeof(NombreFicheroBusca),&flags);
        strupr(NombreFicheroBusca);
        strcpy(LugarEncuentroFichero,"->");
        BuscarFichero();
        t_snd(fd,LugarEncuentroFichero,sizeof(LugarEncuentroFichero),NULL);

      }  /* Final de if (strcmp(strupr(Eleccion),"BUSQUEDA") == 0) */

    if (strcmp(strupr(Eleccion),"IMPRESION") == 0)
```

```
{
  /* Recibiendo el puerto de la impresora elegida por el cliente */

  t_rcv(fd,Puerto,sizeof(Puerto),&flags);

  /* Para definir el puerto para el test de impresora */

  if (strcmp(Puerto,"LPT1") == 0)
    {
      PUERTO=0;
    }
  else
    {
      if (strcmp(Puerto,"LPT2") == 0)
          PUERTO=1;
    }

  /* La función t_rcv se encarga de recibir el fichero */
  /* en paquetes de 132 caracteres. Los paquetes se    */
  /* visualizan en pantalla y se guardan en un fichero */
  /* temporal para posteriormente ser impresos.        */

  temp++;

  if (temp == 1)
    strcpy(Temporal,"TEMP1.TXT");
  if (temp == 2)
    strcpy(Temporal,"TEMP2.TXT");
  if (temp == 3)
    {
      strcpy(Temporal,"TEMP3.TXT");
      temp = 0;
    }

  fich=fopen(Temporal,"w");

  CompruebaImpresora=0;

  while (t_rcv(fd,iobuf,sizeof(iobuf),&flags)!=-1)
        {
          if (strcmp(iobuf,"THE END") != 0)
            fprintf(fich,"%s",iobuf);
          if (strcmp(iobuf,"THE END") == 0)
            {
              fclose(fich);

              /* Si el fichero está vacío lo borra sin más */

              handle = open(Temporal,O_RDONLY);
              if (filelength(handle) == 0)
                {
                  close(handle);
                  remove(Temporal);
                }
              else    /* El fichero no está vacío */
                {
```

```
                        close(handle);
                        if ((fich=fopen(Temporal,"r")) != NULL)

                            while (fgets(iobuf,sizeof(iobuf),fich) != NULL)
                              {
                                CompruebaImpresora++;
                                ImprimirFichero(iobuf);

                                /* Comprueba la impresora */

                                if (CompruebaImpresora==50)
                                  {
                                    window(1,1,80,24);
                                    EstadoImpresora();
                                    CompruebaImpresora=0;
                                    window(3,11,77,22);
                                  }
                              }

                            fclose(fich);
                            remove(Temporal);
                            strcpy(iobuf,"THE END");
                            ImprimirFichero(iobuf);

                          }

                        strcpy(iobuf,"       ");

                      }

                    break;
                  }

              } /* Fin while (t_rcv(fd,iobuf,sizeof(iobuf),&flags)!=-1) */

          }  /* Final de if (strcmp(strupr(Eleccion),"IMPRESORA") == 0) */

    return;
  }

int PasaDireccion( char *direc, IPX_ADDR *destino )
  {
    if ( strlen( direc ) == 21 && direc[ 8 ] == '/' )
    if ( PonDireccion( direc, destino->ipxa_net, 4 ) )
    if ( PonDireccion( &direc[ 9 ], destino->ipxa_node, 6 ) )
      return 1;
    return 0;
  }

int PonDireccion( char *cadena, char *buf, int hexBytes )
  {
    int   i, j, valor;
    char  c;

    for ( i = 0; i < hexBytes; i++ )
      {
```

```
        valor = 0;
        for ( j = 0; j < 2; j++ )
          {
            valor <<= 4;
            if ((c = toupper(*cadena)) >= '0' && c <= '9')
              valor += c - '0';
            else
              if (c >= 'A' && c <= 'F')
                valor += c - 'A' + 10;
              else return 0;
                cadena++;
          }
        *buf++ = valor;
      }

    return 1;
  }

void RazonDesconeccionSPX( int fd )
  {

    struct t_discon    discon;
    char               *msg;

    if ( t_rcvdis( fd, &discon ) == -1 )
      {
        t_error( "La función t_rcvdis ha fallado" );
        exit( 2 );
      }
    switch( discon.reason )
      {
        case TLI_SPX_CONNECTION_FAILED:
          msg = "La conexión ha fallado";
        break;

        case TLI_SPX_CONNECTION_TERMINATED:
          msg = "La conexión ha sido terminada por el cliente";
        break;

        case TLI_SPX_MALFORMED_PACKET:
          msg = "Error de interface, malformación de paquete";
        break;

        default:
          msg = "Causa no recogida";
      }
    printf( "Conexión SPX finalizada: %s\n", msg );

  }

void ImprimirFichero(char iobufI[132])
  {

    FILE  *prn_ptr;

    prn_ptr=fopen(Puerto,"wt");
```

```
    if (strcmp(iobufI,"THE END") == 0)
      fprintf(prn_ptr,"\x0C");            /* Avance de página */
    else
      fprintf(prn_ptr,"%s",iobufI);

    fclose(prn_ptr);
    return;

