TESIS DE LICENCIATURA

ACCESO A INFORMACION LOCAL ENTRE ESTACIONES BAJO NOVELL NETWARE

DIRECTOR
LIC. DANIEL RODRIGUEZ

CODIRECTOR
LIC. OSCAR SORIA

AUTORES

EMILSE SILVINA PEREYRO
L.U. 496/86

SILVANA FERNANDA BERTUZZI
L.U. 673/86

DEPARTAMENTO DE COMPUTACION - FCEYN - UBA

1996

Indice General

RESUMEN

Novell NetWare es un sistema operativo de red diseñado para gestionar la comunicación de datos entre varias computadoras personales. Las mismas pueden trabajar sobre el disco rígido del servidor como si fuera el disco local, pero no pueden tener acceso al disco local de otra estación de trabajo.

El objetivo de la presente tesis es acceder, a través de Novell NetWare, desde una estación de trabajo a la información almacenada en el disco local de otra.

Para ello crearemos nuevos comandos que ampliarán los ya existentes del NetWare. Estos realizarán las siguientes funciones:

- Visualizar la información acerca de archivos y directorios del disco local de otra estación de trabajo.

- Copiar archivos desde el disco local de la estación de trabajo origen al disco local de la estación de trabajo destino o viceversa.

Estos comandos establecerán una conexión con la aplicación residente en la estación de trabajo remota usando el protocolo SPX. Esta aplicación será la encargada de buscar la información requerida por el comando y transmitirla a la estación a través de la red o copiar la información que se le envía desde la estación remota en un archivo.

ABSTRACT

Novell NetWare is a network operative system designed to manage data communications among several PCs. These personal computers can work on the server's hard disk as if it were its own disk drive, but they can not access the disk drive of another workstation.

The objective of this thesis is, by means of Novell Netware, to access, from one workstation, the information stored in the disk drive of another workstation.

Thus we will create new commands that will enhance the Netware's existing ones. These will perform the following functions:

- Display information about the files and directories of another workstation's disk drive.

- Copy files from the sourse workstation's disk drive to that of the receiving workstation or viceversa.

These commands will connect to the remote workstation's aplication via SPX protocol. This application will be in charge of searching for the information required by the command and transmitting in to the workstation through the network, or else copying the information form the remote workstation into a file.

1
INTRODUCCION A REDES DE COMPUTADORES

En un pasado no muy lejano, cuando las computadoras eran muy costosas, las organizaciones no podían entregar a todo el personal un computador para uso individual. En lugar de ello la unidad central de proceso (U.C.P.) tenía que estar compartida; los sistemas de computadores estaban muy centralizados, usualmente en el interior de un cuarto. La idea de que en el futuro sería posible adquirir computadores más pequeños, pero igualmente poderosos, era una locura.

En el esquema representado en la figura, el centro de actividad de cálculo está localizado en la unidad central de proceso, en el computador principal.

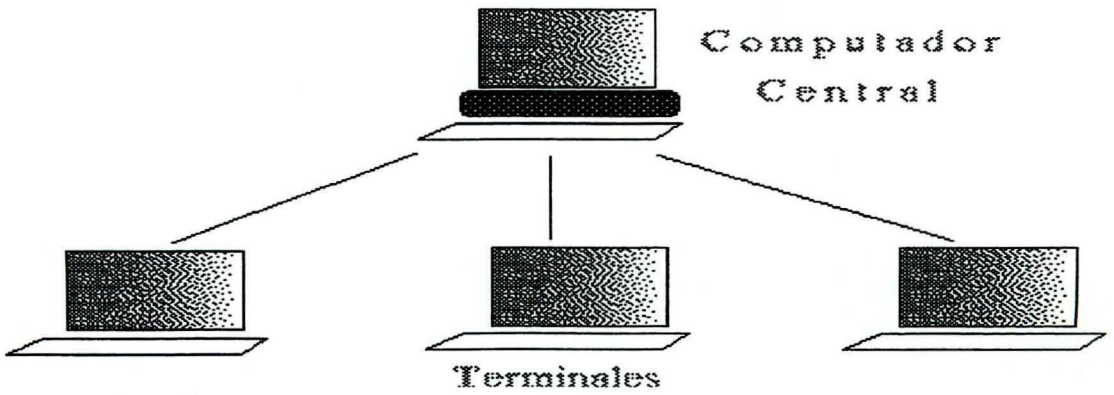


Fig. 1.1 Computador Central

Los usuarios acceden a él a través de terminales satélites (llamadas terminales "bobas", porque no realizan ningún proceso por sí mismas). El propósito básico de una terminal "boba" es proporcionar una interfaz entre el computador principal y los usuarios.

Este modelo tiene la ventaja de proporcionar un proceso centralizado, pero presenta algunas limitaciones:

- El concepto de un solo computador grande haciendo todo el trabajo.

- La idea de que los usuarios traigan su trabajo al computador en lugar de llevar el computador a donde se encuentran los usuarios.

- Los usuarios del procesador central están limitados por las aplicaciones residentes en el mismo, por lo tanto, éstas deben satisfacer las necesidades de todos.

Con la llegada de la tecnología del silicio y el correspondiente abaratamiento de los procesadores, los computadores personales pudieron colocarse en puestos de trabajo individuales.

La mayor fuerza de los computadores personales reside en la independencia y la libertad que proporcionan. Este carácter tan personal es también su debilidad, por ejemplo:

- Los datos de una computadora personal podrían ser compartidos si más de una persona pudiera acceder simultáneamente.

- Los usuarios de una PC gastan la mayoría de su tiempo en componer y crear, y relativamente poco tiempo en usar sus dispositivos perisféricos (impresora, plotter, etc.). Se torna caro tener un dispositivo especializado por PC.

- Desde el punto de vista de la seguridad de los datos, es inconveniente, y a veces riesgozo, tener diferentes individuos que utilizan una misma PC, ya que se tiene acceso a los mismos archivos.

- La copia de seguridad de los datos puede resultar demasiado costosa y consumir mucho tiempo si se realiza en cada PC. Debido a ello esta medida de precaución muchas veces no se lleva acabo, no pudiéndose recuperar la información en caso de pérdida.

Lo que sería interesante lograr es obtener los beneficios que brinda una PC y los que proporciona el procesamiento centralizado.

La fusión de los computadores y las comunicaciones permite reemplazar el modelo centralizado por un gran número de computadores separados, pero interconectados, que efectúen el mismo trabajo. Estos sistemas se conocen como redes de computadores.

Una **red de computadores es una colección interconectada de computadores autónomos.** Se dice que dos computadores están interconectados si éstos son capaces de intercambiar información. La conexión puede hacerse a través de cables, láser, microondas y satélites de comunicaciones. Al indicar que los computadores son autónomos se excluye a los sistemas en los que existe una relación maestro-esclavo (si un computador puede forzosamente arrancar, parar o controlar a otro, éstos no son autónomos). Un sistema constituido por una unidad de control y muchos esclavos no es una red.

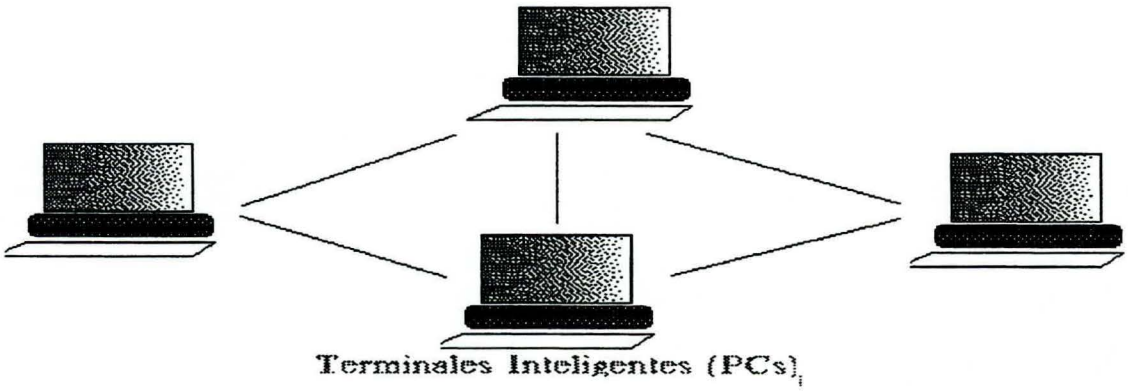


Fig. 1.2 Terminales Inteligentes (PCs)

Existe un solapamiento entre una red de computadores y un sistema distribuido.

En un **sistema distribuido** la existencia de múltiples computadores autónomos es transparente al usuario. Este, no tiene conocimiento de la existencia de múltiples procesadores, ve al sistema como un monoprocesador virtual. La asignación de trabajos al procesador, el acceso a archivos en discos, el movimiento de archivos entre donde se almacenan y donde son necesarios, y todas las demás funciones del sistema, deben ser automáticas.

Con una **red de computadores**, el usuario debe explícitamente entrar en una máquina, explícitamente enviar trabajos remotos, explícitamente mover archivos y, por lo general, gestionar de manera personal toda la administración de la red. Con un sistema distribuido nada se tiene que hacer en forma explícita, todo lo hace de manera automática el sistema sin que el usuario tenga conocimiento de ello.

Un sistema distribuido es un caso especial de una red, aquél cuyo software da un alto grado de cohesividad y transparencia. La diferencia entre un sistema distribuido y una red está dada por el software, en especial por el sistema operativo.

1.1 Objetivo de las Redes

Las redes proporcionan una solución tanto a las limitaciones de los computadores personales aislados como a los entornos de los computadores centralizados.

Sus principales objetivos son:

- **Compartir Recursos**, lo que implica hacer que todos los programas, datos y equipos estén disponibles para cualquier usuario de la red que así lo solicite, sin importar su localización física ni la del recurso. Otro aspecto de compartir recursos es el relacionado con la división de la carga.

- **Alta Fiabilidad**, ya que cuenta con fuentes alternativas de suministros. Por ejemplo, todos los archivos podrían duplicarse en dos o tres máquinas, de tal manera que si una de ellas no se encuentra disponible (como consecuencia de una falla del hardware), podría utilizarse alguna de las otras copias. Además, la presencia de múltiples CPUs permite la posibilidad de que si una de ellas deja de funcionar, las otras puedan ser capaces de encargarse de su trabajo.

- **Ahorro Económico**. Los ordenadores pequeños tienen una mejor relación costo-rendimiento comparada con la ofrecida por las máquinas grandes. Los sistemas de red

generalmente están diseñados con poderosas computadoras personales, una por usuario, y una o más máquinas dedicadas como servidoras de archivos compartidos.

- **Aumento del rendimiento del sistema en forma progresiva,** a medida que crece la carga de trabajo se pueden sumar más procesadores. En las computadoras centralizadas cuando el sistema alcanza su máximo rendimiento, se debe reemplazar por uno más grande, lo cual genera un gran gasto.

Medio de comunicación entre personas.

1.2 Diferencia entre LAN y WAN

Existen dos clases distintas de red. Aquellas que tienen **varios computadores localizados en el mismo edificio,** cuya denominación es **LAN** (red de área local); y las llamadas **WAN** (red de área extendida), que permite las comunicaciones entre **computadores separados por una gran distancia geográfica.**

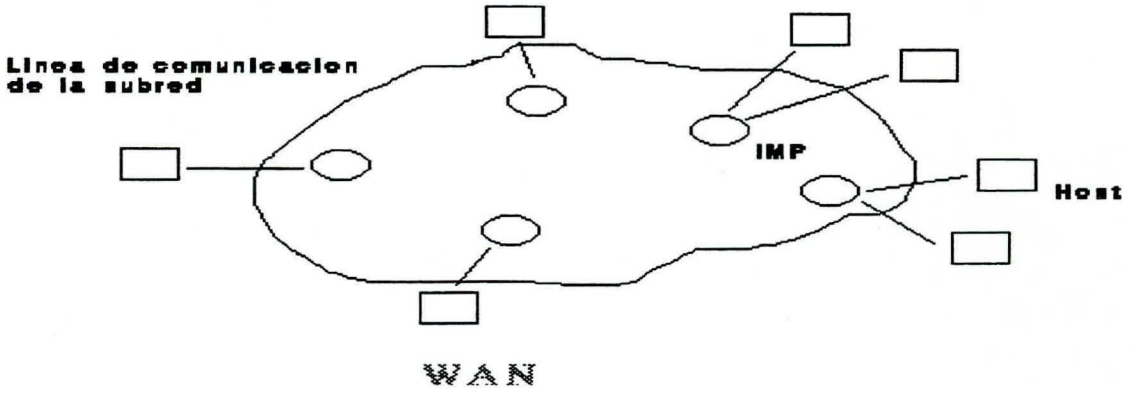


Fig. 1.3 Red de Area Extendida (WAN)

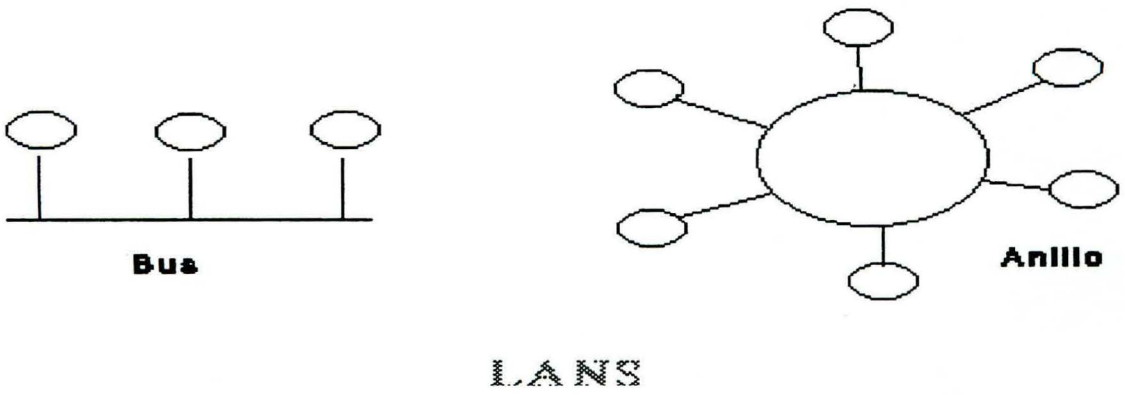


Fig. 1.4 Redes de Area Local (LAN)

Las LAN tienen características particulares:

- Un campo de acción de unos cuantos kilómetros.
- Una velocidad total de datos de, por lo menos, varios Mbps.
- Una pertenencia a una sola organización.

En contraposición, las WAN:

- Abarcan países enteros.
- Tienen uan velocidad de datos inferior a 1 Mbps.
- Pertenecen a múltiples organizaciones.

La necesidad de una red LAN surge cuando se desea conectar entre sí distintas computadoras, cuando existe la posibilidad de un aumento progresivo de la carga de trabajo, o bien, cuando se quiere obtener una mejor relación costo-rendimiento de una red de estaciones de trabajo.

La mayoría de las LAN y un número reducido de WAN utilizan **canales de difusión**. Los sistemas de difusión tienen un solo canal de comunicación que, a su vez, es compartido por todas las máquinas que constituyen la red. Los paquetes que una máquina

cualquiera envía, son recibidos por todas. En el momento en que se recibe un paquete, se verifica si el paquete está destinado a ella, sino se ignora.

En cambio, la mayor parte de las redes WAN utilizan el diseño de **canales de comunicación punto a punto**. La red contiene varios cables o líneas telefónicas alquiladas, conectando cada una de ellas un par de IMP (computadores especializados que se utilizan para conectar dos líneas de transmisión). Si dos IMP desean comunicarse y no comparten un cable común, deberán hacerlo a través de un tercero. Cuando un mensaje se envía de un IMP a otro, a través de uno o más IMP intermedios, el paquete se recibe integramente en cada uno de éstos; se almacena y no continúa su camino hasta que la línea de salida necesaria para reexpedirlo esté libre.

La diferencia clave entre las WAN y las LAN es que en las primeras los diseñadores están obligados por razones de tipo legal, económico y político, a utilizar las redes públicas de teléfonos existentes, sin importar su conveniencia técnica. A los diseñadores de las LAN, nada les impide tender su propio cable con un gran ancho de banda.

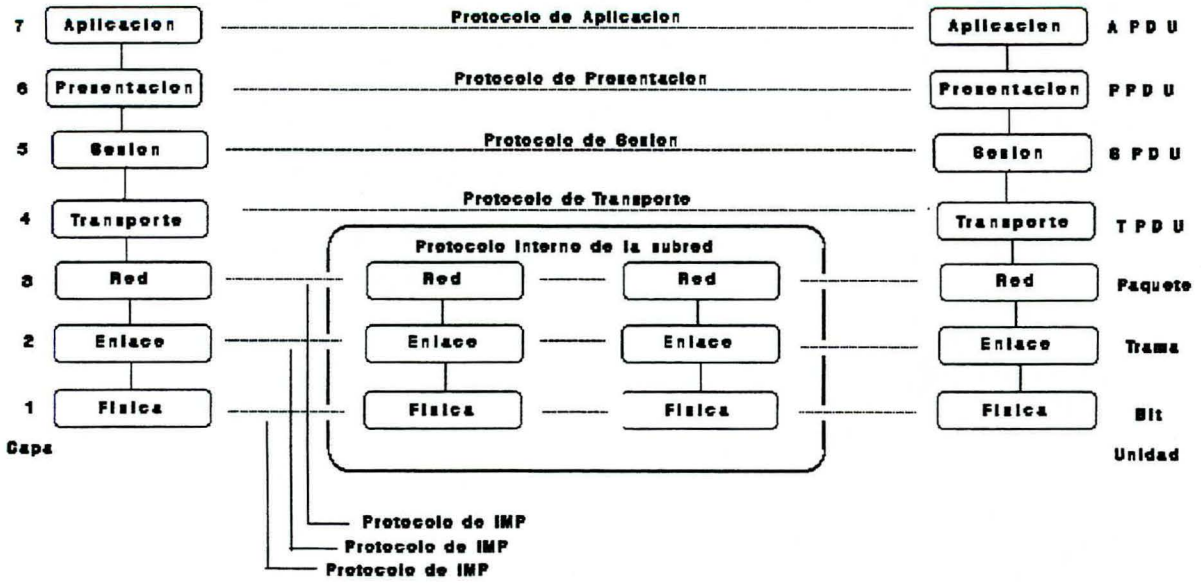
1.3 Arquitectura de Redes

La mayor parte de las redes se organizan en capas o niveles con el objetivo de reducir la complejidad de su diseño. El número de capas, el nombre, contenido y función varían de una red a otra. Sin embargo, en cualquier red, el propósito de cada capa es ofrecer ciertos servicios a las capas superiores, liberándolas del conocimiento detallado sobre cómo se realizan dichos servicios.