  }

/*************************************************************************/
/* Esta función se encarga del control de la impresora. Dá su estado  */
/* antes de mandar algo a imprimir , y  en el caso de  que exista     */
/* algún tipo de error manda el código de error correspondiente y una */
/* señal sonora.                                                       */
/*************************************************************************/

int EstadoImpresora(void)
  {

    int printer;

    /* Esta variable almacena el estado de la impresora */

    printer=biosprint(ESTADO,0,PUERTO);

    textcolor(15);

    if (printer & 0x10)
      {
      gotoxy(2,7);
      cprintf("  La impresora está conectada...\r\n");
      }
    else
      {
      gotoxy(2,7);
      cprintf("  La impresora no está conectada.. \r\n");
      gotoxy(2,8);
      cprintf("  o no esta en línea...            \r\n");
      gotoxy(2,9);
      textcolor(13);
      cprintf("  ......REVISE LA IMPRESORA.......");
      VentanasPantalla();
      return -1;
      }

    if (printer & 0x8)
      {
      gotoxy(2,9);
      cprintf("  Error en el dispositivo de I/O  \r\n");
      gotoxy(2,9);
      textcolor(13);
      cprintf("  ......REVISE LA IMPRESORA......");
      VentanasPantalla();
      return -1;
      }
```

```
    if (printer & 0x20)
      {
      gotoxy(2,8);
      cprintf("  La impresora no tiene papel    \r\n");
      gotoxy(2,9);
      textcolor(13);
      cprintf("  ......REVISE LA IMPRESORA......");
      VentanasPantalla();
      return -1;
      }
    else
      {
      gotoxy(2,8);
      cprintf("  La impresora tiene papel       \r\n");
      }

    if (printer & 0x80)
      {
      gotoxy(2,9);
      cprintf("  Impresora libre                \r\n");
      }
    else
      {
      gotoxy(2,9);
      cprintf("  Impresora ocupada              \r\n");
      }

    return 0;

  }

/*************************************************************************/
/* Se encarga de dibujar las ventanas de la pantalla de presentación */
/*************************************************************************/

void VentanasPantalla(void)
  {
    int col,fil;

    textcolor(14);
    gotoxy(1,4);
    cprintf("╒");

    for (col=2;col<=79;col++)
      {
        gotoxy(col,4);
        cprintf("=");
      }

    gotoxy(38,4);
    cprintf("╤");
    gotoxy(80,4);
    cprintf("╕");

    for (col=2;col<=79;col++)
      {
```

```
      gotoxy(col,6);
      cprintf("=");
    }

  for (fil=5;fil<=10;fil++)
    {
      gotoxy(38,fil);
      cprintf("│");
    }

  for (fil=5;fil<=23;fil++)
    {
      gotoxy(1,fil);
      cprintf("│");
    }

  gotoxy(1,6);
  cprintf("╟");
  gotoxy(1,10);
  cprintf("╟");

  for (col=2;col<=79;col++)
    {
      gotoxy(col,10);
      cprintf("=");
    }

  gotoxy(38,10);
  cprintf("╧");

  for (fil=5;fil<=23;fil++)
    {
      gotoxy(80,fil);
      cprintf("│");
    }

  gotoxy(80,10);
  cprintf("╢");
  gotoxy(38,6);
  cprintf("╤");
  gotoxy(1,24);
  cprintf("╚");

  for (col=2;col<=79;col++)
    {
      gotoxy(col,24);
      cprintf("=");
    }

  gotoxy(80,24);
  cprintf("╝");
  gotoxy(80,6);
  cprintf("╢");
  return;

}
```

```
void FondoPantalla(void)
  {

    int fil,col;

    for (fil=1;fil<=24;fil++)
      for (col=1;col<=80;col++)
        cprintf(" ");

  }

void CabeceraPantalla(void)
  {

    int fil,col;

    gotoxy(1,1);
    for (fil=1;fil<=3;fil++)
      for (col=1;col<=80;col++)
        cprintf(" ");

    textcolor(14);
    for (col=1;col<=80;col++)
      {
        gotoxy(col,1);
        cprintf("-");
      }

    gotoxy(1,1);
    cprintf("┌");
    gotoxy(80,1);
    cprintf("┐");
    gotoxy(1,2);
    cprintf("│");
    gotoxy(80,2);
    cprintf("│");

    for (col=1;col<=80;col++)
      {
        gotoxy(col,3);
        cprintf("-");
      }

    gotoxy(1,3);
    cprintf("└");
    gotoxy(80,3);
    cprintf("┘");
    textcolor(14);
    gotoxy(20,2);
    cprintf(".....PROGRAMA DE IMPRESION REMOTA / BUSQUEDA REMOTA.....");

  }

void Nota(unsigned int frec)
  {
    int x,y;
```

```
    Inipit(frec);
    y=inportb(0x61);
    x=y | 0x03;
    outport(0x61,x);
    y=inport(0x61);
    delay(500L);
    x=y & 0xFC;
    outport(0x61,x);
  }

void Inipit(unsigned int frec)
  {
    unsigned num=1193180L / frec;

    outportb(0x43,0xB6);
    outportb(0x42,num & 0x00FF);
    outportb(0x42,num >> 8);
  }

void AbortarConexion(void)
  {
    int aviso;

    for (aviso=1;aviso<=4;aviso++)
          {
            gotoxy(40,9);
            textcolor(13);
            cprintf("                                    ");
            delay(500L);
            gotoxy(40,9);
            cprintf( ".......CONEXION ABORTADA........");
            delay(1000L);
            Nota(fa2);
            Nota(do1);
          }
  }