La capa **n** en una máquina conversa con la capa **n** de otra máquina. Las reglas y convenciones utilizadas en esta conversación se conocen como **protocolo de la capa n**. A las entidades que forman las capas correspondientes en máquinas diferentes se las denomina **"peer process"**, procesos pares. Son éstos los procesos que se comunican mediante el uso del protocolo.

Cada capa pasa la información de datos y control a la capa inmediatamente inferior, y así sucesivamente hasta que se alcanza la capa más baja de la estructura. Debajo de la capa 1 se encuentra el medio físico, a través del cual se realiza la comunicación real.

Entre cada par de capas adyacentes hay una interfase, la cual define los servicios y operaciones primitivas que la capa inferior ofrece a la superior. Cada capa efectúa un conjunto específico de funciones bien definidas. **Al conjunto de capas y protocolos se le denomina arquitectura de red.**



Arquitectura de Red OSI

Fig. 1.5 Arquitectura de Red OSI

1.3.1 Modelo de Referencia OSI

En los primeros tiempos en que apareció el concepto de redes, cada compañía fabricante de computadores tenía sus propios protocolos. Como consecuencia los usuarios que adquirían computadores de diferentes compañías, no podían conectarlos y establecer una sola red con ellos. A éstos se los denominó **sistemas cerrados.**

El caos generado por la incompatibilidad dió lugar a la exigencia por parte de los usuarios de una normalización al respecto. Los sistemas que se ajustan a un conjunto de reglas y métodos que se han convertido en un estándar en la industria o en el posible mercado se los llama **sistemas abiertos.**

En los últimos años, ha tomado forma de estándar de redes el modelo desarrollado por la Organización Internacional de Normas (ISO), denominado, **OSI** (Interconexión de Sistemas Abiertos), ya que se refiere a la comunicación que se puede establecer entre distintos sistemas.

El modelo OSI por sí mismo no es una arquitectura de red, puesto que no especifica, en forma exacta, los servicios y protocolos que se utilizarán en cada una de las capas. Sólo indica lo que debe hacer cada capa.

1- Capa Física:

Se ocupa de la transmisión de un flujo de bits a lo largo de un canal de comunicación. Los problemas de diseño a considerar aquí son los aspectos mecánicos, eléctricos, de procedimientos de interfase y el medio de transmisión física.

2- Capa de Enlace:

Su tarea consiste en transformar un medio de transmisión común en una línea sin errores de transmisión para la capa de red.

Esta tarea la logra haciendo que el emisor divida la entrada de datos en tramas de datos, las transmita en forma secuencial y procese las tramas de asentimiento devueltas por el receptor. Corresponde a la capa resolver los problemas causados por daño, pérdida o duplicidad de tramas.

También, deberá evitar que un transmisor muy rápido sature con datos a un receptor más lento (regulación de flujo).

3- Capa de Red:

Se encarga del control de la operación de la subred. Cómo encaminar los paquetes del origen al destino.

Si demasiados paquetes están presentes en la subred pueden dar lugar a un cuello de botella, corresponde a esta capa ocuparse del control de congestión.

Además, resuelve problemas de interconexión de redes heterogéneas (protocolos diferentes).

En las redes de difusión el problema de ruteo es simple, por lo cual el trabajo a realizar por la capa sería muy sencillo e incluso ésta podría no existir.

4- Capa de Transporte:

Su función principal consiste en aceptar los datos de la capa de sesión, dividirlos, siempre que sea necesario, en unidades más pequeñas, pasarlos a la capa de red y asegurar que todos ellos lleguen correctamente al otro extremo. Esto se debe realizar de tal forma que aisle la capa de sesión de los cambios inevitables a los que está sujeta la tecnología del hardware.

Crea una conexión de red distinta para cada conexión de transporte solicitada por la capa de sesión. Si la conexión de transporte necesita un gran caudal, ésta podría crear múltiples conexiones de red para mejorar dicho caudal. Si la conexión o mantenimiento de la conexión de una red resulta costoso, la capa de transporte podría multiplexar varias conexiones de transporte sobre la misma conexión de red.

También determina qué tipo de servicio debe dar a la capa de sesión, y en último término a los usuarios de la red. El servicio puede ser: entregar los mensajes sin error en el mismo orden en que fueron enviados, o entregar mensajes aislados sin garantizar el orden de distribución, como así también, la difusión de mensajes a destinos múltiples.

Es una capa de tipo origen-destino o extremo a extremo. Un programa en la máquina origen lleva una conversación con un programa parecido en la máquina destino , utilizando las cabeceras de los mensajes y los mensajes de control. Los protocolos de las capas inferiores son entre cada máquina y su vecino inmediato, y no entre las máquinas origen y destino.

También se encarga del establecimiento y liberación de conexiones a través de la red.

Además, tiene un un mecanismo para regular el flujo de información.

5- Capa de Sesión:

Permite que los usuarios de diferentes máquinas puedan establecer sesiones entre ellos.

Realiza la gestión del control de diálogo. Las sesiones permiten que el tráfico vaya en ambas direcciones al mismo tiempo, o bien, en una sola dirección en un instante dado.

Para algunos protocolos se debe evitar que ambos lados traten de realizar la misma operación en el mismo instante. En estos casos, la capa usa testigos que pueden ser intercambiados.

Otro de sus servicios es la sincronización. La capa inserta puntos de verificación en el flujo de datos, con el fin de que si existiera alguna caída sólo tengan que repetirse los datos que se encuentren después del último punto de verificación.

6- Capa de Presentación:

Se encarga de los aspectos de sintaxis y semántica de la información que se transmite. Facilita la comunicación de computadores con diferentes representaciones. La estructura de los datos que se va a intercambiar puede definirse en forma abstracta, junto con una norma de codificación que se utilice "en el cable". El trabajo de manejar estas estructuras de datos abstractas y la conversión de la representación utilizada en el interior del computador a la representación normal de la red, se lleva a cabo a través de esta capa.

La compresión de datos, la privacidad y la autentificación (criptografía), también son tarea de la capa de presentación.

7- Capa de Aplicación:

Contiene una variedad de protocolos que se necesitan frecuentemente. Por ejemplo, aquellos que se encargan de compatibilizar los distintos tipos de terminales existentes, la transferencia de archivos entre sistemas distintos, el correo electrónico, la entrada de trabajo a distancia, el servicio de directorio y otros servicios de propósito general y específico.

En la figura se muestra un ejemplo de cómo pueden transmitirse los datos mediante el empleo del modelo OSI. El proceso emisor tiene algunos datos que desea enviar al proceso receptor. Este entrega los datos a la capa de aplicación, la

cual añade entonces la cabecera de aplicación, AH (pudiendo ser nula), a la parte delantera de los mismos y entrega el elemento resultante a la capa de presentación.

La capa de presentación transforma este elemento de diferentes formas, con la posibilidad de incluir una cabecera en la parte frontal, dando el resultado a la capa de sesión. La capa de presentación no sabe qué parte de los datos que le dió la capa de aplicación corresponden a AH, y cuáles son los que corresponden a los verdaderos datos del usuario.

Este proceso se sigue repitiendo hasta que los datos alcanzan la capa física, lugar en donde efectivamente se transmiten a la máquina receptora. En la otra máquina, se van sacando una a una las cabeceras, a medida que los datos se transmiten a las capas superiores, hasta que finalmente llegan al proceso receptor.

La idea esencial, es que si bien la transmisión efectiva de datos es vertical, cada una de las capas está programada como si fuera una transmisión horizontal.

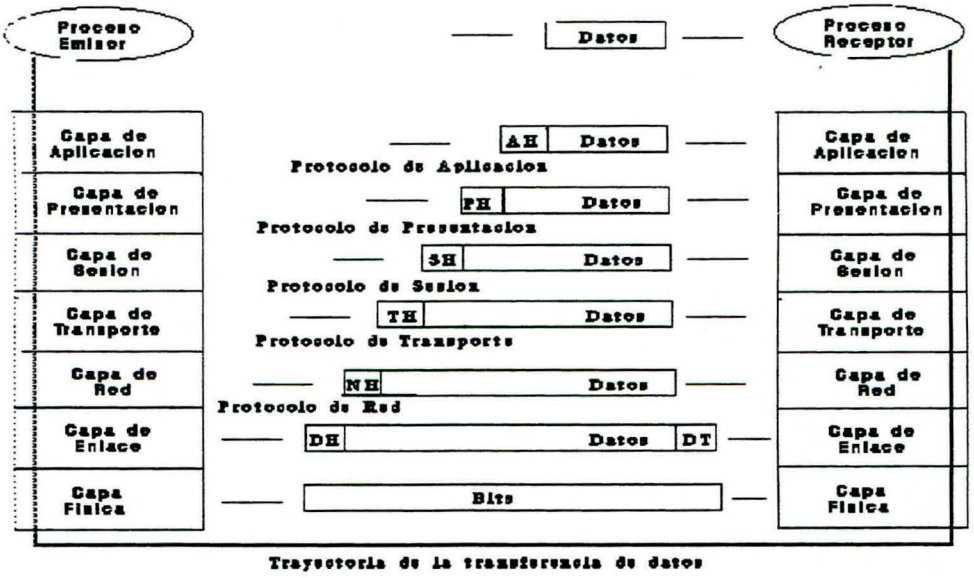


Fig. 1.6 Ejemplo de Arquitectura del Modelo OSI

2
NETWARE NOVELL

NetWare es un sistema operativo multiusuario para computadoras personales, desarrollado por Novell, de Provo, Utah. Fue diseñado para proporcionar una serie de servicios a los usuarios de una Red de Area Local (LAN).

Este permite aprovechar al máximo los recursos limitados del equipo con que se cuenta, ya que se pueden compartir por ejemplo impresoras, tarjetas de fax, instalaciones de almacenamiento en discos, plotters, etc.. No sólo se puede compartir el hardware sino también el software, se puede acceder a las aplicaciones de procesamiento de datos, incluyendo procesadores de palabras, administradores de bases de datos, programas, servicios gráficos. También mejora las comunicaciones entre usuarios de la red.

2.1 Sistema Operativo de una Red Novell

Se ha diseñado para interactuar con otros sistemas operativos, no para reemplazarlos. NetWare soporta sistemas multitarea, como OS/2, PC-MOS, Windows/386 y Concurrent DOS. Todos estos programas permiten ejecutar varias tareas en forma simultánea, utilizando varias ventanas en la estación de trabajo.

NetWare permite la interoperatividad. Las tareas pueden estar distribuidas entre las distintas máquinas, mientras que en cada una se ejecuta su propio sistema operativo.

Quizás el detalle más importante es que NetWare, interactúa con el sistema operativo de la estación de trabajo en lugar de sustituirlo. Tanto el servidor como la estación de trabajo son capaces de realizar tareas por sí mismos o compartir la responsabilidad con otros. Por lo tanto, con NetWare se trabaja en un **entorno operativo distribuido**.

2.1.1 Interoperatividad

NetWare logra la interoperatividad permitiendo que se instale parte de sí mismo (el shell) en la estación de trabajo. El shell del NetWare es el lazo de unión entre el sistema operativo de la estación de trabajo y el NetWare instalado en el servidor. El shell se incorpora en dos programas separados, que deben estar cargados en la estación para poder establecer la comunicación a través de la red.

El primero de estos programas, **IPX.COM**, proporciona el protocolo, o reglas para la comunicación a través de una topología particular de LAN. IPX.COM dirige la información que se transmite entre las estaciones, y entre las estaciones y el servidor. Por lo tanto, debe estar configurado según la topología y las tarjetas de interfase (NIC) de cada una de las estaciones de la red. El IPX.COM de una topología no funcionará en otra distinta.

El segundo de estos programas, **redirector** (en el caso del DOS el progama se llama **NETx.COM**), permite la comunicación con un servidor concreto. En el servidor sólo se ejecuta NetWare. Incluso en el modo no dedicado, el servidor ejecuta DOS como una sesión de usuario independiente, permitiendo ejecutar en un primer plano DOS y cualquier aplicación basada en él, mientras NetWare se ejecuta independientemente en un segundo plano. Si, por ejemplo, esta cargado el programa shell de NetWare, y la combinación de teclas pulsadas es una orden NetWare, los procesos del DOS que se ejecutan en la estación son congelados in situ, permaneciendo residentes. Entonces, se pasa el control de ejecución al shell de NetWare, que envía los mensajes apropiados a las estaciones y/o servidor. Una vez que se ha realizado la función pedida, se devuelve el control al DOS, que descongela su estado anterior y continúa la ejecución de la tarea que estuviera realizando. El redirector tiene las siguientes ventajas:

- Puede ser configurado para un gran número de topologías.
- Está disponible para diversos sitemas operativos.
- Proporciona acceso a muchos de los poderosos servicios de NetWare.

2.2 Componentes de Novell

Los componentes de una red que funciona bajo el NetWare de Novell son los siguientes:

- **Servidores.**
- **Puestos de trabajo con computadoras personales u otros dispositivos inteligentes.**
- **Medios de transmisión.**
- **Distribución y topología de la red**
- **Protocolos de Acceso al Medio**
- **Bridges y gateways**

2.2.1 Servidores

Un servidor de red es una computadora utilizada para gestionar el sistema de archivos. Este brinda servicios a las impresoras de la red, controla las comunicaciones y realiza otras funciones.

Un servidor puede ser dedicado o no. Si es dedicado emplea toda su potencia de procesamiento para realizar funciones de red; si no lo es parte de sus recursos van a ser empleados para ejecutar trabajos como una estación o como una PC.

El sistema operativo de la red está cargado en el disco fijo del servidor junto con las herramientas de administración del sistema y los utilitarios del usuario.

Cada vez que se conecta el sistema de red, NetWare arranca, y el servidor queda bajo su control. A partir de este momento, el sistema operativo DOS no funciona en el mismo, ya que éste funciona bajo el sistema operativo NetWare; sin embargo, la mayoría de los programas del DOS pueden ejecutarse en la forma habitual.

Un aspecto de la seguridad del NetWare proviene del hecho de que las unidades de disco fijo de la red sólo pueden ser activadas o accedidas después de arrancar NetWare, y no DOS. Por lo dicho, las personas que no están autorizadas para entrar en NetWare no podrán acceder a las unidades de disco.

La elección de un servidor de red es crítica para el rendimiento y el funcionamiento de una red. La tarea primordial de un servidor dedicado es procesar los pedidos realizados por la estación de trabajo. Estos pueden ser de acceso a disco, a colas de impresión o de comunicaciones con otros dispositivos. La recepción, gestión y realización de estos pedidos puede requerir un tiempo considerable, que se incrementará de forma paralela al número de estaciones de trabajo activas en la red. Como el servidor gestiona los pedidos de todas las estaciones de trabajo, su carga puede ser muy pesada.

Si las estaciones de trabajo envían pedidos en forma continua se puede llegar a un estado de congestión. El tráfico de la red puede elevarse de manera tal de impedir la recepción de algunos requerimientos enviados por las estaciones. Estas seguirán enviándolos hasta obtener respuesta por parte del servidor. Esto incrementa en mayor grado la carga de trabajo del servidor, ya que tiene que procesar también los intentos adicionales enviados por las estaciones al no obtener respuesta. Además, las estaciones de trabajo empiezan a funcionar más lento al tener que esperar la respuesta enviada por el servidor.

Cuanto más grande es la red, más importante resulta tener un servidor con elevadas prestaciones. Se nesecitan grandes cantidades de memoria RAM para optimizar los accesos a disco y mantener las colas de impresión. El servidor tiene que mantenerse lo más cerca posible del rendimiento previsto. El rendimiento (throughput) de un servidor es una combinación de varios factores, incluyendo el tipo de procesador, la velocidad de éste último, el factor de estados de espera, el tamaño del canal de acceso a memoria, el tamaño del bus, la existencia de memoria cache, así como las prestaciones del disco fijo y otros factores.

Las unidades de disco pueden estar instaladas dentro o fuera del servidor, pero para utilizarse como unidades de almacenamiento de la red tienen que estar conectadas al servidor.

NetWare optimiza la performance del almacenamiento en disco y hace que sea un sistema tolerante a fallas. Un sistema de alimentación ininterrumpida (UPS de uninterruptible power supply) puede proveer protección para fallas de este tipo. Las fallas sobre disco fijo pueden ser desastrosas, los backups deben hacerse diariamente y en algunos casos continuamente en tiempo real usando técnicas de discos espejos o servidores dedicados a backups.

Los periféricos (impresoras, modems y otros dispositivos que pueden ser compartidos por las estaciones de trabajo) están presentes en toda red. Los periféricos compartidos tienen que conectarse al servidor, salvo que se utilice software adicional.

Los modems y otros dispositivos para telecomunicaciones también pueden ser compartidos designando a una estación de trabajo como servidor de comunicaciones. No se recomienda usar el servidor de archivos para esta tarea, ya que los procesos de comunicaciones pueden acaparar una parte considerable del tiempo del servidor.

2.2.2 Estaciones de Trabajo

Las estaciones de trabajo son sistemas inteligentes, tales como computadoras personales IBM o compatibles con unidades de diskette o disco fijo. Las mismas se conectan al servidor a través de la placa de conexión de red y el cableado correspondiente.

El concepto del procesamiento distribuido se basa en que las computadoras personales conectadas a la red realizan su propio procesamiento después de cargar los programas y los datos desde el servidor. Esto libera al servidor para que pueda dedicarse a tareas de la red. Tras el procesamiento, los archivos son almacenados de nuevo en el servidor desde donde pueden ser utilizados por otras estaciones de trabajo, o incluidos en la copia de seguridad realizada a partir del servidor.

El costo de las estaciones de trabajo puede disminuirse utilizando estaciones de trabajo sin discos. Una estación de trabajo sin discos tiene la misma capacidad de memoria y de cómputo que una computadora personal ordinaria. Las estaciones de trabajo sin disco necesitan circuitos especiales en la placa

de red para poder acceder al servidor. Una vez que acceden al servidor estos equipos arrancan desde el disco fijo de éste, igual que lo harían desde el suyo propio. Un archivo especial almacenado en el servidor funciona como una unidad de disco para la estación de trabajo sin disco. Este archivo contiene todos los archivos de arranque y configuración contenidos generalmente en una unidad de arranque, tales como los archivos de sistema del DOS, los archivos AUTOEXEC.BAT, CONFIG.SYS y los archivos de arranque de NetWare. El administrador del sistema crea un archivo de arranque para cada tipo distinto de estación de trabajo después de la instalación de NetWare.