/* Para el menú de pantalla */

void Menu(void)
  {
    clrscr();
    textbackground(1);
    FondoPantalla();
    VentanasPantalla();
    textbackground(3);
    CabeceraPantalla();
    textbackground(1);
    textcolor(14);
    gotoxy(40,5);
    cprintf("........ ESTADO DE LA CONEXION.........\r\n");
    gotoxy(2,5);
    cprintf(".......ESTADO DE LA IMPRESORA.......\r\r\n\n");
  }

void EntradaDatosImp(void)
```

```
{
  FILE   *imp;
  int    ExisteFicheroDatosImp,NumeroImp,i;
  char   opcion,continua;

  /* Abre el fichero si ya existe */

  ExisteFicheroDatosImp=1;

  if ((imp=fopen("DATO_IMP.TXT","r")) == NULL)
    {
      cprintf("El fichero no existe\r\n");
      ExisteFicheroDatosImp=0;
    }

  else   /* Como existe leer datos y presentar */
    {
      cprintf("Características de su/s impresora/s.\r\n");
      cprintf("————————————————————————————\r\n");

      while (fgets(iobuf,sizeof(iobuf),imp) != NULL )
        {
          cprintf("%s\r",iobuf);
        }

      cprintf("¿Desea cambiar los datos de la impresora? S/N\r\n");
      opcion=getch();
      cprintf("\r");

      if (opcion=='S')
        ExisteFicheroDatosImp=0;
      else
        fclose(imp);

      if (opcion=='s')
        ExisteFicheroDatosImp=0;
      else
        fclose(imp);

    }

  if (ExisteFicheroDatosImp == 0)  /* Si no existe lo crea */
    {
      if ((imp=fopen("DATO_IMP.TXT","w")) == NULL)
        cprintf("No se pudo crear\r\n");
      else
        {
          /* Entrada de datos para la configuración de la impresora */

          cprintf("¿Cuantas impresoras desea instalar? 1 o 2\r\n");
          opcion=getch();
          cprintf("\r");

          i=0;
          do
            {
```

```
    if (opcion == '1')
      {
        NumeroImp=1;
        i=1;
      }
    else
      if (opcion == '2')
        {
          NumeroImp=2;
          i=1;
        }
      else
        {
          cprintf("¿Cuantas impresoras desea instalar? 1 o 2\r\n");
          opcion=getch();
          cprintf("\r");
        }

  } while (i == 0);

for (i=0;i<NumeroImp;i++)
  {
    cprintf("\r");
    cprintf("Elija el tipo (LASER, MATRICIAL, CHORRO TINTA...).\r\n");
    cprintf("Tipo: ");
    gets(NomImp);
    strcpy(iobuf,"TIPO: ");
    strcat(iobuf,strupr(NomImp));

    cprintf("\r");
    cprintf("Introduzca el modelo: ");
    gets(ModeloImp);
    strcat(iobuf," ↔ MODELO: ");
    strcat(iobuf,strupr(ModeloImp));

    cprintf("\r");
    cprintf("Introduzca el despacho o lugar donde se encuentra: ");
    gets(Despacho);
    strcat(iobuf," ↔ DESPACHO: ");
    strcat(iobuf,strupr(Despacho));

    cprintf("\r");
    cprintf("Elija el puerto de su impresora (LPT1 o LPT2).\r\n");
    cprintf("Puerto: ");
    gets(Puerto);
    cprintf("\r");
    strupr(Puerto);

    /* Comprobación de que se ha introducido el puerto correcto */

    continua=0;
    do
      {

        if (strcmp(Puerto,"LPT1") != 0 )
          if (strcmp(Puerto,"LPT2") != 0 )
```

```
                    {
                      cprintf("Puerto: ");
                      gets(Puerto);
                      cprintf("\r");
                    }
                  else
                    continua=1;
                else
                  continua=1;

              } while (continua == 0 );

            strcat(iobuf," ⇔ PUERTO: ");
            strcat(iobuf,Puerto);
            fprintf(imp,"%s\n",iobuf);

          }
        fclose(imp);
      }
    }
  }

void BuscarFichero(void)
  {
    /* Comprobamos la unidades locales existentes */

    UnidadesLogicasDisponibles();

    if (UnidadBusquedaC == 1)  /* Buscamos en unidad C: */
      {
        if (_chdrive(3) != 0)
          perror("No se puede cambiar a C:");
        /*system("C:");*/
        strcpy(UnidadBusqueda,"C:\\");
        RutaOriginal();
        BuscaDirectorio();
        ReponerRutaOriginal();
      }

    if (UnidadBusquedaE == 1)  /* Buscamos en unidad E: */
      {
        if (_chdrive(5) != 0)
          perror("No se puede cambiar a E:");
        /*system("E:");*/
        strcpy(UnidadBusqueda,"E:\\");
        RutaOriginal();
        BuscaDirectorio();
        ReponerRutaOriginal();
      }

    if (UnidadBusquedaD == 1)  /* Buscamos en unidad D: */
      {
        /*system("D:");*/
        if (_chdrive(4) != 0)
          perror("No se puede cambiar a D:");
        strcpy(UnidadBusqueda,"D:\\");
```

```
        RutaOriginal();
        BuscaDirectorio();
        ReponerRutaOriginal();
      }

    return;

  }

/* Comprobar unidades lógicas disponibles para la búsqueda */

void UnidadesLogicasDisponibles(void)
  {
    int   save,disk,disks;
    char  cad[3];

    /* Salvamos la unidad lógica original */

    save=getdisk();

    /* Número de unidades lógicas */

    disks=setdisk(save);

    /* Unidades lógicas disponibles */

    for (disk=0; disk < 26; disk++)
      {
        setdisk(disk);
        if (disk == getdisk())
          {
            sprintf(cad,"%c",disk + 'a');
            if (strcmp(cad,"c") == 0)
              UnidadBusquedaC=1;
            if (strcmp(cad,"d") == 0)
              UnidadBusquedaD=1;
            if (strcmp(cad,"e") == 0)
              UnidadBusquedaE=1;
            /*if (strcmp(cad,"f") == 0)
              UnidadBusquedaF=1;*/
          }
      }

    setdisk(save);

  }
```

LISTADO
MÓDULO ENSAMBLADOR

```
IGROUP group _text              ;Resumen de los segmentos de programa
DGROUP group _bss, _data        ;Resumen de los segmentos de datos
       assume CS:IGROUP, DS:DGROUP, ES:DGROUP, SS:DGROUP

_BSS   segment word public 'BSS'  ;Este segmento acoge todas las variables
_BSS   ends                       ;estáticas no inicializadas

_DATA  segment word public 'DATA' ;Todas las variables globales y estáticas
                                   ;inicializadas se guardan en este
                                   ;segmento

extrn  __psp : word               ;dirección de segmento del PSP del programa C

_DATA  ends

;Constantes

TC_STACK   equ 1536                ;Stack para Turbo C
ARG_WORDS  equ 32                  ;Stack-Words copiados en TsrCall
I2F_FKT_0  equ 0AAh                ;Código para INT 2F, Función 0
I2F_FKT_1  equ 0BBh                ;Código para INT 2F, Función 1
TIME_OUT   equ 9                   ;Time-Out para activación en Ticks

;Programa

_TEXT  segment byte public 'CODE' ;el segmento de datos

;Referencia a funciones (C) externas

extrn      _GetHeapEnd:near            ;develve la dirección final del Heap

;Declaraciones  Public de funciones internas

public     _TsrInit              ;permite la llamada del programa C
public     _TsrIsInst
public     _TsrUnInst
public     _TsrCanUnInst
public     _TsrCall
public     _TsrSetHotkey
public     _TsrSetPtr

;Variables para los controladores de interrupciones
;(sólo accesible mediante el segmento de código

call_ptr   equ this dword
call_ofs   dw 0                   ;Direcc. de offset para TSRCall
call_seg   dw 0                   ;Direcc. de segmento aún no inicializada

ret_ax     dw 0                   ;Guardar el resultado de función en
ret_dx     dw 0                   ;TSR-Call

;variables que neceseitan para la activación del programa C

c_ss       dw 0                   ;Segmento de Stack C
c_sp       dw 0                   ;Puntero de Stack C
c_ds       dw 0                   ;Segmento de datos C
```

```
c_es       dw 0                ;Segmento extra C
c_dta_ofs  dw 0                ;dirección DTA de los programas C
c_dta_seg  dw 0
c_psp      dw 0                ;dirección de segmento del PSP del programa C
break_adr  dw 0                ;dirección Break de Heap
fkt_adr    dw 0                ;dirección de la función TSR del C

;Variables para la activación TSR

tsrnow      db 0                ;¿espera TSR la activación?
tsrn1       db 0
tsrnow2     db 0
tsrnow3     db 0
tsrnow4     db 0
tsrnow5     db 0
tsrnow6     db 0
tsractive   db 0                ;¿está TSR activo?
in_bios     db 0                ;indica actividad de disco BIOS

daptr       equ this dword      ;Puntero al DOS Indos-Flag
daptr_ofs   dw 0                ;Dirección de offset
daptr_seg   dw 0                ;Dirección de segmento

;las siguientes variables acogen las direcciones antiguas de los
;control. de interrupción, que ha sido sustituidos por los nuevos

int8_ptr    equ this dword      ;antíguo vector de interr. 8h
int8_ofs    dw 0                ;dirección de offset del control. antiguo
int8_seg    dw 0                ;dirección de segmento del control. antiguo

int28_ptr   equ this dword      ;vector de interrupción antiguo 28h
int28_ofs   dw 0                ;dirección de offset del control. antiguo
int28_seg   dw 0                ;dirección de segmento del control. antiguo

int2F_ptr   equ this dword      ;vector de interr. antiguo 2Fh
int2F_ofs   dw 0                ;dirección de offset del control. antiguo
int2F_seg   dw 0                ;dirección de segmento del control. antiguo

;Variables, que acogen las informaciones sobre el programa interrumpido


u_dta_ofs   dw 0                ;Dirección DTA del prog. interr.
u_dta_seg   dw 0

u_psp       dw 0                ;Dirección de segmento del PSP del prog. interr.