Las placas de conexión a la red de estos equipos tienen que instalarse con una PROM de inicialización remota.

Para conectar una computadora personal a una red, hay que instalar previamente la placa de conexión de red y el cableado adecuado. Después del arranque del sistema, para conectarse lógicamente con el servidor, se utilizan archivos especiales de arranque de la red creados durante el proceso de instalación. Estos archivos son creados de forma específica, dependiendo del tipo de la estación de trabajo y de la placa de red usada en cada uno. Los archivos especiales de arranque de NetWare llevan a cabo, en primer lugar, la conexión física con los dispositivos de la red, tales como la placa de comunicaciones y el cableado; seguidamente, dotan al sistema operativo (DOS) de la estación de trabajo de una interfase software (shell) que le permite comunicarse con el sistema operativo NetWare. En el caso de las estaciones de trabajo sin disco, los archivos de arranque residen en el servidor.

Hay dos razones poderosas por las que se consideran convenientes las estaciones de trabajo sin discos. La primera es, su precio. La segunda está relacionada con la seguridad. Como las estaciones de trabajo no tienen unidades de disco, no existe la posibilidad de descargar archivos confidenciales en diskettes y sacarlos de la instalación. Además, los usuarios no pueden enviar archivos al servidor, lo que puede resultar importante cuando el administrador del sistema desea evitar que los usuarios llenen el servidor con archivos innecesarios. Las estaciones de trabajo sin discos también ofrecen una forma de mantener el sistema libre de virus informáticos.

2.2.3 Medios de Transmisión

Se pueden utilizar distintos medios físicos para transportar un flujo de bits de una máquina a otra.

Estos medios están formados por cables y placas de interfase, los cuales deben ser compatibles.

2.2.3.1 Placas de Interfaz de Red (NIC)

Una placa de red ofrece el conector necesario para unir el cable de la red al sevidor o estación de trabajo. Los circuitos incluidos en la placa suministran los protocolos y las ordenes necesarias para soportar el tipo de red al que está destinada la placa. Muchas placas tienen memoria adicional para almacenar temporalmente los paquetes de datos enviados y recibidos, mejorando el rendimiento de la red. También pueden albergar un zócalo para una PROM de inicialización remota, pudiendo montarse en este caso en una estación sin discos. Las placas de red también tienen varios interruptores y conmutadores (jumpers o minipentes) para seleccionar distintas interrupciones hardware, direcciones de entrada/salida y otras prestaciones que harán que la tarjeta sea compatible con el equipo en que se monta.

Existen placas de conexión a la red que siguen el sistema antiguo de ocho bits y otras que utilizan el nuevo y más rápido bus de dieciseis bits.

El tipo de cable y su distribución en la instalación depende del tipo de placa utilizada.

2.2.3.2 Cables de la Red

Los más populares son:

- Par trenzado
- Cable Coaxil
- Fibra Optica

Par Trenzado

El cable con par trenzado consiste en dos hilos de cobre trenzado, aislados en forma independiente y trenzados entre sí. El par está cubierto por una capa aislante externa. Ofrece las siguientes ventajas:

- Es una tecnología bien estudiada.
- No requiere una habilidad especial para la instalación.
- La instalación es rápida y fácil.
- La emisión de señales al exterior es mínima.
- Ofrece alguna inmunidad frente a interferencias, modulación cruzada y corroción.

Cable Coaxil

El cable coaxil se compone de un hilo conductor de cobre envuelto por una malla trenzada plana que hace las funciones de tierra. Entre el hilo conductor y la malla hay una capa gruesa de material aislante, y todo el conjunto está protegido por una cobertura externa. El cable está disponible en dos espesores. El cable grueso soporta largas distancias, pero es más caro. El cable fino puede ser más práctico para conectar puntos cercanos.

El cable coaxil ofrece las siguientes ventajas:

- Soporta comunicaciones en banda ancha y banda base.
- Es útil para varias señales, incluyendo voz, video y datos.
- Es fácil de instalar.
- Es una tecnología bien estudiada.
- Puede ya estar instalada.

Fibra Optica

Aunque la conexión mediante fibra óptica es cara, permite transmitir la información a gran velocidad e impide la intervención de las líneas. Como la señal es transmitida a través de luz, existen muy pocas posibilidades de interferencias eléctricas o emisión de señal. El cable consta de dos núcleos ópticos, uno interno y otro externo, que refractan la luz en forma distinta. La fibra está encapsulada en un cable protector y ofrece las siguientes ventajas:

- Alta velocidad de transmisión.
- No emite señales eléctricas o magnéticas, lo cual redunda en la seguridad.
- Inmunidad frente a interferencias y modulación cruzada.
- Mayor economía que el cable coaxil en algunas instalaciones.
- Soporta mayores distancias.

2.2.4 Distribución y Topología de la Red

La topología de la red es la forma en que se distribuyen los cables para conectar entre sí el servidor y las estaciones de trabajo. Determina dónde pueden colocarse las estaciones, cuán fácil será tender los medios de transporte y cuál será el costo de todo el sistema de cableado.

2.2.4.1 Topología en Estrella

Se utiliza un dispositivo como punto de conexión de todos los cables que parten de las estaciones de trabajo. El dispositivo central puede ser el servidor de archivo en sí o un dispositivo especial de conexión.

Es fácil el diagnóstico de problemas en la red, debido a que las estaciones de trabajo se comunican a través del equipo central. Las fallas en los nodos se detectan con rapidez.

El cambio de los cables puede realizarse sin inconvenientes, como así también la ampliación del sistema.

La colisión entre datos es imposible, ya que cada estación tiene su propio cable. Sin embargo, en grandes instalaciones, los cables de las estaciones de trabajo tienden a agruparse en una unidad central creando una situación propensa a errores de gestión. En estos casos pueden necesitarse grandes cantidades de cable, así como un servidor dedicado, lo que aumentaría el costo.

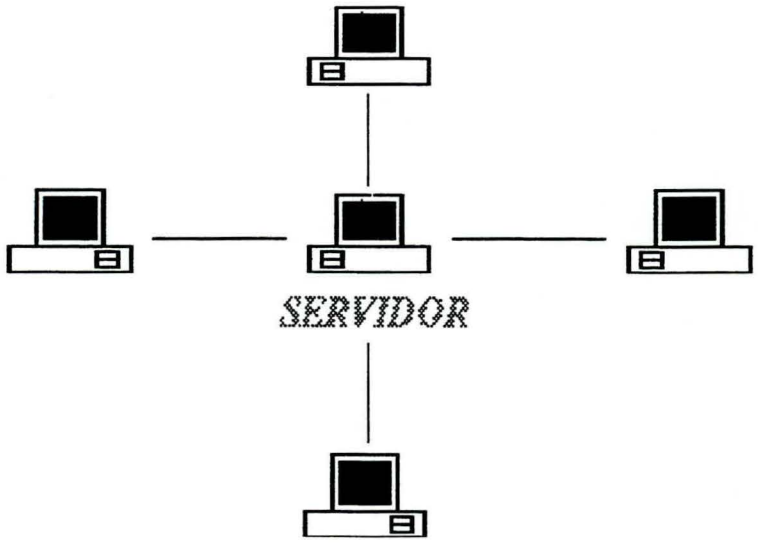


Fig. 2.1 Topología en Estrella

2.2.4.2 Topología en Anillo

Las señales viajan en una única dirección a lo largo de un cable que forma un circulo cerrado. Las mismas pasan de

un nodo a otro. Cada uno de ellos tiene asignada una dirección, la cual debe ser especificada en los datos transmitidos para que lleguen al destino deseado.

Con la topología en anillo, las redes pueden extenderse a menudo a grandes distancias, y el costo total del cableado será menor que en una configuración en estrella y posiblemente igual al de un bus lineal.

Como consecuencia de que el complicado cableado debe cerrarse sobre sí mismo, cualquier rotura del anillo hará caer la red.

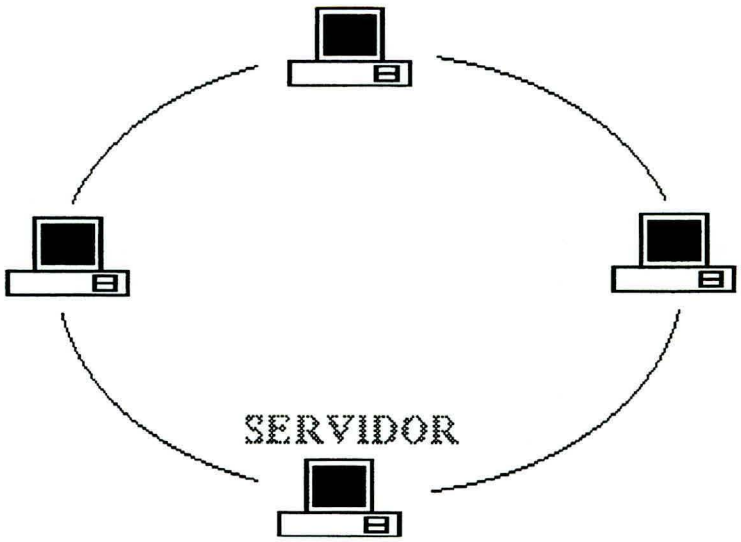


Fig. 2.2 Topología en Anillo

2.2.4.3 Topología de Bus Lineal

El servidor de archivos y todas las estaciones de trabajo están conectados a un cable coaxil general. Todos los nodos comparten este cable. Las señales se transmiten en ambos sentidos asociadas a una dirección de destino. Cada nodo verifica las direcciones de los paquetes que circulan por la red para ver si coincide con la suya propia.

Un ejemplo de ésta topología es la red Ethernet, que utiliza un acceso múltiple por detección de portadora (tono continuo) con detección de colisiones (CSMA/CD). Tiene una velocidad de transmisión de 10 Mbits por segundo.

Existen dos tipos de cable coaxil que se pueden utilizar: grueso y fino. Mientras el cable fino es más económico que el grueso, su máxima longitud es de 185 metros. En cambio, el cable grueso permite una longitud de 500 metros, y es usado a menudo como una columna vertebral para conectar múltiples redes.

El cable fino Ethernet se tiende de una estación de trabajo a otra formando un único tronco lineal. Las terminaciones de cada segmento son ajustadas a un conector tipo BNC, el cual está conectado en T a la tarjeta NIC (Network Interface Card). En las puntas del tronco se ajustan terminadores, uno de los cuales está conectado a tierra.

Las redes Ethernet también pueden construirse usando el cable par trenzado o la fibra óptica.

La topología de bus es fácil de instalar, ya que puede extenderse por un edificio siguiendo las mejores rutas posibles.

Esta topología tiene desventajas. El cable central puede convertirse en un cuello de botella en entornos con tráfico elevado, ya que todas las estaciones de trabajo comparten el mismo cable. También, es difícil aislar los problemas de cableado en la red y determinar qué estación o segmento del cable los origina. Una rotura en el cable hará caer el sistema.

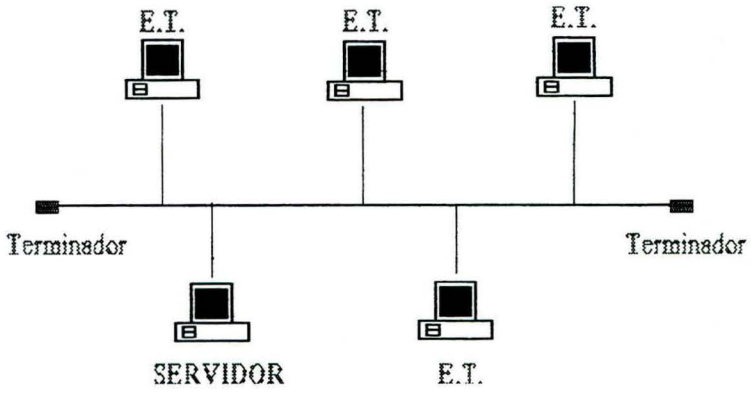


Fig. 2.3 Topología de Bus Lineal

2.2.4.4 Topología Combinada Estrella/Bus

Esta topología tiene un multiplexor de señal (hub) que actúa como dispositivo central. El sistema de cableado de la red puede tomar la topología de bus lineal o de anillo. Este ofrece ventajas en el cableado de edificios que tienen grupos de trabajo separados por distancias considerables.

Un ejemplo de esta topología es la red ARCNET que comúnmente usa cable coaxil. Su topología es de bus, utilizando los hubs para distribuir estaciones de trabajo en una configuración de estrella. Los segmentos de cable vinculan las tarjetas y los hubs usando conectores BNC.

Las estaciones de trabajo ARCNET, para acceder a la red usan el esquema paso de testigo, trama de control que actúa como autorización para enviar datos. El testigo atraviesa un anillo lógico, ya que cada estación tiene asignado un número y el testigo pasa por ellas siguiendo el orden numérico, aunque las estaciones no estén físicamente en orden.

Los hubs pueden ser pasivos o activos. Los nodos pueden estar conectados a 609 metros de un hub activo y a 30 metros de un hub pasivo. Los hubs pasivos usualmente tienen cuatro ports; los activos ocho ports. El concepto de hub provee

una única manera para distribuir las estaciones de trabajo. Por ejemplo, un hub activo puede ser usado para distribuir estaciones dentro de un departamento. Cada hub activo de departamento puede estar conectado a un segmento lineal para crear una red extendida.

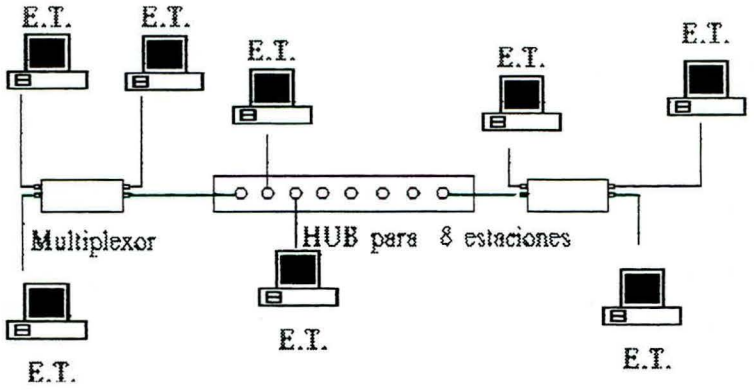


Fig. 2.4 Topología Combinada Estrella/Bus

2.2.4.5 Topología Combinada Estrella/Anillo

Una Topología combinada estrella/anillo usa el esquema de acceso de paso de testigo sobre una topología de anillo. También, el paso de testigo en anillo puede tomar la apariencia de topología de estrella, ya que las estaciones pueden ramificarse desde un hub central, o unidad de acceso a múltiples estaciones (MAU, Multistation Access Unit). Normalmente se usa un cable especialmente protegido, pero también se puede usar par trenzado.

Un ejemplo es la red Token Ring de IBM, que tiene una velocidad de 4 Mbits o 16 Mbits por segundo. La longitud del anillo completo no puede exceder los 366 metros. La distancia máxima desde un MAU a la estación es de 100 metros, usando par trenzado.

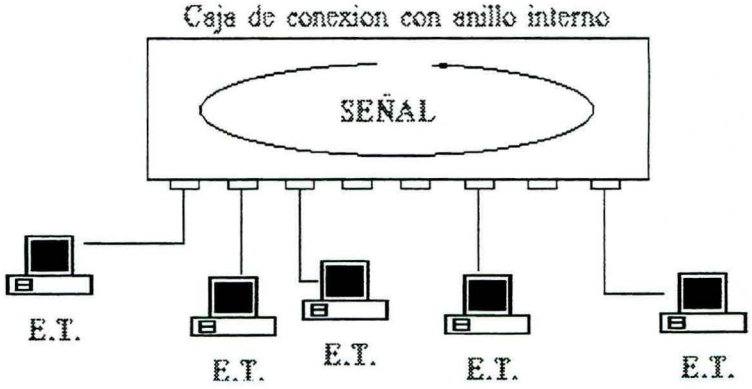


Fig. 2.5 Topología Combinada Estrella/Anillo

2.2.5 Protocolos de Acceso al Medio

Las redes pueden dividirse en dos categorías: las que utilizan conexiones punto a punto y aquellas que utilizan **canales de difusión.** Estos últimos se conocen también como **canales de acceso múltiple.** NetWare Novell es una red de difusión.

El punto clave en este tipo de redes consiste en cómo determinar quién tiene el derecho de utilizar un canal, cuando existe competencia por éste.

Las placas de conexión de red están diseñadas para trabajar con un tipo de topología. Los circuitos de la placa suministran los protocolos para la comunicación entre las estaciones de la red a través del medio de transmisión. Un protocolo de comunicaciones establece cómo y cuándo un nodo puede acceder al medio de transporte y enviar paquetes de datos a otro nodo.

Los protocolos utilizados con más frecuencia en las redes son cuatro. Se diferencian en el punto en que reside el control y en la forma de acceso al cable.

2.2.5.1 Protocolo de Conmutación de Circuito

Tiene un circuito de control encargado de dar a los nodos permiso de acceso a la red.

Cuando un nodo desea transmitir datos, debe solicitar permiso de acceso al circuito de control. Este le será otorgado si la línea no está ocupada. En el momento en que se establece la comunicación entre dos nodos, el circuito de control impide el acceso a la red al resto de las estaciones.

2.2.5.2 Control de Acceso por Sondeo

Un contralor central pregunta a los nodos si quieren o no enviar datos; si, así lo desean, les da permiso para hacerlo.

Este método se diferencia del anterior en que, en aquél los nodos solicitan permiso para enviar datos al dispositivo de control. En éste es el dipositivo de control el que invita a los nodos a enviar datos.

2.2.5.3 Acceso múltiple por detección de portadora (CSMA, de "Carrier Sense Multiple Access")

A aquellos protocolos en los que las estaciones escuchan a una portadora (es decir, una transmisión), y actúan en consecuencia, se les llama protocolos de detección de portadora. Este método se utiliza en las redes con topología de bus.

Los nodos escuchan constantemente la línea para verificar si algún otro está transmitiendo, y si éstos datos están dirigidos a él. Cuando la estación desea enviar alguna información, primero escucha el canal para saber si alguien está transmitiendo; si el canal está ocupado, la estación espera a que se libere.