uprg_ss     dw 0                ;SS y SP del prog. interr.
uprg_sp     dw 0

_TsrInit  proc    near

sframe0     struc               ;Estructura para el acceso al Stack
bp0         dw ?                ;acoge BP
ret_adr0    dw ?                ;dirección de retorno
tc0         dw ?
fktptr0     dw ?                ;puntero a la función TSR de C
```

```
heap0     dw ?                  ;bytes del Heap necesitados
sframe0   ends                  ;Fin de la estructura

frame     equ [ bp - bp0 ]

          push bp               ;Guardar BP en el Stack
          mov  bp,sp            ;Transferir SP aBP

          mov  tsrnow,TIME_OUT
          mov  tsrn1,TIME_OUT
          mov  tsrnow2,TIME_OUT
          mov  tsrnow3,TIME_OUT
          mov  tsrnow4,TIME_OUT
          mov  tsrnow5,TIME_OUT
          mov  tsrnow6,TIME_OUT

          ;guardar el registro de segmentos del C

          mov  c_ss,ss          ;guardar los registros en las
          mov  c_sp,sp          ;guardar variables
          mov  c_es,es
          mov  c_ds,ds

          ;guardar los parámetros pasados

          mov  ax,frame.fktptr0 ;obtener puntero a la función TSR de C
          mov  fkt_adr,ax       ;y guardarlo

          ;obtener la dirección DTA del programa C

          mov  ah,2fh           ;Nº Func.: obtener direcc. DTA
          int  21h              ;llamar interrupción DOS
          mov  c_dta_ofs,bx     ;Dirección en los correspondientes
          mov  c_dta_seg,es     ;guardar variables

          ;Obtener la direcc. del INDOS Flag

          mov  ah,34h           ;Nº Func.: Obtener direcc. INDOS Flag
          int  21h              ;llamar interrupción DOS
          mov  daptr_ofs,bx     ;Dirección en los correspondientes
          mov  daptr_seg,es     ;guardar variables

          ;obtener las direcciones de los control. de interr. a desviar

          mov  ax,3508h         ;obtener vector de interr. 8h
          int  21h              ;llamar interrupción DOS
          mov  int8_ofs,bx      ;guardar la dirección del controlador en
          mov  int8_seg,es      ;las variables correspondientes

          mov  ax,3528h         ;obtener vector de interr. 28h
          int  21h              ;llamar interrupción DOS
          mov  int28_ofs,bx     ;guardar la dirección del controlador en
          mov  int28_seg,es     ;las variables correspondientes

          mov  ax,352Fh         ;obtener vector de interr. 2Fh
          int  21h              ;llamar interrupción DOS
```

```
        mov  int2F_ofs,bx       ;guardar la dirección del controlador en
        mov  int2F_seg,es       ;las variables correspondientes

        ;intalar los nuevos control. de interr.

        push ds                 ;guardar segmento de datos
        mov  ax,cs              ;cargar CS a AX y después a DS
        mov  ds,ax

        mov  ax,2508h           ;Nº Func.: fijar interr. 8h
        mov  dx,offset int08    ;DS:DX acoge la dirección del controlador
        int  21h                ;llamar interrupción DOS

        mov  ax,2528h           ;Nº Func.: fijar interr. 28h
        mov  dx,offset int28    ;DS:DX acoge la dirección del controlador
        int  21h                ;llamar interrupción DOS

        mov  ax,252Fh           ;fijar número de función de la interrupción 2Fh
        mov  dx,offset int2F    ;DS:DX acoge la dirección del controlador
        int  21h                ;llamar interrupción DOS

        pop  ds                 ;recuperar DS del stack

        ;calcular número de párrafos que se han de quedar en la memoria


        call _GetHeapEnd        ;llamar función C en el módulo TSR
        add  ax,frame.heap0     ;sumar memoria Heap necesitada

        ;en TURBO-C el Stack se encuentra detrás del Heap y
        ;comienza con el fin de segmento, por ello se ha de
        ;traer hasta el Heap.

        cmp  byte ptr frame.tc0,0  ;¿Se trabaja con TURBO-C?
        je   msc2                  ;NO, con MSC

        add  ax,TC_STACK-1      ;calcular nuevo puntero del Stack para TC
        mov  c_sp,ax            ;y guardarlo
        inc  ax                 ;fijar dirección Break

        ;calcular número de párrafos que se han de quedar
        ;residentes en memoria

msc2:   mov  dx,ax              ;obtener dirección de Break desde DX
        add  dx,15              ;evitar pérdidas por división de enteros
        mov  cl,4               ;desplazar 4 veces a la derecha y con
        shr  dx,cl              ;ello dividirlo entre 16
        mov  ax,ds              ;obtener DS de AX
        mov  bx,__psp           ;obtener dirección de segmento del PSP
        mov  c_psp,bx           ;guardar en una variable
        sub  ax,bx              ;restar DS del PSP
        add  dx,ax              ;y sumarlo a la cantidad de los párrafos
        mov  ax,3100h           ;Nº Func.: terminar programa residente
        int  21h                ;Llamar interr. DOS y terminar con ello
                                 ;el progama
```

```
_TsrInit  endp

          assume CS:IGROUP, DS:nothing, ES:nothing, SS:nothing


_TsrIsInst proc    near

sframe2    struc                ;Estructura para el acceso al Stack
bp2        dw ?                 ;acoge BP
ret_adr2   dw ?                 ;dirección de retorno
i2F_code2  dw ?                 ;Número de función para INT 2F
sframe2    ends                 ;Fin de la estructura

frame      equ [ bp - bp2 ]

           push bp              ;Guardar BP en el Stack
           mov  bp,sp           ;Transferir SP aBP

           mov  ah,byte ptr frame.i2F_code2 ;Nº func: para INT 2F
           mov  i2F_code,ah                 ;guardarla
           mov  al,I2F_FKT_0                ;nº subfunción
           mov  bx,ax                       ;guardar ambos números
           int  2Fh
           xchg bh,bl                       ;girar números
           cmp  ax,bx                       ;comparar con retorno
           mov  ax,0                        ;no partir de la instal.
           jne  isi_end                     ;diferente --> no instalar

           ;dirección de segmento de la copia ya instalada

           mov  ah,i2f_code       ;No, cargar dirección de segmento
           mov  al,I2F_FKT_1      ;mediante INT 2Fh, subfunción 01h
           int  2Fh
           mov  call_seg,ax       ;y guardar en variables
           mov  ax,-1             ;está instalado

isi_end:   pop  bp                ;BP de nuevo del stack
           ret                    ;de vuelta

_TsrIsInst endp                   ;Fin del procedimiento

tsrlist    db  08h,28h,2Fh,00h ;lista de las INT redireccionadas
                                    ;00h marca el fin
_TsrCanUninst proc  near

           push di                  ;guardar DI (si es variable de registro)
           mov  dx,call_seg         ;cargar segmento de la copia instalada
           mov  di,offset tsrlist-1 ;DI a la lista

tcu_1:     inc  di                ;DI al siguiente nº de Interr.
           mov  al,cs:[di]        ;siguiente nº de interr. a AL
           or   al,al             ;¿Fin de la lista?
           je   tcu_ok            ;Si, todos los vectores OK

           mov  ah,35h            ;Nº de func: para "Get Interrupt"
           int  21h               ;llamar interrupción DOS
```

```
            mov  cx,es              ;ES para comaparar en CX
            cmp  dx,cx              ;¿en el mismo segmento?
            je   tcu_1              ;Si, no es posible reistalación

            xor  ax,ax              ;NO, no es posible reinstalación
            pop  di                 ;recuperar DI
            ret

tcu_ok:     mov  ax,-1
            pop  di                 ;recuperar DI
            ret

_TsrCanUninst endp

_TsrUninst    proc    near

            push ds
            mov  es,call_seg        ;Cargar segmento del TSR instalado

            ;reinstalar los controladores de interrupciones del programa TSR

            cli                     ;no permitir interrupciones
            mov  ax,2508h           ;Nº func: fijar control. a INT 8h
            mov  ds,es:int8_seg     ;dirección de segmento del controlador antiguo
            mov  dx,es:int8_ofs     ;dirección de offset del controlador antiguo
            int  21h                ;instalar de nuevo el controlador antiguo

            mov  ax,2528h           ;Nº Func.: fijar controlador para INT 28
            mov  ds,es:int28_seg    ;dirección de segmento del controlador antiguo
            mov  dx,es:int28_ofs    ;dirección de offset del controlador antiguo
            int  21h                ;instalar de nuevo el controlador antiguo

            mov  ax,252Fh           ;Nº Func.: fijar controlador para INT 2F
            mov  ds,es:int2F_seg    ;dirección de segmento del controlador antiguo
            mov  dx,es:int2F_ofs    ;dirección de offset del controlador antiguo
            int  21h                ;instalar de nuevo el controlador antiguo

            sti                     ;permitir interrupciones

            ;Liberar memoria de nuevo

            mov  es,es:c_psp        ;Direcc. de segm. del PSP del prog. TSR
            mov  cx,es              ;guardar el CX
            mov  es,es:[ 02ch ]     ;direcc. de segm. del environmente del PSP
            mov  ah,49h             ;Liberar memoria alojada
            int  21h                ;llamar interrupción DOS

            mov  es,cx              ;recuperar ES de CX
            mov  ah,49h             ;Liberar memoria alojada
            int  21h                ;llamar interrupción DOS

            pop  ds                 ;recuperar DS y BP del stack
            ret                     ;volver al invocador

_TsrUninst endp                     ;Fin del procedimiento
```

```
_TsrSetPtr  proc    near

sframe3     struc                   ;Estructura para el acceso al Stack
bp3         dw ?                    ;acoge BP
ret_adr3    dw ?                    ;dirección de retorno
fktptr3     dd ?                    ;puntero a la rutina a llamar
sframe3     ends                    ;Fin de la estructura

frame       equ [ bp - bp3 ]

            push bp                 ;Guardar BP en el Stack
            mov  bp,sp              ;Transferir SP aBP

            mov  ax,word ptr frame.fktptr3  ;direcc. de offset a AX
            mov  call_ofs,ax                ;y guardarlo

            mov  ax,offset _TsrCall;devolver puntero Near a TSRCall
            pop  bp                 ;BP de nuevo del stack