El retardo de propagación tiene un efecto muy importante en el comportamiento de este protocolo. Existe la posibilidad de que, después que una estación comience a transmitir otra estación llegue a estar lista para hacerlo y escuche el canal. Si la señal correspondiente a la primera estación

todavía no ha alcanzado a la segunda, esta última detectará un canal desocupado, y también empezará a transmitir, dando como resultado una colisión. En este caso las estaciones dejarán de transmitir para reintentarlo posteriormente. Cuanto mayor sea el retardo de propagación, más importante será este efecto.

2.2.5.4 Paso de Testigo

En un esquema de paso de testigo, se envía un testigo (token) a lo largo de la red como trama de control de transmisión. Los nodos pueden utilizarlo, si no está siendo usado, para enviar datos a otros nodos. Como sólo hay un testigo es imposible que haya colisiones, y el rendimiento de la red permanece constante.

2.2.6 Bridges y Gateways

Dentro de cualquier LAN puede haber un dispositivo que la conecte a otra LAN, denominado **bridge**, o a otro sistema operativo, denominado **gateway**.

Los bridge son dispositivos que almacenan y reexpiden tramas de datos. Un bridge acepta una trama completa y la pasa a la capa de enlace en donde se comprueba su código de redundancia; entonces la trama se trasmite a la capa física para que se reexpida hacia una subred diferente.

Los gateways son conceptualmente similares a los bridges, con la única excepción que se localizan en la capa de red.

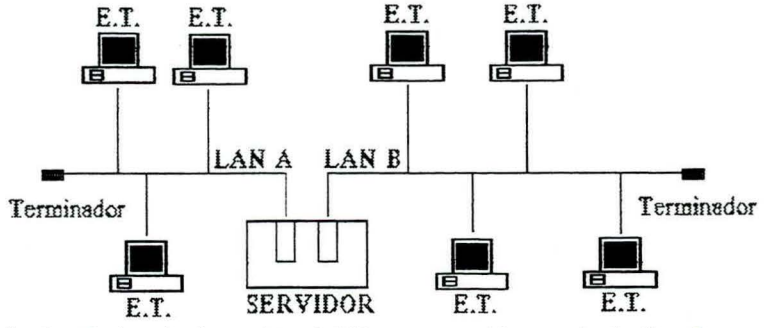
Las redes que están conectadas mediante un gateway pueden diferir más que aquellas conectadas por un bridge. Las redes tipo LAN se conectan mediante bridge, en tanto, que los retransmisores de redes LAN-WAN y WAN-WAN son gateways. Una ventaja de los gateways sobre los bridges, es que pueden conectar redes que tiene formatos de direccionamiento incompatibles.

Las conexiones de LAN con redes de distintos sistemas operativos se realizan generalmente con grandes computadoras o minicomputadoras. El proceso de realizar conexiones entre redes que no tiene las topologías de LAN mencionadas se denomina **internetworking** (interconexión entre redes).

Existe una amplia variedad de métodos dentro del ámbito de las conexiones entre redes. Por ejemplo, podemos conectar una LAN local con una LAN remota a través de una conexión telefónica. También podemos establecer una comunicación entre dos redes locales a través del servidor, lo que se denomina **interconexión interna**, o podemos realizar la conexión a través de una estación de trabajo, realizando entonces una **interconexión externa**. Generalmente se usa una máquina dedicada para una interconexión externa, incrementando de esta manera el rendimiento de la interconexión.

Una de la ventajas del NetWare de Novell es que las interconexiones entre sus LANs pueden ser muy transparentes para el usuario, incluso aunque las redes conectadas utilicen distintos protocolos de red o estén configuradas con topologías diferentes. Un usuario de la red externa puede acceder al servidor a través de un bridge desconociendo prácticamente los métodos de comunicación y el nivel de configuración del cableado.

NetWare ofrece la posibilidad de conectarse con muchos sistemas operativos.



La instalacion de dos tarjetas LAN en un servidor puede duplicar la distancia cubierta por una LAN.

Fig. 2.6

3
PROTOCOLOS DE COMUNICACION PUNTO A PUNTO

Punto a punto significa que las estaciones de trabajo pueden hablar con otra como con un par, ellas pueden enviar paquetes directamente sin necesitar del servidor de archivos. Esta comunicación permite incretar la performance y reducir la sobrecarga.

La performance es incrementada eliminando intermediarios. Las estaciones no necesitan estar "logged in" al servidor de archivos para comunicarse. En realidad, no se necesita la presencia del servidor para que dos estaciones puedan comunicarse directamente mientras el sistema de cableado funcione y el shell IPX.COM esté cargado en las estaciones.

La sobrecarga del servidor es reducida porque el único tiempo en el que está involucrado en una comunicación de estación a estación es cuando actúa como router. El resto del tiempo el servidor queda libre para dedicarse a otras tareas.

3.1 Servicios Orientados a Conexión y Sin Conexión

El servicio orientado a conexión se modeló basándose en el sistema telefónico. Para poder hablar con alguien se debe tomar el teléfono, marcar el número, hablar y colgar. En forma similar, para utilizar una red con **servicio orientado a conexión**, el usuario del servicio establece primero una conexión, la utiliza, y después la termina. El aspecto fundamental de la conexión es que actúa en forma parecida a un tubo: el que envía, introduce objetos por un extremo, y el receptor los recoge, en el mismo orden, por el otro extremo.

A diferencia de esto, el **servicio sin conexión** se modela en base al sistema postal. Cada mensaje (carta) lleva consigo la dirección completa de destino y cada uno de ellos se encamina, en forma independiente, a través del sistema. Normalmente, cuando dos mensajes se envían al mismo destino, el primero que se envíe será el primero en llegar. Es posible, sin embargo, que el primero que se envíe sufra un retardo y llegue antes el que se envió en segundo lugar. Con un servicio orientado a conexión es imposible que suceda esto.

Cada servicio se caracteriza por su calidad; algunos de ellos son confiables en la medida en que nunca pierden la información que transportan. Por lo general, un servicio confiable se logra haciendo que el receptor notifique haber recibido cada mensaje para que el transmisor esté seguro de que su mensaje llegó a destino. El proceso de notificación introduce un exceso de tráfico y retardos, que a menudo son convenientes, pero algunas veces indeseables. Un servicio sin conexión que no es confiable (es decir que no tiene confirmación), se conoce con frecuencia como **servicio datagrama**, por analogía con el servicio de telegramas, el cual tampoco proporciona acuse de recibo de la información al emisor.

3.2 Protocolos Punto a Punto

Novell NetWare ofrece tres protocolos para comunicaciones punto a punto: Internetwork Packet Exchance (IPX), Sequenced Packet Exchance (SPX), y NetBIOS.

3.2.1 Internetwork Packet Exchange (IPX)

IPX es una implementación de Internet Datagram Protocol (IDP) de Network Systems (XNS).

Es el protocolo de más bajo nivel en el ambiente NetWare. Ejecuta servicios que corresponden al **nivel de red** en el Modelo de Referencia OSI. Direcciona y rutea paquetes, pero no ofrece ninguna garantía de entrega. IPX es un elemento crítico en la arquitectura NetWare, define esquemas de direccionamiento entre redes y entre nodos.

El número de red es la base del direccionamiento entre redes. Cada segmento de red sobre una internetwork NetWare debe tener asignado un único número de red. Este número es usado para rutear paquetes expedidos al segmento destino final.

El direccionamiento IPX entre nodos está dado por el número de nodo y el número de socket. El número de nodo identifica a la estación y el número de socket identifica a los proceso que están dentro de ella, así cada proceso puede distinguirse en IPX. En la medida que un proceso necesite comunicarse dentro de la red, requiere que se le asigne un número de socket. Cualquier paquete que IPX recibe, direccionado a este socket es pasado al proceso. Por consiguiente, el número de socket provee un método rápido de ruteo de paquetes dentro de un nodo. Ya que éstos son internos a cada nodo, varias estaciones de trabajo pueden usar el mismo número de socket al mismo tiempo sin confundirse.

La unidad fundamental de operación bajo IPX es el **paquete**, el cual es una entidad lógica. IPX conoce la dirección fuente, la dirección destino y el ruteo de paquetes entre las mismas. No conoce nada acerca de la relación entre los paquetes. Es decir, no le concierne cómo se combinan para formar mensajes. El paquete como pieza aislada de información es llamado **datagrama**.

IPX no tiene conocimiento sobre las conversaciones lógicas entre estaciones que involucran el establecimiento de un enlace permanente (conexiones o sesiones).

IPX es la base sobre la cual se erigen los protocolos de niveles superiores, SPX y NetBIOS. Es también, el medio usado por los servidores especializados para anunciar su presencia en la red. Novell llama a este último Service Advertising Protocol (SAP).

3.2.2 Sequenced Packet Exchange (SPX)

SPX es una implementación de Sequenced Packet Protocol (SPP) de Xerox Network Systems (XNS).

SPX ejecuta servicios que corresponden al **nivel de transporte** en el Modelo de Referencia OSI. Es usado para ofrecer comunicación orientada a conexión. Ofrece los siguientes servicios: provee garantía de entrega de paquetes, entrega de paquetes en secuencia correcta, controla el flujo (evitando que se produzca overflow en los buffers internos de la estación receptora), y suprime paquetes duplicados.

A diferencia de IPX, el cual presta un servicio sin conexión, SPX presta un servicio orientado a conexión. Cuando una conexión se establece entre dos estaciones de trabajo, monitorea el intercambio de paquetes por medio de tres contadores llamados: **número de secuencia, número de confirmación y número de ubicación.**

Número de Secuencia: es el número del último paquete enviado o recibido por la estación de trabajo. Cada dirección de flujo de datos tiene su propio número de secuencia. Cada paquete que envía una estación tiene un número de secuencia de transmisión; éste es confrontado con el número de confirmación de recepción de la estación receptora para asegurar que el paquete arribó en secuencia. El receptor confirma el recibo de un paquete, enviando una contestación que contiene el número de secuencia del paquete que se recibió. El transmisor chequea éste contra su número de confirmación de transmisión para asegurarse que el paquete llegó a destino.

Número de Confirmación: es el número del próximo paquete que la estación espera recibir de su compañera. Para el transmisor, es el último paquete enviado pero no confirmado. Para el receptor, es el número del próximo paquete que la estación espera que envíe el transmisor. Si el paquete arriba con un número de secuencia menor, se asume que es un duplicado y se ignora. Si viene con un número mayor, el receptor puede asumir que algunos paquetes no han arribado, y pide retransmisión.

Número de Ubicación: controla el flujo de paquetes. El número de ubicación de recepción indica el número de secuencia de transmisión más alto que la estación está preparada para recibir. El número de ubicación de transmisión es el número de secuencia de transmisión más alto que la estación puede transmitir. A medida que una estación receptora procesa sus paquetes, incrementa su número de ubicación de recepción. Esto

a su vez permite incrementar el número de ubicación de transmisión del trasnmisor, de manera que se pueden liberar más paquetes para transmitir.

Ejemplo del uso de los contadores de control de SPX

Supongamos que las estaciones A y B han establecido una conexión. A desea enviar alguna información a B. Cada una de ellas tiene suficientes buffers para almacenar tres paquetes. La siguiente tabla muestra el estado de los contadores de cada estación justo después de que la conexión se ha establecido.

Tabla 1. :SPX Números de Control al Inicio de la Conexión

	Estación A		Estación B	
	Tx	**Rx**	**Tx**	**Rx**
Secuencia:	-1	-1	-1	-1
Confirmación:	-1	0	-1	0
Ubicación:	2	2	2	2

La estación A transmite los paquetes número 0, 1 y 2. En este punto, el número de secuencia de transmisión iguala al número de ubicación, así A para de transmitir y espera por la confirmación de B. La siguiente tabla muestra el estado de los contadores después que A ha enviado esos tres paquetes y B los ha recibido.

Tabla 2. :SPX Números de Control después que A Transmite

	Estación A		Estación B	
	Tx	**Rx**	**Tx**	**Rx**
Secuencia:	2	-1	-1	2
Confirmación:	-1	0	-1	3
Ubicación:	2	2	2	2

El número de secuencia de transimisión de A, es ahora 2. El número de secuencia de recepción de B es 2, y su número de confirmación de recepción es 3, porque este es el próximo

paquete que espera recibir de A. El número de confirmación de recepción de A está en 0, porque no ha recibido ninguna confirmación para el paquete 0 desde B. También se debe notar que, por los números de ubicación, A no puede enviar ningún paquete y B no puede recibir ninguno.

Ahora B comienza a confirmar los paquetes que ha recibido desde A. Primero confirma el paquete 0.

Tabla 3. :SPX Contadores después que B Confirma el Paquete 0 de A

	Estación A		Estación B	
	Tx	**Rx**	**Tx**	**Rx**
Secuencia:	2	0	0	2
Confirmación:	0	1	-1	3
Ubicación:	3	3	3	3

A chequea el número de secuencia del paquete que ha recibido de B contra su número de confirmación de recepción, porque éste es el próximo número de paquete que espera. Son iguales, entonces el paquete está en secuencia. A cambia su número de secuencia de recepción a 0, y su número de confirmación de recepción a 1. También modifica su número de confirmación de transmisión para reflejar el hecho que B ha recibido el paquete 0. B, después de procesar este paquete, libera el buffer de entrada, así le dice a A que puede transmitir otro paquete. A coloca su número de ubicación de transmisión a 3, y B hace lo mismo con su número de ubicación de recepción. A ha usado un buffer de entrada, pero lo procesó inmediatamente, así que coloca su número de ubicación de recepción a 3; B puede también colocar su número de ubicación de transmisión a 3 .

Ahora A y B envían un paquete. A envía el paquete con el número de secuencia 3, ya que B le ha dado permiso para que lo haga. B envía la confirmación del paquete número 1 de A.

Del lado de A, un paquete con número de secuencia 1 llegó como confirmado. Este es el próximo paquete que esperaba recibir, así que incrementa sus números de secuencia y de confirmación de recepción. También modifica el número de confirmación de transmisión. B también le dice a A que ha

Tabla 4. :SPX Contadores después que B Confirma el Paquete 1 de A y A Transmite el Paquete 3

	Estación A		Estación B	
	Tx	Rx	Tx	Rx
Secuencia:	3	1	1	3
Confirmación:	1	2	-1	4
Ubicación:	4	4	4	4

liberado un buffer de entrada; A incrementa su número de ubicación de transmisión acorde a esto. Del mismo modo, A ha procesado y liberado un buffer con información recibida, así que incrementa su número de ubicación de recepción a 4. A también setea su número de secuencia a 3.

Del lado de B, el paquete número 3 llegó. Este era el próximo paquete que esperaba recibir, así que modifica sus números de secuencia y confirmación el de recepción. Como B libera un buffer de recepción, confirmando el paquete número 1, incrementa su número de ubicación de recepción a 4. B modifica su número de secuencia de transmisión para reflejar el hecho que reconoce el paquete 1, y modifica su número de ubicación de transmisión, ya que A ha liberado un buffer de entrada.

Asumiremos que llegamos a un estado de equilibrio después que A envía a B el paquete número 3. La siguiente tabla muestra los contadores después que B ha recibido y confirmado todos los paquetes.

Tabla 5. :SPX Contadores después que B Confirma el Paquete 3 de A

	Estación A		Estación B	
	Tx	Rx	Tx	Rx
Secuencia:	3	3	3	3
Confirmación:	3	4	-1	4
Ubicación:	6	6	6	6

Ahora B envía un par de paquetes y arbitrariamente el paquete 4 nunca llega a A.

Tabla 6. :SPX Contadores después que B Transmite los Paquetes 4 y 5

	Estación A		Estación B	
	Tx	**Rx**	**Tx**	**Rx**
Secuencia:	3	3	5	3
Confirmación:	3	4	-1	4
Ubicación:	6	6	6	6

El único contador que ha cambiado es el número de secuencia de transmisión de B. El paquete 4 nunca llega a A. El paquete 5 llega, y A nota que el número de secuencia del paquete no concuerda con el número de confirmación de recepción. A concluye que el paquete que ha arribado está fuera de secuencia, y pide a B que transmita nuevamente comenzando con el paquete número 4. B reenvía el 4 y el 5, A los confirma, y nuevamente se alcanza un estado de equilibrio.

Tabla 7. :SPX Contadores después que B Trasmite Exitosamente los Paquetes 4 y 5

	Estación A		Estación B	
	Tx	**Rx**	**Tx**	**Rx**
Secuencia:	5	5	5	5
Confirmación:	3	6	5	6
Ubicación:	8	8	8	8

Finalmente debemos considerar qué sucede si A envía el paquete 6 y 7. B los recibe y los confirma, pero la confirmación del paquete 6 se pierde. La siguiente tabla muestra los contadores después que A a enviado el paquete 6 y 7 y B los ha recibido.

Tabla 8. :SPX Contadores después que B Recibe los Paquetes 6 y 7

	Estación A		Estación B	
	Tx	Rx	Tx	Rx
Secuencia:	7	5	5	7
Confirmación:	3	6	5	8
Ubicación:	8	8	8	8

El primer paquete que recibe A es la confirmación del paquete 7 de B. A sabe que B no hubiera aceptado el paquete 7 si no recibe antes el 6. De esta manera, A puede asumir que el paquete 6 fue transmitido con éxito, y modifica sus contadores de acuerdo a esto. La siguiente tabla muestra el estado final de los contadores.

Tabla 9. :SPX Contadores después que B Confirma los Paquetes 6 y 7

	Estación A		Estación B	
	Tx	Rx	Tx	Rx
Secuencia:	7	7	7	7
Confirmación:	7	8	5	8
Ubicación:	10	10	10	10

3.2.3 NetBIOS

Es un protocolo desarrollado por IBM. El **Network Basic Input/Output System** funciona prestando un servicio orientado a conexión o sin conexión.

Una aplicación escrita para la interfase NetBIOS puede ser diseñada para usar uno de estos servicios. Por ejemplo, si una aplicación funciona de la forma pregunta/respuesta con un tamaño de transferencia de un paquete, entonces un servicio sin conexión se prodría usar, para aprovechar las ventajas de tiempos de respuestas del servicio. Por otro lado, si las

tranferencias son de un sólo lado o intervienen un gran número de paquetes, se podría usar un servicio orientado a conexión para asegurar la entrega de paquetes y la integridad de los datos. El emulador NetBios de Novell está construido sobre IPX de la mimsa forma que lo está SPX.

El emulador NetBIOS es llamado emulador porque está implementado completamente con software, puesto que el NetBIOS original está ubicado en firmware.

Una característica que NetBIOS tiene para ofrecer en la industria de las redes es su asignación de fácil resolución de direcciones entre estaciones localmente conectadas. Todos los nodos de una red que usa NetBIOS registran un nombre único. Cuando un nodo desea comunicarse con otro, todo lo que necesita conocer es el nombre único que NetBIOS asigna al nodo y éste asegura que el paquete arribe al lugar correcto.

3.3 Las APIs de NetWare

Las APIs (Aplication Program Interfaces) son todas las llamadas a funciones del sistema e interfases que las aplicaciones pueden usar para aumentar sus servicios.

Las APIs de comunicaciones que permiten acceder a los servicios IPX/SPX, son las siguientes:

APIs IPX	APIs SPX
IPXCancelEvent	SPXAbortConnection
IPXCloseSocket	SPXEstablishConnection
IPXDisconnectFromTarget	SPXGetConnectionStatus
IPXGetInternetworkAddress	SPXInitialize
IPXGetMarker	SPXListenForConnection
IPXGetLocalTarget	SPXListenForSequencedPacket
IPXInitalize	SPXSendSequencedPacket
IPXOpenSocket	SPXTerminateConnection
IPXRelinquishControl	
IPXScheduleIPXEvent	
IPXSendPacket	

Serán tratadas en detalle en la sección 3.12 APIs del Servicio de Comunicación.

3.4 Comunicación Sin Conexión bajo IPX

Como ya se mencionó anteriormente IPX no conoce nada acerca del intercambio de información entre dos estaciones; todo lo que sabe es entregar paquetes.

La secuencia de pasos involucrados en un intercambio de información usando IPX es la siguiente:

1- Llamar a **IPXInitalize()** para inicializar el puntero a las rutinas IPX. Esto es obligatorio. No hay manera de testear si IPX está instalado, pero no es perjudicial llamar en forma redundante a la rutina.

2- Llamar a **IPXOpenSocket()** para abrir el socket que se usará para comunicaciones.

3- Colocar solicitudes a escuchar sobre el socket con **IPXListenForPacket()**.IPX necesita varios ECBs (EventControlBlocks) escuchando, de 5 a 20 son usualmente suficientes. De otro modo, IPX puede descartar paquetes porque no tiene buffers para recibirlos.

Los ECBs son el enlace entre la aplicación e IPX; cada uno ellos contiene la información que IPX necesita para mandar o recibir un paquete.

4- Usar **IPXSendPacket()** e **IPXListenForPacket()** para intercambiar datos con la estación de trabajo destino.

5- Cuando se finaliza, llamar a **IPXCloseSocket()** para cerrar el socket.

Los sockets pueden ser de corta vida o de larga vida. Los de larga vida quedan abiertos después que la aplicación termina y son ventajosos para programas que quedan residentes en memoria.

Es especialmente importante que la aplicación cierre este tipo de sockets. Si el socket fue abierto como de corta vida, IPX cierra el socket cuando el programa termina.

3.5 Comunicación Orientada a Conexión bajo SPX

Comunicarse bajo SPX involucra más preparación que la requerida para comunicarse bajo IPX. Los pasos involucrados en intercambios de información usando SPX son:

1- Llamar a **IPXInitialize()**.

2- Llamar a **SPXInitialize()** para averiguar si SPX es soportado por la estación de trabajo.

3- Asumiendo que es soportado, llamar a **IPXOpenSocket()** para abrir el socket.

4- Colocar algunos ECBs a escuchar usando **SPXListenForSequenced- Packet()**. SPX requiere que al menos dos de estos buffers se usen para escuchar un pedido de conexión de otra estación. El número de ECBs escuchando determina cuántos paquetes de confirmación puede transmitir el compañero; esto es lo que SPX usa para regular el flujo de datos sobre la conexión. Un gran número de ECBs escuchando mejora la eficiencia de las comunicaciones.

5- Si se desea establecer una conexión, llamar a **SPXEstablishConnection()**. Si se está escuchando por una conexión, llamar a **SPXListenForConnection()**.

6- Una vez que la conexión es establecida usar **SPXSendSequencedPacket()** y **SPXListenForSequencedPacket()** para intercambiar datos. Asegurarse que SPX tenga un buen suministro de ECBs escuchando y de reciclar éstos cuando sus eventos se completen.

7- Cuando finaliza la comunicación, llamar a **SPXTerminateConnection()** para cerrar la conexión, luego a **IPXCloseSocket()** para cerrar el socket.

3.6 Diferencias, Ventajas y Desventajas entre IPX y SPX

La diferencia entre un protocolo orientado a paquete como IPX y un protocolo orientado a conexión como SPX es semenjante a la diferencia que existe entre enviar una carta y hablar con alguien por teléfono. Cuando se envía una carta por correo, se tiene la certeza razonable que se recibirá donde se envió, pero no estamos seguros hasta obtener respuesta. Con una conversación telefónica, por otro lado, un teléfono establece una conexión física con otro telefóno entonces ambos lados establecen una conexión lógica, cuando el que recibe la llamada contesta y el llamador se identifica. Sólo en este punto comienza el intercambio de información. Durante la conversación, se sabe que la información que se envía llega a destino porque del otro lado contestan apropiadamente. Al finalizar, ambos acuerdan terminar la conexión, y cuelgan.

Como ya se mencionó IPX trata con datagramas, es decir, paquetes de datos que no requieren de confirmación y secuenciamiento. Los datagramas reducen el tráfico de la red. En IPX el porcentaje típico de éxito en la entrega de datagramas es del 95%. Consecuentemente, si se usa IPX se necesita desarrollar una estrategia para confirmar entregas, y posiblemente, guardar los paquetes en orden. Si se transmiten mensajes cortos que se adaptan dentro de un solo paquete y no requieren de confirmación, se prodría usar IPX tal cual es. Si se usa IPX para ejecutar transacciones simples de pedido/respuesta, la respuesta puede servir como confirmación. Si no se recibe confirmación dentro de un cierto período de tiempo, se puede asumir que la entrega ha fallado y reaccionar de acuerdo a esto.

Si se necesita conocer con certeza que los datos están siendo recibidos en el orden en que se enviaron, se debe construir la aplicación de tal manera que lo garantice o usar SPX. SPX provee el secuenciamiento de paquetes, garantiza la entrega, la detección y corrección de errores, asociado con un servicio orientado a conexión. Estas características suman algo de overhead a las

comunicaciones de la red, ya que dos o más paquetes se envían por cada paquete de información. Además, no permite que un paquete se envíe a más de una estación (comunicaciones broadcast), la conexión debe ser establecida con cada receptor y los mensajes no pueden enviarse si el receptor no está disponible.

3.7 Estructura del Paquete IPX

Un paquete IPX consta de una cabecera de 30 bytes y una porción de datos que puede ir de 0 a 546 bytes. Por lo tanto la longitud mínima de un paquete es de 30 bytes (cabecera más datos de longitud 0) y el máximo es de 576 bytes (cabecera más datos de longitud 546).

La cabecera está compuesta por los siguientes campos:

Cabecera del Paquete de IPX

Offset	Contenido	Tipo
0	Checksum	WORD
2	Longitud	WORD
4	Control de Transporte	BYTE
5	Tipo de Paquete	BYTE
6	Red Destino	BYTE[4]
10	Nodo Destino	BYTE[6]
16	Socket Destino	BYTE[2]
18	Red Origen	BYTE[4]
22	Nodo Origen	BYTE[6]
28	Socket Origen	BYTE[2]
30	Datos	BYTE[0 a 546]

Para enviar un paquete se deben inicializar determinados campos. El campo **Tipo de Paquete** y los campos de **dirección de Destino (Red, Nodo y Socket)** deben ser inicializados por el programador.

Todos los campos, excepto los Datos y aquellos que tienen sólo un byte, son high-low.

Checksum

Es usado por compatibilidad con la cabecera del paquete original de Xerox. Este campo es seteado siempre a 0xFFFF por IPX; las tarjetas LAN calculan el checksum de un paquete completo de IPX por hadware, haciendo que el campo sea innecesario.

Longitud

Contiene la longitud del paquete completo IPX (longitud de la cabecera -30- + datos -0 a 546-). Este campo es seteado por IPX.

Tipo de Paquete

Indica el tipo de servicio ofrecido o requerido por el paquete. Xerox ha definido los siguientes valores:

0	Tipo de paquete desconocido
1	Paquete de información de ruteo
2	Paquete echo
3	Paquete de error
4	Paquete de intercambio
5	Paquete de protocolo con secuencia
16-31	Protocolos experimentales
17	Protocolo de NetWare

Los usuarios de IPX deberían setear el Tipo de Paquete a 0 o 4, los usuarios de SPX a 5.

Red Destino

Contiene el número de red en el cual el nodo destino reside. Este número es un valor de cuatro-bytes dado por el servidor a las estaciones que pertenecen al mismo segmento de red. Cuando este campo es 0, Novell asume que el nodo destino está sobre la misma red física que el nodo origen, por lo tanto, el paquete no es enviado a través de un puente.

Nodo Destino

Es un campo de seis-bytes que especifica la dirección física de el nodo destino. El número de bytes depende de el hadware de la LAN usada. Las tarjetas Ethernet usan seis bytes. Si una dirección ocupa menos la dirección debería completarse con ceros. Una dirección de nodo seteada con todos los bytes en FFh hace que IPX se comporte en modo broadcast, envía el paquete a todos los nodos de la red destino.

Socket Destino

Es la dirección del proceso del paquete destino. Debido a que en cada nodo pueden existir varios procesos, es necesario distinguirlos entre sí. El socket permite que se entregue el paquete a un determinado proceso del nodo. Xerox a reservado los siguientes números de Socket:

1	Paquete de información de ruteo
2	Paquete de echo del protocolo
3	Paquete de manejo de errores
20h-3Fh	Experimental
1h-BB8h	Registrado con Xerox
BB9h	Asignable dinámicamente

Xerox ha asignado a Novell los siguientes sockets para red:

451h	Paquete de servicio de archivos
452h	Paquete de servicio de advertencia
453h	Paquete de ruteo de información
455h	Paquete de NetBIOS
456h	Paquete de diagnóstico

Los programadores pueden asignar a sus programas el número de socket en forma permanente. Los números dinámicos son asignados por el shell al azar, comenzando con 4000h. Los números de socket asignados por Novell comienzan en 8000h. Los números de socket mayores a 8000h deberían ser evitados si desean usarse en modo permanente, a menos que ellos hayan sido registrados con Novell.

Red Origen

Es el número de red de la estación que envía el paquete IPX. IPX le da a éste campo el número de red de la estación que está haciendo el requerimiento; si es cero, la red física es desconocida.

Nodo Origen

Es seteado por IPX con la dirección física del nodo origen.

Socket Origen

Es la dirección del sockek del proceso que envía el paquete, y es el socket especificado en el ECB. Es seteado por IPX. Al igual que el socket destino, puede ser estático o dinámico, y sigue las mismas convenciones que el primero.

Los procesos de comunicación no necesitan enviar y recibir paquetes sobre el mismo número de socket. En un ambiente cliente-servidor, el nodo server usualmente escucha sobre un específico socket los paquetes que entran con pedidos. En este caso, el servidor debería responder a el socket origen.

3.8 Estructura del Paquete SPX

Un paquete SPX es idéntico a un paquete IPX pero tiene 12 bytes más en su cabecera; y como la cabecera se incrementa de 30 a 42 bytes, el máximo es de 576 bytes, así la porción de datos es decrementada a 534 bytes.

Excepto por el **tipo de paquete**, el cual es siempre 5 para SPX, y por el **nodo destino**, el que no puede ser broadcast (seis bytes de FFh) en SPX, los campos comunes a IPX y SPX son los mismos en las cabeceras de los protocolos.

Nuevamente, todos los campos (con más de un byte) son high-low.

Control de Conexión

Contiene cuatro bits de flags usados por SPX para controlar el flujo bidireccional de datos a través de la conexión:

- 1-8 (Indefinido) Ignorados por SPX.
- 10h (Fin de Mensaje) Un cliente setea este flag para avisar el fin de conexión. SPX lo ignora.
- 20h (Atención) Un cliente setea este flag si el paquete es atención. Este aspecto no ha sido implementado, así que SPX ignora este bit, también.
- 40h (Confirmación) SPX setea este bit si un paquete de confirmación es necesario. Debido a que SPX maneja confirmaciones de los paquetes que son enviadados y los paquetes que son recibidos, las aplicaciones no necesitan saber acerca de ellos.

Cabecera del Paquete de SPX

Offset	Contenido	Tipo
0	Checksum	WORD
2	Longitud	WORD
4	Control de Transporte	BYTE
5	Tipo de Paquete	BYTE
6	Red Destino	BYTE[4]
10	Nodo Destino	BYTE[6]
16	Socket Destino	BYTE[2]
18	Red Origen	BYTE[4]
22	Nodo Origen	BYTE[6]
28	Socket Origen	BYTE[2]
30	Control de Conexión	BYTE
31	Tipo de Datos	BYTE
32	ID de Conexión Origen	WORD
34	ID de Conexión Destino	WORD
36	Número de Secuencia	WORD
38	Número de Confirmación	WORD
40	Número de Ubicación	WORD
30	Datos	BYTE[0 a 534]

80h (Paquete del Sistema) SPX setea este bit si el paquete es un paquete del sistema. Tales paquetes no son entregados a las aplicaciones.

Las aplicaciones no deberían usar o modificar éste campo para uso propio, pues está reservado para SPX.

Tipo de Datos

Es un flag de un byte que indica el tipo de dato del paquete. Los posibles valores y definiciones para el campo son:

0-FDh (Definido por el Cliente) Este campo es definido por la aplicación y es ignorado por SPX.

FEh (Fin de Conexión) SPX genera este paquete cuando el cliente llama para terminar una conexión activa; éste es el último mensaje entregado sobre la conexión.

FFh (Confirmación de Fin de Conexión) SPX genera este paquete del sistema; el mismo no es entregado a los clientes conectados.

ID Conexión Origen

Es un número asignado por SPX a los paquetes del nodo origen.

ID Conexión Destino

Es un número asignado por SPX a los paquetes del nodo destino. Debido a que algunas conexiones activas al mismo tiempo sobre una estación pueden usar el mismo socket, este campo es usado para multiplexar los paquetes que entran de diferentes conexiones abiertas sobre el mismo socket.

Número de Secuencia

Tiene un contador de los paquetes enviados sobre una dirección de la conexión. Cada parte guarda su propio contador. El número puede oscilar entre 0000h y FFFFh. SPX maneja este campo; las aplicaciones no necesitan saber acerca de él.

Número de Confirmación

Es el número de secuencia del próximo paquete que espera recibir SPX. Si se recibe un paquete con un número de secuencia menor a este, el paquete es duplicado, por lo tanto no necesita retransmitirse. SPX maneja este campo; las aplicaciones no deben preocuparse por él.

Número de Ubicación

Es el número de buffers en escucha de paquetes que llegan en una dirección sobre la conexión. SPX puede solamente enviar paquetes hasta que el número de secuencia iguale a el número de ubicación remoto. SPX maneja este campo; las aplicaciones no deben preocuparse por él.

3.9 Bloque de Control de Eventos (ECBs)

Dos estructuras son necesarias para enviar o recibir paquetes bajo IPX o SPX (la cabecera de IPX-SPX, explicado anteriormente, y un ECB).

Un ECB es una estructura que literalmente controla el "evento", ya sea un envío o una recepción. No existe una diferencia estructural entre un Enviar ECB y un Recibir ECB, la diferencia está dada en el contenido; un Enviar ECB necesita una dirección destino en su campo de **Dirección Inmediata** y el Recibir ECB no.

Estructura ECB

Offset	Contenido	Tipo	Orden
0	Dirección de Enlace	LONG	seg-offset
4	Dirección ESR	LONG	seg-offset
8	Flag "En Uso"	BYTE	
9	Código de Completitud	BYTE	
10	Número de Socket	WORD	high-low
12	Area de Trabajo de IPX	LONG	
16	Area de Trabajo del Driver	BYTE[12]	
28	Dirección Inmediata	BYTE[6]	high-low
34	Contador de Fragmento	WORD	high-low
36	Dirección de Fragmento 1	LONG	seg-offset
40	Tamaño de Fragmento 1	WORD	
42	Dirección de Fragmento 2	LONG	seg-offset
46	Tamaño de Fragmento 2	WORD	

Para enviar un paquete los siguientes campos deben ser inicializados por el programador: **Dirección ECB, Número de Socket, Dirección Inmediata** y al menos un **Fragmento.**

Para recibir un paquete, los siguientes campos deben ser inicializados por el programador: **Dirección ECB, Número de Socket** y al menos un **Fragmento.**

Dirección de Enlace

Mientras el ECB está en uso este campo es utilizado por IPX; mientras el ECB no está en uso, está disponible para la aplicación. Las aplicaciones generalmente lo usan como un campo de enlace para escuchar ECBs libres.

Dirección ECB

Contiene la dirección de una rutina de servicio de evento (ESR) definada por la aplicación. La misma es llamada cuando IPX envía o recibe paquetes. Si el ESR no es usado, este campo debe ser un puntero nulo (cuatro bytes o cero).

Flag "En Uso"

Este campo es administrado por IPX. Es seteado a cero cuando IPX terminó de usar el ECB. Cuando el ECB está en uso los posibles valores son:

FBh	Un envío o una recepción ha ocurrido, pero el ECB está en una cola esperando ser procesado.
FDh	Un evento está siendo registrado e IPX aguarda la expiración de su intervalo de tiempo.
FEh	IPX está escuchando sobre un socket paquetes entrantes.
FFh	El ECB está en uso para enviar un paquete.
FAh	El ECB está siendo procesado por IPX.
F8h	Un envío fue intentado mientras IPX estaba ocupado, por lo tanto el paquete enviado y el ECB han sido ingresados en una cola para procesarlos más tarde.

Código de Completitud

Es seteado por IPX para mostrar el status final de un ECB; el campo no es válido hasta que IPX ponga el Flag En Uso en cero. Algunos de los siguientes códigos pueden ser retornados por un **Enviar ECB:**

00	(Exitoso) El paquete fue enviado.
FCh	(Cancelado) El envío fue cancelado.

FDh (Malformado) El paquete está malformado. La longitud total del paquete es menor que 30 bytes (IPX) o 42 bytes (SPX); o la longitud del paquete es mayor que 576 bytes; o el primer fragmento es demasiado pequeño para la cabecera; o el Contador de Fragmento es cero.