            ret                     ;y argumentos del stack

_TsrSetPtr endp                     ;Fin del procedimiento

_TsrCall    proc near

            ;realizar cambio de contexto del programa turbo y
            ;y llamar el procedimiento indicado

            push di                 ;guardar DS, SI y DI
            push si
            push ds

            mov  ah,2fh             ;Nº Func.: obtener direcc. DTA
            int  21h                ;llamar interrupción DOS
            mov  u_dta_ofs,bx       ;guardar dirección del DTA del programa
            mov  u_dta_seg,es       ;interrumpido

            mov  es,call_seg        ;direcc. de segmento del TSR instalado
                                     ;a ES
            mov  ah,50h             ;Nº func.: fijar dirección del PSP
            mov  bx,es:c_psp        ;obtener dirección de segmento del PSP
            int  21h                ;llamar interrupción DOS

            mov  ah,1ah             ;Nº func.: fijar direcc. DTA
            mov  dx,es:c_dta_ofs    ;obtener direcc. de offset del nuevo DTA
            mov  ds,es:c_dta_seg    ;y direcc. de segmento del nuevo DTA
            int  21h                ;llamar interrupción DOS

            ;copiar argumentos al Stack del TSR instalado

            push ss                 ;colocar DS:SI a los aargumentos
            pop  ds                 ;del Stack actual
            mov  si,sp
            add  si,8               ;a través de RET y PUSH DS, DI, SI

            mov  cx,ARG_WORDS*2
```

```
        mov  di,es:c_sp         ;ES:DI al Stack del
        sub  di,cx
        mov  es,es:c_ss
        rep movsb               ;copiar argumentos

        ;ajustar y llamar el registro de segmento del TSR instalado

        mov  es,call_seg        ;direcc. de segmento del TSR instalado
                                 ;a ES

        cli                     ;no permitir interr.
        mov  uprg_ss,ss         ;guardar segm. de stack y puntero
        mov  uprg_sp,sp         ;de stack

        mov  ss,es:c_ss         ;activar el Stack de la copia instalada
        mov  sp,es:c_sp         ;del programa
        sub  sp,ARG_WORDS*2     ;a través de los argumentos
        sti                     ;permitir interrupciones

        mov  ds,es:c_ds         ;fijar registro de segmento para
        mov  es,es:c_es         ;el programa

        call [call_ptr]         ;la función llamada mediante TSRSETPTR
        mov  ret_ax,ax          ;guardar resultado de función
        mov  ret_dx,dx

        cli                     ;no permitir interr.
        mov  ss,uprg_ss         ;conmutar de nuevo al Stack
        mov  sp,uprg_sp         ;propio
        sti                     ;permitir interrupciones

        ;realizar cambio de contexto al programa actual

        mov  ah,1ah             ;Nº func.: fijar direcc. DTA
        mov  dx,u_dta_ofs       ;cargar direcc. de offset y segmento del
        mov  ds,u_dta_seg       ;DTA del programa interrumpido
        int  21h                ;llamar interrupción DOS

        mov  es,call_seg        ;recuperar ES y DS
        pop  ds                 ;recuperar

        mov  ah,50h             ;Nº func.: fijar dirección del PSP
        mov  bx,cs              ;CS a BX
        sub  bx,10h             ;calcular direcc. de segm. del PSP
        int  21h                ;llamar interrupción DOS

        mov  ax,ret_ax          ;recuperar resultado de función
        mov  dx,ret_dx

        pop  si                 ;recuperar registro
        pop  di
        ret                     ;volver al invocador

_TsrCall  endp                  ;Fin del procedimiento
```

```
dosactive   proc near

            push ds                 ;guardar DS y BX en el stack
            push bx
            lds  bx,daptr           ;DS:BX apuntan ahora al INDOS-Flag
            cmp  byte ptr [bx],0    ;¿activa función DOS?
            pop  bx                 ;recuperar BX y DS del stack
            pop  ds

            ret                     ;volver al invocador

dosactive   endp


;El nuevo controlador de interr. 08h (Timer)

int08       proc far

            dec  tsrnow
            cmp  tsrnow,0                   ;¿debe activarse TSR?
            jne  i8_end

            ;TSR debe activarse, pero ¿es posible?

            call dosactive          ;¿se puede interrumpir DOS?
            je   i8_tsr             ;Si, llamar TSR

i8_end:     jmp  [int8_ptr]         ;saltar al controlador antiguo

            ;activar TSR

i8_tsr:     mov  tsrnow,TIME_OUT          ;TSR ya no espera a la activación
            mov  tsractive,1        ;TSR está activo
            pushf                   ;simular llamada del controlador antiguo
            call [int8_ptr]         ;mediante el comando INT 8h
            call start_tsr          ;ejecutar el programa TSR
            iret                    ;volver al programa interrumpido

int08       endp

;el nuevo control. de interr. 28h

int28       proc far

            cmp  tsrnow,0           ;¿espera TSR la activación?
            je   i28_end            ;No, volver al invocador

            cmp  in_bios, 0         ;¿Si, pero está activo el Disk-Intr?
            je   i28_tsr            ;Si, así que no activarlo

i28_end:    jmp  [int28_ptr]        ;al antiguo controlador

            ;Ejecutar TSR

i28_tsr:    mov  tsrnow,0           ;TSR ya no espera a la activación
            mov  tsractive,1         ;TSr está activo
```