FEh (No entregado) El paquete no pudo ser entregado. El nodo destino no estaba listo para recibir el paquete.

FFh (Falla de Hardware/Network) La red no puede enviar el paquete por falla del hardware o network.

Cuando un ECB está sometido a IPX para **escuchar paquetes**, alguno de los siguiente códigos pueden ser informados:

00 (Exitoso) El paquete fue recibido con éxito.

FCh (Cancelado) El pedido de escuchar ha sido cancelado.

FDh (Overflow) Un paquete overflow ha sido recibido. Un paquete ha sido recibido, pero el Contador de Fragmento en el ECB está en cero, o el espacio disponible (del Descriptor del Fragmento) es inadecuado.

FFh (Cerrado) El socket está cerrado.

Cuando un ECB está bajo IPX para **cancelar**, los siguiente códigos pueden ser informados:

00 (Exitoso) El evento fue cancelado.

F9h (Falla de Cancelación) El evento no pudo ser cancelado.

Número de Socket

Es el número de socket asociado con un ECB. Para un Enviar ECB, es el Número de Socket del cual el paquete es enviado; para un Recibir ECB, es el Número de Socket sobre el cual el paquete es recibido.

Area de Trabajo de IPX

Es un campo de cuatro bytes reservado para IPX. Este no necesita ser inicializado y no debe ser cambiado mientras IPX está usando el ECB. Cuando no está siendo usado por IPX, una aplicación puede utilizarlo.

Area de Trabajo del Driver

Es un campo de doce bytes reservado para el driver de red. Nuevamente, éste no necesita ser inicializado y no debe ser cambiado mientras IPX está usando el ECB. Las aplicaciones pueden usarlo en otro momento.

Dirección Inmediata

Es un campo de seis bytes. Contiene la dirección del nodo al cual el paquete es enviado o desde el cual arriba; o la dirección de un puente sobre la red local si el paquete no es enviado o recibido desde un nodo de la red local.

En IPX, una aplicación debe inicializar este campo antes que el ECB sea enviado. En SPX, el campo es inicializado durante la creación de una conexión.

Contador de Fragmento

Es el número de buffers que componen un paquete; debe ser mayor que cero.

Descriptor de Fragmento

Contiene la dirección y el tamaño del buffer de un paquete. Siempre un ECB de IPX debe tener al menos un Descriptor de Fragmento. Puede tener un número arbitrario de descriptores adicionales, éstos últimos forman una lista de Descriptores de Fragmentos.

Los paquetes pueden ser enviados o recibidos con un sólo buffer, poniendo el Contador de Fragmento en uno y describiento un buffer de tamaño adecuado. El primer descriptor de la lista debe ser la cabecera de los paquetes, tendrá una longitud de al menos 30 bytes para IPX y 42 bytes para SPX y deben completarse los campos que correspondan.

Lista de Descriptores de Fragmentos

Offset	Contenido	Tipo	Orden
0	Dirección de Enlace	LONG	seg-offset
4	Tamaño	WORD	high-low

3.10 Administrador de Eventos Asincrónicos (AES)

El administrador de eventos asincrónicos permite que eventos de IPX sean ejecutados en un determinado momento. Las rutinas AES pueden enviar paquetes en intervalos regulares de tiempo. Por ejemplo, se usan AES cuando el server debe enviar paquetes de identificación en forma broadcast periódicamente.

Otro ejemplo de uso de AES es el proceso de watchdog ("perro guardian") de NetWare. Si un server no recibe paquetes de alguna estación existente en su tabla de conexión durante un lapso de tiempo de más de cinco minutos, éste envía un paquete watchdog a esa estación. Si la misma no responde, el server desconecta la estación y resetea la conexión.

3.11 Rutinas de Servicio de Eventos (ESR)

Una rutina de servicio de eventos es una rutina programada para ser llamada después que un evento particular ocurra. Un evento puede ser el pedido de envío de un paquete o el poner un paquete a escuchar, realizado en forma exitosa. Se puede tratar de un evento de IPX o un evento de propósito especial definido por la aplicación.

Un ECB contiene un campo que apunta a un ESR, **Dirección ESR.**

ESRs son llamadas por los ECBs al concluir un evento, cuando se cumplen las siguientes condiciones: el campo **Flag En Uso** del ECB tiene valor cero, el campo **Código de Completitud** ha sido seteado, el ECB contiene todo la información apropiada sobre el evento, y el ECB ha sido removido de la lista de IPX.

Un ESR es una rutina del servicio de interrupción. Requiere ser llamada con las interrupciones del procesador deshabilitadas. La misma debe ser diseñada cuidadosamente y ser tratada como una interrupción del servicio. Debe ejecutar tan pocas instrucciones como sea posible y no debe manejar nada que se pudiera hacer en el cuerpo de la aplicación.

Un camino para manejar múltiples ECBs es creando una cola de servicios de ECBs y usando ESRs para notificar que han sido recibidos ECBs, y fueron puestos en la cola.

3.11.1 Interfase de Rutinas de Servicio de Eventos

Cuando ESRs son invocados:

AL — Contiene el identificador del proceso llamado:
00h si AES llama al ESR
FFh si IPX llama al ESR

ES:SI — Contiene un puntero al ECB asociado.

Antes de llamar a una rutina del servicio de eventos, el shell guarda todos los flags y registros, en el stack, excepto el SS y SP. Los programadores deberían reservar su propio stack. El ESR es responsable de guardar el SS y SP para poder retornar apropiadamente después de su ejecución. El shell también deshabilita las interrupciones antes de hacer la llamada al ESR.

Cuando el shell llama al ESR, pone el Flag En Uso del ECB a cero, registra toda la información apropiada perteneciente al evento en el ECB, y remueve el ECB de la lista interna de IPX o AES. Es en este momento cuando el ESR está en condiciones de trabajar con el ECB.

Cuando el ESR comienza, la rutina debe inicializar el registro DS con las variables de la aplicación que pueden ser referenciadas.

Un ESR retorna sin devolver ningún valor o flag y con las interrupciones deshabilitadas.

Si bien un programador está libre para habilitar interrupciones en un ESR, éstas deberían ser deshabilitadas nuevamente antes de retornar. Este proceso se debe hacer cuidadosamente debido a que el ESR debería ser diseñado como una rutina re-entrante, pues éste puede ser llamado nuevamente mientras una interrupción esté habilitada. También debería ejecurtarse en poco tiempo, de lo contrario su ejecución se puede tornar peligrosa, los eventos de enviar y recibir podrían ser pospuestos indefinidamente.

3.12 APIs del Servicio de Comunicación

Las llamadas a IPX (00h a 0Bh) están incluidas en las versiones de Advanced NetWare 1.02, 2.0 y 2.1, en NetWare 386 y en las últimas versiones. Las llamadas a SPX (10h a 17h) están incluidas en Advanced NetWare 2.1, en NetWare 386 y en las últimas versiones. En esta sección explicaremos cada una de ellas.

3.12.1 APIs del Protocolo IPX

IPX Inicialize (FFh)

Retorna la dirección de las rutinas del servicio de interrupción de IPX.

Registros de Entrada

AL	FFh

Registros de Salida

AL	00h si IPX está instalado
ES:DI	Dirección de la rutina

La interfase en C de NetWare es :

```
BYTE IPXInicialize (void)
```

IPX Open Socket (00h)

Abre un socket IPX.

Registros de Entrada

BX	00h
AL	Flag de vida del socket
DX	Número de socket pedido (hi-lo)

Registros de Salida

AL	Código de Completitud
DX	Número de socket asignado

Código de Completitud

00h	Exitoso
FFh	Socket ya está abierto
FEh	Tabla de socket llena

Una aplicación debe abrir el socket antes de enviar o recibir paquetes por el mismo.

El **Flag de Vida del Socket** especifica cuánto tiempo el socket permanecerá abierto. Si este campo toma el valor 00h significa que el socket permanecerá abierto hasta que sea cerrado con una llamada al IPX Close Socket o hasta que la aplicación termine. Un valor de FFh indica que el socket permanecerá abierto hasta que sea cerrado exitosamente con un IPX Close Socket. Aplicaciones TSR (programas que terminan y permanecen residentes) deberían usar FFh para dejar el socket abierto, otros deberían usarlo en 00h para asegurarse que el socket se cierre cuando termina la aplicación.

El **Número de Socket Pedido** es el número de socket que se desea abrir. Si se asigna el valor 0000h a este campo, IPX dará al socket un número dinámicamente.

Por defecto, una estación puede tener 20 sockets abiertos al mismo tiempo, pero se puede incrementar a 150 a través del archivo de configuración SHELL.CFG.

La interfase en C de NetWare es :

```
int IPXOpenSocket (BYTE *socketNumber, BYTE
socketTYPE)
```

IPX Close Socket (01h)

Cierra un socket IPX.

Registros de Entrada

BX	01h
DX	Número de Socket (hi-lo)

Registros de Salida

Ninguno

El **Número de Socket** especifica el socket a ser cerrado.

Código de Completitud
Ninguno

Cuando el socket es cerrado, IPX cancela todos los eventos definidos por los ECBs asociados con el socket (tales como recibir). También, IPX retorna FCh como **Código de Completitud** en el ECB, indicando que fueron cancelados. Por último, IPX pone en cada ECB el campo **Flag En Uso** en 00h (disponible) y no llama a ESRs asociados.

TSRs debería cerrar el socket antes de terminar.

Estos llamados no pueden ser invocados desde rutinas de servicio de eventos (ESRs).

La interfase en C de NetWare es :
void IPXCloseSocket (WORD socketNumber)

IPX Get Local Target (02h)

Retorna la dirección del nodo fuente de seis bytes.

Registros de Entrada

BX	02h
ES:SI	Dirección de buffer de pedido
ES:DI	Dirección de buffer de respuesta

Registros de Salida

AL	Código de Completitud
CX	Tiempo de Transporte

Código de Completitud

00h	Exitoso
FAh	No se encontró camino al nodo destino

Buffer de Pedido (12 bytes)

Offset	Contenido	Tipo	Orden
0	Red Destino	BYTE[4]	high-low
4	Nodo Destino	BYTE[6]	high-low
10	Socket Destino	BYTE[2]	high-low

Buffer de Respuesta (6 bytes)

Offset	Contenido	Tipo	Orden
0	Nodo Fuente	BYTE[6]	high-low

Los programadores envían una dirección completa de red a el server de archivos compuesta por las direcciones de **Red, Nodo, y Socket Destino.** El server retorna la dirección del puente local más cercano y la cantidad de tiempo estimado para entregar el paquete al destino.

Tiempo de Transporte es el tiempo estimado (en ticks de reloj de aproximadamente 1/18 segundos) requerido para entregar 576-bytes al destino.

Esta llamada debería ser usada para dar valor al campo **Dirección Inmediata** del ECB. Antes de llamar a Enviar Paquete, una aplicación debería llamar primero a esta función y copiar el **Nodo Origen** en el campo **Dirección Inmediata** del ECB. Sin embargo, si la aplicación está respondiendo a un paquete recientemente enviado, el ECB que escucha, contendrá la **Dirección Inmediata** para enviar un paquete de regreso.

Esta función puede ser llamada desde una rutina del servicio de eventos de IPX, una AES, o directamente desde una aplicación (pero no puede ser llamada desde alguna interrupción del servicio).

La interfase en C de NetWare es :

```
int GetLocalTarget (BYTE *networkAddress,
                    BYTE *immediateAddress,
                    int  transportTime)
```

IPX Send Packet (03h)

Inicia la transmisión de un paquete IPX.

Registros de Entrada

BX	03h
ES:SI	Dirección ECB

Registros de Salida

Ninguno

Código de Completitud
Ninguno

Este llamado pasa un ECB a IPX para enviar un paquete. El llamado retorna inmediatamente mientras IPX intenta enviarlo.

Para hacer esta llamada, la aplicación debe inicializar los siguientes campos del ECB: **Dirección ESR, Número de Socket, Dirección Inmediata, Contador de Fragmento, Descriptor de Fragmento.** La aplicación debe también inicializar los campos de la cabecera del paquete IPX, **Tipo de Paquete, Red Destino, Nodo Destino** y **Socket Destino.**

La Dirección ECB apunta a un Bloque de Control de Eventos IPX.

En principio, IPX dará el valor FFh al campo **Flag En Uso** del ECB para indicar que está enviando un paquete. IPX también inicializará los campos de la cabecera: **Checksum, Longitud, Control de Transporte, Red Origen, Nodo Origen** y **Socket Origen.**

Después de hacer este llamado, el programador debería llamar a IPX Relinquish Control iterativamente para esperar el envío del paquete. Después que el paquete ha sido transmitido, el programador debería chequear el campo **Código de Completitud** del ECB.

Después de enviar el paquete, el campo Flag En Uso será puesto en 0 (disponible) y el ESR de envío será llamado, si existiera.

Los posibles valores del **Código de Completitud** son:

00h	Exitoso, enviado, no necesariamente recibido.
FCh	La aplicación caceló el pedido de transmisión.
FDh	El paquete fue malformado. El tamaño es menor que 30 bytes o mayor que 576 bytes, el primer fragmento es menor que 30 bytes, o el valor en el Contador de Fragmento es 00h.
FEh	El paquete fue perdido.
FFh	IPX está deshabilitado para transmitir paquetes debido a fallas de hadware o de red.

Debido a que éste es un protocolo de datagramas, el 00h en el Código de Completitud indica que el paquete fue enviado en forma exitosa, pero no garantiza que el paquete haya sido bien recibido por el nodo destino. Las posibles razones para que se produzca la pérdida de paquetes pueden ser: el medio de transmisión puede perder o duplicar paquetes, o el socket destino puede no estar abierto o no escuchando la llegada de los mismos.

El valor FEh en el Código de Completitud indica que el paquete no fue entregado. Esto puede producirse porque: IPX puede no hallar el puente que permita llegar a la red destino, la dirección del nodo fuente no existe, o el socket destino no fue abierto o no tiene ECB escuchando.

Una estación puede transmitir paquetes IPX a cualquier estación, incluyéndose a sí misma.

La interfase en C de NetWare es :

```
void IPXSendPacket (ECB *evenControlBlock)
```

IPX Listen For Packet (04h)

Prepara a IPX para recibir un paquete.

Registros de Entrada

BX	04h
ES:SI	Dirección ECB

Registros de Salida

AL	Código de Completitud

Código de Completitud

FFh	No existe un socket escuchando

Este llamado pasa un ECB a IPX para comenzar a escuchar la llegada de paquetes. El llamado retorna inmediatamente mientras IPX intenta escuchar paquetes.

Antes de hacer este llamado, el programador debe abrir el socket e inicializar los campos del ECB: **Dirección ESR, Número de Socket, Contador de Fragmento, Descriptor de Fragmento.**

Cuando este llamado es hecho, IPX setea el campo **Flag En Uso** a FEh indicando que el ECB está esperando para recibir un paquete IPX, también suma el ECB a un pool de buffers de ECBs, todos los cuales están escuchando la llegada de paquetes sobre un mismo socket. IPX no tiene límite sobre el número de ECBs que pueden estar escuchando concurrentemente sobre un socket.

Existen dos caminos para notificar la llegada de paquetes: a través de un polling o a través de interrupciones.

El primer método consiste en chequear continuamente el valor de el campo Flag En Uso; si éste cambia a 0, un paquete ha sido recibido. También se debe verificar el Código de Completitud para asegurarse que ha sido recibido correctamente.

El segundo método puede utilizar ESRs como interrupciones.

Cuando un paquete es recibido, IPX emplaza el paquete en un ECB que está escuchando (ellos no son ocupados en un orden particular), se asigna valor al **Código de Completitud** y el campo **Dirección Inmediata** toma el valor de la dirección del nodo. De esta forma, cuando el paquete es retornado, la **Dirección Inmediata** ya está seteada. Finalmente, IPX setea el campo **Flag En Uso** a 0, disponible, y llama al ESR, si existiera.

Los posibles valores del **Código de Completitud** son los siguientes:

00h	Exitoso, el paquete ha sido recibido.
FCh	La aplicación caceló el pedido de escuchar.
FDh	El paquete fue recibido, pero el valor en el Contador de Fragmento es 00h, o el espacio en el buffer para el Descriptor del Fragmento es demasiado pequeño para contenerlo.
FFh	El socket definido por el ECB no está abierto.

La interfase en C de NetWare es :

```
void IPXListenForPacket (ECB *evenControlBlock)
```

IPX Schedule IPX Event (05h)

Administra un evento IPX.

Registros de Entrada

BX	05h
AX	Tiempo Diferido
ES:SI	Dirección ECB

Registros de Salida

Ninguno

Código de Completitud

Ninguno

Este llamado difiere un ECB durante un número de ticks (1/18 segundos) dado en el campo **Tiempo Diferido.**

Antes de hacer el llamado, el programador debe inicializar el campo **Dirección ESR** del ECBs y el **Número de Socket.** IPX pone el campo **Flag En Uso** en FDh para indicar que el evento está contando tiempo de demora. En el último tick del reloj, IPX inicia el evento, setea el **Flag En Uso** a 00h, y llama a la rutina del servicio de eventos.

Algunas aplicaciones o un ESR puede invocar a esta rutina. Una aplicación nunca debería usar este llamado para pasar la dirección del ECB que está siendo utilizado por IPX.

La interfase en C de NetWare es :

```
void IPXScheduleIPXEvent (WORD timeUnits,
                          ECB  *evenControlBlock)
```

IPX Cancel Event (06h)

Cancela un evento definido por un ECB.

Registros de Entrada

BX	06h
ES:SI	Dirección ECB

Registros de Salida

AL	Código de Completitud

Código de Completitud

00h	Exitoso

F9h ECB no puede ser cancelado
FFh ECB no está en uso

Los eventos que puede cancelar son enviar, escuchar, adminitrar o eventos de propósito especial.

Cuando este llamado es hecho, IPX setea el campo **Flag En Uso** del ECB a un valor distinto de 00h, indicando que el ECB está disponible para otras aplicaciones. Después de intentar cancelar el evento definido por el ECB, IPX setea el **Código de Completitud** al valor apropiado. Este también setea el campo **Flag En Uso** a 00h (disponible para uso). IPX no llama a un ESR.

Los posibles valores del **Código de Completitud** son:

FCh Evento Cancelado

Este llamado no cancela paquetes que ya han sido transmitidos.

La interfase en C de NetWare es :

```
int IPXCancelEvent (ECB  *evenControlBlock)
```

IPX Get Interval Marker (08h)

Marca el tiempo de IPX.

Registros de Entrada
BX 08h

Registros de Salida
AX Marca de Intervalo

Código de Completitud
Ninguno

Marca de Intervalo es un valor entre 0 y 65535 (0000h y FFFFh), representando un tick de reloj (1/18 segundos).

Una aplicación puede usar este llamado para testear la cantidad de tiempo entre eventos. Esta función es llama una vez, ejecuta el evento, y la llama nuevamente, y hace la resta entre el segundo y el primero.

La interfase en C de NetWare es :

```
void IPXGetIntervalMarker (void)
```

IPX Get Internetwork Address (09h)

Retorna la dirección de red y el nodo de la estación pedida.

Registros de Entrada

BX 09h

Registros de Salida

ES:SI Dirección de buffer de respuesta

Código de Completitud

Ninguno

Buffer de Respuesta (10 bytes)

Offset	Contenido	Tipo	Orden
0	Dirección de Red	BYTE[4]	high-low
0	Dirección de Nodo	BYTE[6]	high-low

Este llamado puede ser usado por las aplicaciones que necesitan información acerca de la dirección completa de red de otras estaciones. Los 2-bytes del socket deben ser adicionados al final de la dirección completa de red, 12-bytes, porque pueden existir varios socket abiertos sobre una estación.

La interfase en C de NetWare es :

```
void GetInternetworkAddress (BYTE  *networkAddress)
```

IPX Relinquish Control (0Ah)

Devuelve el control a una CPU de una estación.

Registros de Entrada
BX 0Ah

Registros de Salida
Ninguno

Código de Completitud
Ninguno

Cuando este llamado es hecho, la aplicación devuelve temporariamente el control del tiempo del procesador a otros procesos que están corriendo en la estación. Este llamado es importante cuando se está ejecutando sobre un server o un puente.

También informa al driver de comunicaciones que la CPU está temporariamente libre. Sobre una estación normal, este llamado invoca a un procedimiento polling provisto por el driver de red, para dar la oportunidad de usar la CPU para enviar o recibir paquetes a otros procesos.

La interfase en C de NetWare es :
void IPXRelinquishControl (void)

IPX Disconnect From Targer (0Bh)

Informa a un socket específico que está esperando recibir paquetes, que la aplicación no intenta transmitir ninguno.

Registros de Entrada
BX 0Bh

ES:SI Dirección de buffer de pedido

Registros de Salida
Ninguno

Código de Completitud
Ninguno

Buffer de Pedido (12 bytes)

Offset	Contenido	Tipo	Orden
0	Red Destino	BYTE[4]	high-low
4	Nodo Destino	BYTE[6]	high-low
10	Socket Destino	BYTE[2]	high-low

Esta llamada es una cortesía del driver de comunicación de red que opera estrictamente sobre una comunicación punto-a-punto en el nivel de transporte físico. Una vez informado el driver sobre el nodo destino puede cerrar alguna conexión virtual con el nodo local.

Una aplicación nunca debería llamar a esta rutina con un ESR.

La interfase en C de NetWare es :

```
void IPXDisconnectFromTarget (BYTE *networkAddress)
```

3.12.2 APIs del Protocolo SPX

SPX Initialize (10h)

Determina cuando SPX está instalado sobre el nodo donde la aplicación está corriendo.

Registros de Entrada

BX	10h
AL	00h

Registros de Salida

AL	SPX Flag de Instalación
BH	Número de Versión Mayor SPX
BL	Número de Versión Menor SPX
CX	Máximo de conexiones soportadas por SPX
DX	Cantidad de Conexiones Disponibles SPX

Código de Completitud

Ninguno

El **Flag de Instalación** SPX indica cuando SPX está instalado:

00h	No instalado
FFh	Instalado

El **Número de Versión Mayor SPX** y el **Número de Versión Menor SPX** indican cuál versión está instalada. Por ejemplo, si la versión es 1.0, retorna 1 en el primer campo y 0 en el segundo.

El **Máximo de Conexiones Soportadas** por SPX es el número máximo de conexiones SPX que están en el archivo SHELL.CFG.

La **Cantidad de Conexiones Disponibles** SPX indica cuántas conexiones están disponibles para una aplicación.

La interfase en C de NetWare es :

```
BYTE SPXInitialize (BYTE *majorRevisionNumber,
                    BYTE *minorRevisionNumber,
                    WORD *maxConnections,
                    WORD *availableConnections)
```

SPX Establish Connection (11h)

Intenta establecer una conexión con un socket que está escuchando.

Registros de Entrada

BX	11h
AL	Contador de Reintentos
AH	Flag Watchdog de SPX
ES:SI	Dirección ECB

Registros de Salida

AL	Código de Completitud
DX	ID de Conexión

Código de Completitud

00h	SPX intentando conectarse con el Socket Destino
EFh	Tabla de Conexiones Locales Completa
FDh	Contador de Fragmento no es 1; el tamaño del buffer no es 42
FFh	Socket de Transmisión no está abierto

El programador debe pasar el **Contador de Intentos, Flag Watchdog**, y la **Dirección ECB** con el propósito de establecer una conexión SPX.

Antes de hacer este llamado, la aplicación debe inicializar los siguientes campos del ECB: **Dirección ESR, Número de Socket, Contador de Fragmento** y **Descriptor de Fragmento.** El campo **Contador de Fragmento** debería ser inicializado en 01h. El **Descriptor de Fragmento** debe apuntar a un buffer de 42-bytes que contenga la cabecera de un paquete SPX. También la aplicación debe inicializar los campos de esta cabecera, **Red Destino, Nodo Destino** y **Socket Destino.** Ninguno de estos campos puede setearse a -1 (broadcast).

Finalmente, la aplicación debe crear al menos dos ECBs para escuchar, y pasarlos a SPX con **SPX Listen For Sequenced Packet.** Una vez que SPX envía el paquete Establish Connection mencionado anteriormente, SPX usará uno de los ECBs que están escuchando para recibir la confirmación del paquete desde el nodo destino.

Cuando se hace este llamado, SPX trata de establecer la conexión en dos pasos. Primero, SPX crea una conexión local y verifica que esté funcionando correctamente. Segundo, SPX transmite un paquete Establish Connection a través de esta conexión para un socket específico de destino que está escuchando en espera de establecer una conexión. Para recibir el paquete Establish Connection, el socket destino tiene ECBs escuchando, que fueron puestos con una llamada al SPX Listen For Connection.

Para el primer paso, SPX requiere que el **Contador de Intentos**, el **Flag Watchdog** y los campos de **Dirección** ECB tengan los valores apropiados.

El **Contador de Reintentos** especifica cuántas veces SPX retransmitirá paquetes de confirmación antes de concluir que el nodo destino no está funcionando apropiadamente. Este campo debería setearse en 00h, instruyendo a SPX para que use su propio contador de reintentos. Un valor de 1 a 255 (inclusive) indica que SPX debería retransmitir paquetes el número especificado de veces.

El proceso **watchdog** monitorea una conexión SPX, asegurando que la misma esté apropiadamente funcionando cuando no haya tráfico en ella. Si el proceso watchdog determina que una conexión SPX ha fallado, el campo **Código de Completitud** tendrá

el valor EDh (falla de conexión) en algún ECB que está escuchando. Este proceso también registra la falla en el **Número ID de Conexión**, en el campo del ECB **Area de Trabajo**, y en la rutina de servicio de evento del ECB.

La aplicación debería inicializar el campo **Flag Watchdog** de SPX con el valor 01h para habilitar este proceso. Un valor de 00h (exitoso), lo deshabilita.

Una vez que SPX crea y verifica la conexión local, la llamada retorna un **Código de Completitud.** SPX también registra éste código en el campo **Código de Completitud** del ECB transmitido. Si el llamado retorna un valor distinto de 00h (exitoso), el intento de establecer una conexión termina en este punto.

Si la llamada retorna un Código de Completitud de 00h (exitoso), también retorna un **Número ID de Conexión.** Aunque la conexión no haya sido establecida con el socket destino, este Número ID de Conexión ocupa una entrada en la Tabla de Conexión SPX. SPX entonces transmite un paquete Establish Connection a el nodo destino.

Después de enviar el paquete Establish Connection al nodo destino (y retransmitirlo un número específico de veces), SPX setea el **Código de Completitud** del ECB a un valor apropiado. SPX también setea el **Flag En Uso** a 00h (disponible).

Los posibles valores del **Código de Completitud** son:

EDh	SPX no responde desde el nodo origen, una falla de hardware ha ocurrido, o la aplicación ha usado un SPX Abort Connection.
EFh	La Tabla de Conexiones está completa, ninguna conexión puede ser inicializada antes que una conexión activa termine. Esta tabla tiene 100 entradas.
FCh	El socket de SPX está cerrado, el socket de la conexión fue cerrado antes que el comando se completara y la rutina de servicio de evento del ECB no fue ejecutada.
FDh	El paquete SPX está malformado, el contador de fragmento no es 1 o el tamaño del buffer no es 42.
FFh	Socket SPX no está abierto.

La Tabla de Conexiones de una estación está limitada a un número configurable de conexiones. Sin embargo, todas las entradas no represetan necesariamente una conexión. Los siguientes ECBs ocupan una entrada en la Tabla del nodo: ECBs que participan en una conexión y ECBs que intentan escuchar para establecer una conexión.

Los programadores deberían usar SPX Abort Connection (no IPX Cancel Event) para cancelar SPX Establish Connection.

Ambos lados de una conexión SPX pueden residir en una estación.

La interfase en C de NetWare es :

```
int SPXEstablishConnection (BYTE retryCount,
                            BYTE watchDog
                            WORD *connectionID,
                            ECB  *eventControlBlock)
```

SPX Listen For Connection (12h)

Intenta recibir un paquete Establish Connection.

Registros de Entrada

BX	12h
AL	Contador de Reintentos
AH	SPX Flag Watchdog
ES:SI	Dirección ECB

Registros de Salida

Ninguno

Código de Completitud

Ninguno

Este llamado debe tener inicializados el **Contador de Reintentos, SPX Flag Watchdog** y **Dirección de ECB** a SPX con el propósito de escuchar y recibir un paquete Establish Connection. Una vez hecho el llamado, la rutina retorna y SPX comienza a escuchar un paquete Establish Connection.

Una aplicación debe inicializar el campo **Dirección ESR** y el campo **Número de Socket** del ECB que usará, luego debe crear al menos dos ECBs para escuchar y pasarlos al SPX con un SPX Listen For Sequenced Packet.

SPX Establish Connection y SPX Listen For Connection trabajan juntos para establecer una conexión entre sockets de una red.

SPX usa dos pasos para tratar de establecer una conexión. Primero, SPX crea una conexión local y verifica que funcione correctamente. Segundo, después que un paquete Establish Connection es recibido, la función intenta crear una conexión local. Si tiene éxito, un paquete de confirmación es enviado al nodo fuente, y ambos lados están listos para enviar y recibir paquetes de datos sobre la conexión. En suma, la llamada para escuchar un paquete Establish Connection enviado por el nodo origen usando SPX Establish Connection recibe el paquete, y envía una confirmación del paquete al nodo origen.

Para completar el primer paso que crea una conexión local, SPX requiere que el **Contador de Reintentos, SPX Flag Watchdog,** y la **Dirección ECB** sean inicializados con los valores apropiados.

El proceso **watchdog** monitorea una conexión SPX, asegurando que la misma esté apropiadamente funcionando cuando no haya tráfico en ella. Si el proceso watchdog determina que una conexión SPX ha fallado, el campo Código de Completitud tendrá el valor EDh (falla de conexión) en algún ECB que está escuchando. Este proceso también registra la falla en el Número ID de Conexión, en el campo del ECB Area de Trabajo, y en la rutina de servicio de evento del ECB.

La aplicación debería inicializar el campo Flag Watchdog de SPX con el valor 01h para habilitar este proceso. Un valor de 00h (exitoso), lo deshabilita.

Si SPX no puede crear una conexión local porque el socket no está abierto o la Tabla de Conexión está llena, SPX setea el campo Código de Completitud con uno de los siguientes códigos y el campo Flag En Uso a 00h (disponible).

EDh	La Tabla de Conexiones Locales está completa
FFh	Socket receptor de SPX no está abierto.

Si SPX no tiene problemas, el ECB está listo para recibir un paquete Establish Connection.

Si la aplicación cancela el Establish Connection en alguno de los reintentos, SPX registra un valor FCh (pedido cancelado) en el Código de Completitud del ECB.

Tanto los ECBs que intenta establecer una conexión como los ECBs que participan en conversaciones ocupan una entrada en la Tabla de Conexiones de SPX del nodo.

Se puede usar IPX Cancel Event para cancelar un SPX Listen For Connection.

La interfase en C de NetWare es :

```
void SPXListenForConnection (BYTE retryCount,
                             BYTE watchDog,
                             ECB  *eventControlBlock)
```

SPX Terminate Connection (13h)

Termina una conexión SPX.

Registros de Entrada

BX	13h
DX	ID de Conexión
ES:SI	Dirección ECB

Registros de Salida

Ninguno

Código de Completitud

Ninguno

Este llamado le pasa un número **ID de Conexión** y la **Dirección de un ECB** al SPX, retornando inmediatamente al programador mientras intenta terminar la conexión.

Antes de hacer esta llamada la aplicación debe inicializar los campos **Dirección ESR, Contador de Fragmento, Descriptor de Fragmento.** La aplicación debe setear el **Contador de Fragmento** a 1 y el **Descriptor del Fragmento** debe apuntar a un paquete SPX.

Cuando el llamado se está ejecutando, SPX registra el valor FEh (terminar conexión) en el campo Tipo de Dato de la cabecera del SPX referenciado en el ECB. SPX entonces entrega el paquete al nodo compañero, retorna el Código de Completitud en el campo Código de Completitud del ECB, le pone 00h (disponible para uso) en el campo Flag En Uso. Finalmente, SPX llama a la rutina de servicio de evento apuntada por el campo Dirección ESR (si corresponde).

Los posibles valores del **Código de Completitud** son los siguientes:

00h	La conexión fue terminada con éxito.
FDh	El paquete está malformado. El contador de fragmento no es 1 o el tamaño del buffer no es 42. En este caso, la conexión es abortada.
EEh	El Número ID de Conexión contenido en DX no corresponde con una conexión válida. Ninguna conexión fue terminada.
EDh	La conexión terminó anormalmente. El nodo de la conexión remota no envío la confirmación del pedido de terminar la conexión en una cantidad de tiempo apropiada. La conexión es terminada sobre el lado local, pero SPX no puede garantizar que la otra parte haya visto el Terminate Connection. Este error también es retornado como falla de red o el paquete no pudo ser entregado al destino especificado.
ECh	La conexión fue terminada por la otra estación sin confirmar este paquete Terminate Connection. La conexión es cerrada, pero SPX no puede garantizar que la otra parte vea el paquete Terminate Connection.

Una vez que SPX termina una conexión, la entrada del número ID de Conexión ocupado en la Tabla de Conexiones se libera.

Una aplicación debería usar un SPX Abort Connection (no IPX Cancel Event) para cancelar SPX Terminate Connection.

La interfase en C de NetWare es :

```
int SPXTerminateConnection (WORD connectionIDNumber,
                            ECB  *evenControlBlock)
```

SPX Abort Connection (14h)

Aborta una conexión SPX.

Registros de Entrada

BX	14h
DX	ID de Conexión

Registros de Salida

Ninguno

Código de Completitud

Ninguno

El llamado retorna inmediatamente mientras SPX aborta solamente el lado de la conexión que hace este llamado.

SPX no intenta informar al otro nodo que la conexión va a ser abortada. Este lo descubrirá cuando intente enviar un paquete sobre la conexión o cuando su watchdog haga un chequeo después que el tiempo de inactividad expire.

Algún paquete de SPX Establish Connection, SPX Terminate Connection, o SPX Send Sequenced pueden estar siendo transmitidos sobre la conexión que aborta. En estos casos el Código de Completitud será puesto en EDh, indicando terminación anormal de la conexión.

El programador debería asegurarse que la rutina del servicio de evento de los ECBs afectados sean llamadas (a menos que la Dirección ESR sea nula). Si alguno de los ECBs involucrados están en un estado en el cual no pueden ser cancelados (como cuando la tarjeta de interfase de red está en la mitad de una transmisión de paquete), el ECB será cancelado lo más rápido posible. En este caso, Código de Completitud será puesto en EDh, y el ESR es llamado si existe.

Este llamado es incluido para permitir que un lado de la conexión unilateralmente termine con la ella si alguna condición fatal es detectada. Bajo condiciones normales, el SPX Terminate Connection debería ser usado para asegurarse que ambos lados abortan la conexión en una forma controlada.

La interfase en C de NetWare es :

```
void SPXAbortConnection (WORD connectionIDNumber)
```

SPX Get Connection Status (15h)

Retorna el status de una conexión SPX.

Registros de Entrada

BX	15h
DX	ID de Conexión
ES:SI	Dirección del Buffer de Respuesta

Registros de Salida

AL	Código de Completitud

Código de Completitud

00h	La conexión está activa
EEh	No existe la conexión

Buffer de Respuesta (46 bytes)

Offset	Contenido	Tipo	Orden
0	Estado de la Conexión	BYTE	high-low
1	Watchdog habilitado	BYTE	high-low
2	ID de Conexión Local	WORD	high-low
4	ID de Conexión Remota	WORD	high-low
6	Número de Secuencia	WORD	high-low
8	Número de Confirmación Local	WORD	high-low
10	Número de Ubicación Local	WORD	high-low
12	Número de Confirmación Remoto	WORD	high-low
14	Número de Ubicación Local	WORD	high-low
16	Socket Local	BYTE[2]	high-low
18	Dirección Inmediata	BYTE[6]	high-low
24	Red Remota	BYTE[4]	high-low
36	Contador de Retransmisiones	WORD	high-low
38	Estimación de Demora de ida-vuelta	WORD	high-low
40	Paquetes Retransmitidos	WORD	high-low
42	Paquetes Suprimidos	WORD	high-low

Este llamado permite chequear el estado de una conexión. Si la conexión especificada no existe, entonces el Código de Completitud (EEh) será retornado.

En la explicación que sigue, la estación local es la estación donde la aplicación está corriendo y el SPX local reside. El SPX local puede tener varias conexiones. Las estaciones remotas, sin embargo, se refieren a las estaciones que están en el otro lado de la conexión. El SPX remoto está corriendo en la estación remota.