```
            pushf                   ;Llamada del control. de interr. antiguo
            call [int28_ptr]        ;simular sobre INT 28h
            call start_tsr          ;ejecutar el programa TSR
            iret                    ;volver al invocador

int28       endp

;El nuevo controlador de interr. 2Fh

int2F       proc far

            cmp  ah,i2F_code        ;¿Llamada a este TSR?
            jne  i2F_end            ;No, al controlador antiguo

            cmp  al,I2F_FKT_0       ;¿Si, es la subfunción 00h?
            je   i2F_0              ;Si, ejecutar

            cmp  al,I2F_FKT_1       ;¿A lo mejor subfunción 01h?
            je   i2F_1              ;Si, ejecutar

            iret                    ;No, ignorar llamada

i2F_end:    ;No es el TSR, continuar llamada

            jmp  [int2F_ptr]        ;al antiguo controlador

i2F_0:      ;Subfunción 00: Installation-Check

            xchg ah,al              ;Intercambiar nº de función y subfunción
            iret                    ;volver al invocador

i2F_1:      ;Subfunción 01: devolver la dirección de segmento

            mov  ax,cs              ;Direcc. de segm. a AX
            iret                    ;volver al invocador

int2F       endp

;START_TSR: activar programa TSR

start_tsr   proc near

            ;realizar cambio de contexto al programa C

            cli                     ;no permitir interr.
            mov  uprg_ss,ss         ;guardar segm. de stack y puntero
            mov  uprg_sp,sp         ;de stack

            mov  ss,c_ss            ;activar el Stack del programa C
            mov  sp,c_sp
            sti                     ;permitir interrupciones

            push ax                 ;los registros de procesador al Stack de C
            push bx                 ;al Stack
            push cx
            push dx
```

```
        push bp
        push si
        push di
        push ds
        push es

        ;guardar 64 Words en el Stack del DOS

        mov  cx,64              ;contador de bucle
        mov  ds,uprg_ss         ;color DS:SI al final del Stack del DOS
        mov  si,uprg_sp

tsrs1:  push word ptr [si]      ;Word del Stack del DOS al Stack del C
        inc  si                 ;y colocar SI a la siguiente
        inc  si                 ;palabra
        loop tsrs1              ;procesar los 64 Words

        mov  ah,51h             ;Nº Func.: obtener la dirección del PSP
        int  21h                ;llamar interrupción DOS
        mov  u_psp,bx           ;guardar la dirección de segm. del PSP

        mov  ah,2fh             ;Nº Func.: obtener direcc. DTA
        int  21h                ;llamar interrupción DOS
        mov  u_dta_ofs,bx       ;guardar dirección del DTA del programa
        mov  u_dta_seg,es       ;interrumpido

        mov  ah,50h             ;Nº func.: fijar dirección del PSP
        mov  bx,c_psp           ;obtener dirección de segmento PSP del Programa C
        int  21h                ;llamar interrupción DOS

        mov  ah,1ah             ;Nº func.: fijar direcc. DTA
        mov  dx,c_dta_ofs       ;obtener direcc. de offset del nuevo DTA
        mov  ds,c_dta_seg       ;y direcc. de segmento del nuevo DTA
        int  21h                ;llamar interrupción DOS

        mov  ds,c_ds            ;registro de segmento para el Programa C
        mov  es,c_es            ;pantalla

        call [fkt_adr]          ;llamar la función icicial del programa C

        ;realizar cambio de contexto del prog. interr.

        mov  ah,1ah             ;Nº func.: fijar direcc. DTA
        mov  dx,u_dta_ofs       ;cargar direcc. de offset y segmento del
        mov  ds,u_dta_seg       ;DTA del programa interrumpido
        int  21h                ;llamar interrupción DOS

        mov  ah,50h             ;Nº func.: fijar dirección del PSP
        mov  bx,u_psp           ;direcc. de segm. PSP del prog. interr.
        int  21h                ;llamar interrupción DOS

        ;restaurar de nuevo el Stack del DOS

        mov  cx,64              ;contador de bucle
        mov  ds,uprg_ss         ;DS:SI con la dirección final del Stack del DOS
        mov  si,uprg_sp         ;del stack DOS
```

```
            add  si,128             ;colocar SI al incio del Stack del DOS
tsrs2:      dec  si                 ;SI a la palabra de Stack anterior
            dec  si
            pop  word ptr [si]      ;traer Word del Stack del C al Stack del DOS
            loop tsrs2              ;procesar los 64 Words

            pop  es                 ;recuperar los registros guardados
            pop  ds                 ;recuperar Stack del C
            pop  di
            pop  si
            pop  bp
            pop  dx
            pop  cx
            pop  bx
            pop  ax

            cli                     ;no permitir interrupciones
            mov  ss,uprg_ss         ;Volver a colocar el puntero de stack y
            mov  sp,uprg_sp         ;el segm. de stack del prog. interr.

            mov  tsractive,0        ;TSR ya no está activo
            sti                     ;permitir llamadas de interr.

            ret                     ;volver al invocador

start_tsr   endp

_text       ends                    ;Fin del segmento de código
            end                     ;Fin del programa
```