Estado de la Conexión: indica el estado actual de la conexión que se especifica. Los posibles estados definidos son cuatro:

- 01h **Espera:** SPX está escuchando (hay un SPX Listen For Connection) sobre la conexión, está esperando recibir un paquete Establish Connection.
- 02h **Comenzando:** SPX está intentando crear (hay un SPX Establish Connection) una conexión completa con una estación remota mediante el envío de un paquete Establish Connection sobre su conexión local.
- 03h **Establecida:** SPX ha establecido una conexión con una estación remota, y la conexión está abierta para la transmisión de paquetes en ambos sentidos.
- 04h **Terminada:** El SPX remoto ha terminado la conexión (SPX Terminate Connection). Sin embargo, el SPX local aún no ha terminado su conexión local.

Watchdog Habilitado: indica si el proceso watchdog está monitoreando el SPX local. Si lo está, el segundo bit del campo (0x02) indica la presencia del mismo. Los otros siete bits del campo son de uso interno.

ID de Conexión Local: es el número de la conexión SPX desde el punto de vista de la estación local. El número ID de Conexión Local es el mismo número que la aplicación lee en el registro DX antes de hacer el llamado.

Número de Secuencia: indica el número de secuencia del próximo paquete de la conexión local que será enviado a la estación remota. SPX asigna un número de secuencia (0000h a FFFFh) a cada paquete que envía a la estación remota. Este asegura la

transmisión en secuencia en la conexión local y la recepción en secuencia en la estación remota. Cuando el número de secuencia llega a FFFFh, éste comienza nuevamente en 0000h. Este campo no es válido si la estación está esperando.

Número de Ubicación Local: es el número de buffers disponibles por la conexión SPX para recibir los paquetes que llegan. El SPX remoto puede enviar paquetes con un número de secuencia inferior o igual, pero no superior, al Número de Ubicación Local. El SPX local incrementa el Número de Ubicación Local de acuerdo a los ECBs generados para escuchar en la estación. En este sentido, el SPX local regula el número de paquetes que el SPX remoto le envía, y permite que no lo invada con paquetes que no se puede recibir. Cuando este número llega a FFFFh, este retorna a 0000h. Este campo no es válido si el estado de la conexión es espera. Este número está basado en el número de los ECBs listos para escuchar.

Número de Confirmación Remota: es el Número de Secuencia del próximo paquete que el SPX remoto espera recibir desde el SPX local. Cuando este número de secuencia alcanza el FFFFh, comienza nuevamente en 0000h. Este campo no es válido si el estado de la conexión es espera.

El SPX local puede enviar paquetes con un Número de Secuencia inferior o igual, pero no superior, al Número de Ubicación Remota. El SPX remoto incrementa el Número de Ubicación Remota cuando la estación remota genera ECBs para escuchar. En este sentido, el SPX local regula el número de paquetes que el SPX remoto le envía, y permite que no los invada con paquetes que no se pueden recibir. Cuando este número llega a FFFFh, este retorna a 0000h. Este campo no es válido si el estado de la conexión es espera. Este número está basado en el número de los ECBs listos para escuchar.

Socket Local: es el Número de Socket que el SPX local está usando para enviar y recibir paquetes.

Dirección Inmediata: retorna la dirección del nodo puente (sobre la red local) que rutea el paquete a la estación remota o desde ella. Si la estación local y remota residen sobre la misma red, la Dirección Inmmediata es la dirección del nodo de la estación remota (en este caso el puente no es necesario). Este campo no es válido si la conexión está en estado de espera.

Red Remota: retorna el número de la red sobre la cual la estación remota reside.

Nodo Remoto: retorna la dirección del nodo destino. Este campo no es válido si la conexión está en estado de espera.

Contador de Retransmisión: es el número de veces que SPX intenta retransmitir un paquete de confirmación antes de determinar que el SPX remoto está inoperable o inalcanzable. Este campo no es válido si la conexión está en estado de espera.
Estimación de Demora: es el tiempo estimado de ida y vuelta de un paquete. Indica en ticks de reloj el tiempo que SPX espera (1/18 ticks de reloj son aproximadamente 1 segundo) la confirmación de un paquete. Debido a que el SPX local tiene fluctuaciones en el tiempo y ajustes, el valor de tiempo estimado de ida y vuelta puede cambiar. Este campo no es válido si la conexión está en estado de espera.

Paquetes Retransmitidos: es el número de veces que el SPX local ha tenido que retransmitir un paquete sobre la conexión antes de recibir la confirmación esperada. Cuando este número llega a FFFFh, este retorna a 0000h. Este campo es válido solamente si el estado de la conexión es establecida o terminada.

Paquetes Suprimidos: es el número de veces que el SPX local ha recibido y descartado un paquete porque el mismo es el duplicado de otro recibido o porque el paquete está fuera de los límites de la ventana de recepción. Cuando este número llega a FFFFh, este retorna a 0000h. Este campo es válido solamente si el estado de la conexión es establecida o terminada.

La interfase en C de NetWare es :

```
int SPXGetConnectionStatus (WORD  connectionIDNumber,
                            CONNECTION_INFO_STRUCT *connectionInfo)
```

SPX Send Sequenced Packet (16h)

Envía un paquete SPX.

Registros de Entrada

BX	16h
DX	ID de Conexión
ES:SI	Dirección ECB

Registros de Salida
Ninguno

Código de Completitud
Ninguno

Cuando hace esta llamado retorna inmediatamente mientras SPX intenta enviar el paquete SPX.

La aplicación debe inicializar los campos **Dirección del ESR, Contador de Fragmento, Descriptor de Fragmento.** El buffer de fragmento referenciado por el primer **Descriptor de Fragmento** debe contener al menos 42 bytes para la cabecera del paquete SPX. La aplicación debe también inicializar el **bit Fin de Mensaje** de la cabecera del paquete de SPX, en los campos **Control de Conexión** y **Tipo de Dato.**

Una vez hecha la llamada, SPX setea el campo Flag En Uso a un valor distinto de cero para indicar que el ECB está enviando un paquete. SPX también pone la combinación ECB/paquete en una cola para transmitirlo.

Cuando SPX completa su intento de transmitir el paquete, registra el Código de Completitud en el campo Código de Completitud, y setea el campo Flag En Uso a 00h (disponible), y llama a la rutina de servicio de evento, si existe alguna.

Los posibles valores del **Código de Completitud** del ECB son:

00h	El paquete fue enviado satisfactoriamente y recibido en orden por la otra parte. Una confirmación del lado opuesto fue recibida.
ECh	La parte remota terminó la conexión sin confirmación del paquete. SPX no puede garantizar que el paquete haya sido recibido antes de la destrucción de la conexión.
EDh	La conexión finalizó anormalmente. Una de las partes usó la llamada Abort Connection para abortar la conexión, o el otro lado falló al enviar la confirmación de recepción del paquete. Este error es también informado como falla de hardware o el paquete no puede ser entregado al nodo destino.
EEh	El registro DX no contiene el número ID de Conexión de una conexión establecida.

FCh El socket de conexión fue cerrado. En este caso, la rutina del servicio de evento no es llamada.

FDh El paquete fue malformado. El Contador de Fragmento es cero, el primer fragmento es menor que 42 bytes, o el paquete entero es mayor que 576 bytes. Este error causa que la conexión sea abortada.

SPX pone los ECBs en una cola y los envía en el orden en que son recibidos. Los sockets usados por conexiones SPX no pueden ser usados para enviar o recibir paquetes IPX. Finalmente, cuando una aplicación o algún otro agente cierra un socket SPX, SPX termina todas las conexiones asociadas con el socket.

Una aplicación puede establecer una conexión entre dos sockets que residen en el mismo nodo.

Una aplicación debería usar SPX Abort Connection (no IPX Cancel Event) para cancelar un evento SPX Send Sequenced Packet.

La interfase en C de NetWare es :

```
void SPXSequencedPacket (WORD connectionIDNumber,
                         ECB  *eventControlBlock)
```

SPX Listen For Sequenced Packet (17h)

Recibe un paquete en secuencia SPX.

Registros de Entrada

BX	17h
ES:SI	Dirección ECB

Registros de Salida

Ninguno

Código de Completitud

Ninguno

Este llamado retorna inmediatamente mientras SPX comienza a escuchar la llegada de paquetes SPX.

La aplicación debe inicializar los campos **Dirección del ESR, Número de Socket, Contador de Fragmento, Descriptor de Fragmento.** El buffer de fragmento referenciado por el primer **Descriptor de Fragmento** debe contener al menos 42 bytes (para recibir la cabecera del paquete SPX que llega).

Inicialmente, SPX setea el campo Flag En Uso del ECB a un valor distinto de cero, indicando que el ECB está esperando recibir un paquete. SPX también suma el ECB a un pool de buffers de ECBs que están escuchando paquetes en secuencia sobre el mismo socket. Cuando un paquete arriba, SPX usa uno de los ECBs que están escuchando para recibirlo. SPX registra un valor apropiado en el campo Código de Completitud del ECB, setea el campo Flag En Uso a 00h (disponible para uso), y llama a la rutina de servicio de evento referenciada por el campo Dirección ESR (si corresponde).

Los posibles valores del **Código de Completitud** del ECB son:

00h	El paquete fue recibido correctamente sobre una conexión activa.
FFh	El socket, en el cual el ECB debe escuchar, no fue abierto; el Contador de Fragmento no es cero, o el primer fragmento de la Lista de Descriptores de Fragmentos es menor que 42 bytes.
FDh	Un overflow de paquete ha ocurrido. Un paquete ha sido recibido en el orden correcto, pero el espacio disponible por la Lista de Descriptores de Fragmentos no es lo suficientemente grande para contenerlo.
FCh	El pedido de escuchar fue cancelado por un Close Socket o Cancel Event. En este caso, la rutina de servicio de eventos no es llamada.
EDh	El SPX watchdog determina que una falla de conexión o un abortar conexión ha ocurrido. El Número ID de Conexión de la conexión terminada está en la primer palabra del campo Area de Trabajo del IPX ECB.

La interfase en C de NetWare es :

```
void SPXListenForSequencedPacket (ECB
                                  *eventControlBlock)
```

4
ADMINISTRACION DE LA INFORMACION

Mientras un proceso está corriendo puede almacenar una cantidad de información limitada a la capacidad de la memoria, la cual puede ser inadecuada.

Al finalizar el proceso la información manejada por el mismo se pierde. Para muchas aplicaciones (por ejemplo bases de datos) ésta debe ser retenida por mucho tiempo, quizás para siempre.

También, se puede hacer necesario que múltiples procesos accedan a la información simultáneamente.

Por lo tanto, existen tres razones esenciales para almacenar la información en forma permanente:

1- Debe ser posible almacenar una gran cantidad de información.

2- La información debe sobrevivir a la terminación del proceso que la usa.

3- Múltiples procesos deben poder acceder a ella en forma concurrente.

La solución de todos estos problemas es almacenar la información sobre discos u otros medios en unidades llamadas **archivos.**

Los procesos pueden leerlos o crearlos. El almacenamiento es persistente, no se ve afectado por la creación o terminación de procesos. Un archivo solamante desaparece cuando es borrado.

Los archivos son almacenados por el sistema operativo. Este determina cómo ellos son estructurados, nombrados, accedidos, usados, protegidos e implementados. La parte del sistema operativo que administra los archivos recibe el nombre de **Sistema de Archivos** o **Administrador de la Información.**

4.1 Archivos

En esta sección describiremos qué es un archivo, cómo es usado y qué propiedades tiene.

4.1.1 Nombre de Archivos

Los archivos son un mecanismo de abstracción. Ellos proveen un camino para almacenar información sobre discos y recuperarla. Esto es hecho de tal forma que los usuarios se desentienden de los detalles de cómo y dónde la información es almacenada, y cómo los discos trabajan sobre ella.

Probablemente la característica más importante del mecanismo de abstracción es que el archivo recibe un nombre. Cuando un proceso crea un archivo, éste le da un nombre. Cuando el proceso termina, el archivo continúa existiendo, y se puede accederse a él usando el nombre que se le asignó.

Existen reglas para dar nombres a los archivos, éstas varian de un sistema operativo a otro.

La reglas para el sistema operativo MS-DOS son:

- La longitud máxima que puede tener el nombre es de ocho caracteres.
- Puede estar formado por letras, dígitos, y caracteres especiales, excepto: coma, blanco y punto.
- No pueden usarse las palabras reservadas: CON, AUX, PRN, CLOCK$, LPT1, LPT2, LPT3, NULL, COM1, COM2, COM3, COM4.

No se distingue entre letras mayúsculas y minúsculas, siendo iguales, por ejemplo, los nombres Pepe y pepe.

El nombre se puede dividir en dos partes separadas por un punto. La segunda parte se denomina **extensión.** Usualmente indica alguna propiedad del archivo. En DOS puede tener como máximo tres caracteres.

4.1.2 Tipos de Archivos

Los sistemas operativos soportan distintos tipos de archivos. MS-DOS, por ejemplo, tiene archivos regulares, directorios, archivos de caracteres especiales, archivos de bloque especiales y archivos binarios.

Los **archivos regulares** son los que contienen información del usuario.

Los **directorios** son sistemas de archivos que permiten mantenerlos organizados.

Los **archivos de caracteres especiales** están relacionados con la entreda/salida y son usados por los dispositivos del modelo serial E/S tales como terminales, impresoras y redes.

Los **archivos de bloques especiales** generalmente son archivos ASCII o archivos binarios. Los archivos ASCII están compuestos por líneas de texto. En algunos sistemas cada línea termina por un caracter carriage return, en otros, se utiliza el caracter line feed. Ocacionalmete, ambos son requeridos (MS-DOS). Las líneas no necesitan ser de la misma longitud.

Los **archivos binarios** usualmente tienen una estructura interna y pueden ser ejecutados por el sistema operativo.

4.1.3 Acceso a Archivos

Al comienzo los sistemas proveeían sólo **acceso secuencial** a los archivos. En estos sistemas, un proceso puede leer todos los bytes o registros de un archivo en orden, comenzando por el principio, pero no pueden saltearse partes o leerlos fuera de orden. Los archivos secuenciales son convenientes cuando el medio de almacenamiento es una cinta, pero no cuando es un disco.

Con el advenimiento del uso de los discos, se hizo posible leer los bytes o los registros de un archivo fuera de orden, o el acceso a registros por clave, en lugar de por posición. Archivos cuyos bytes o registros pueden ser leídos en cualquier orden son llamados **archivos de acceso random.**

Existen dos métodos para especificar dónde se comienza a leer. El primero, una operación LEER se posiciona al comienzo del archivo. El segundo, una operación especial BUSCAR, setea la posición deseada. Después de posicionarse en el archivo, se puede leer desde la posición actual en forma secuencial.

4.1.4 Atributos de los Archivos

Todos los archivos tienen un nombre y datos. Todos los sistemas operativos asocian otra información a un archivo, por ejemplo, la fecha y la hora de creación. Llamamos a estos items extras **atributos de archivos.** La lista de atributos varía de un sistema a otro.

En MS-DOS los atributos son:

Atributo	Significado
Flag de Oculto	0 normal, 1 no se muestra.
Flag de Sistema	0 archivo normal, 1 archivo del sistema.
Flag de Lectura Solamente	0 lectura/escritura, 1 lectura solamente.
Flag ASCII/binario	0 archivo ASCII, 1 archivo binario.
Hora	Hora de creación o modificación del archivo.
Fecha	Fecha de creación o modificación del archivo.
Tamaño	Número de bytes del archivo.

Los flags son bits que controlan o habilitan algunas propiedades específicas.

4.1.5 Operaciones sobre los Archivos

Los dististintos sistemas operativos proveen diferentes operaciones sobre los archivos.

- **Crear:** el archivo es creado sin datos. El propósito de esta operación es que el archivo exista y posea algunos de los atributos.

- **Borrar:** Cuando el archivo no es necesario puede ser borrado para dejar espacio libre en el disco. Todos los sistemas tienen esta operación.

- **Abrir:** Antes de usar un archivo un proceso debe abrirlo. El propósito de esta operación es que el sistema conozca los atributos del mismo y guarde en memoria la lista de direcciones del disco para acceder a él más rápidamente.

- **Cerrar:** cuando todos los accesos han finalizado, los atributos y las direcciones del disco no son más necesarios, por lo tanto puede ser cerrado el archivo para liberar el espacio interno ocupado. Muchos sistemas operativos tienen limitado el número de archivos que un proceso puede abrir.

- **Leer:** los datos son leídos desde un archivo. Usualmente se leen los bytes desde la posición corriente. Quien hace el llamado debe especificar la cantidad de bytes ha leer y debe proveer un buffer donde poner la información.

- **Escribir:** los datos son escritos en un archivo, usualmente en la posición corriente. Si la posición actual es el fin del archivo, el tamaño del archivo se incrementa. Si en la posición actual existen datos, los nuevos se sobre escriben, y los primeros se pierden.

- **Adicionar:** esta operación es una forma de restringir la escritura, solamente sumar datos al final del archivo. Algunos sistemas operativos no tienen esta operación.

- **Buscar:** para archivos de acceso random se necesita un método para especificar desde dónde se toman los datos. Una aproximación es esta llamada al sistema operativo, la cual posiciona el puntero en un lugar específico del archivo. Luego que esta llamada finaliza, los datos pueden leerse desde esta posición o escribirse a partir de la misma.

4.2 Directorios

Para organizar los archivos se utilizan **directorios**. En esta sección describiremos los directorios, su organización, sus propiedades, y las operaciones que se pueden ejecutar sobre ellos.

4.2.1 Sistema de Directorio Jerárquico

Un directorio contiene un número de entrada por archivo. Una posibilidad es la mostrada en la figura 4.1, en la cual cada entrada contiene el nombre de un archivo, los atributos del archivo y la dirección del disco donde se encuentra alojado. Otra posibilidad es la mostrada en la figura 4.2. Aquí una entrada en el directorio contiene el nombre de un archivo y apunta a otra estructura de datos donde se encuentran el resto de los atributos y la dirección sobre el disco. Ambos sistemas son usados comúnmente.

Estructura de Datos Conteniendo Atributos

game	atributos
mail	atributos
ms	atributos

Fig. 4.1

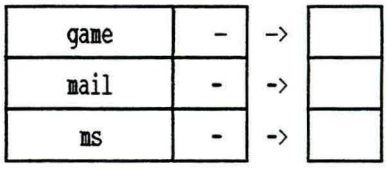


Fig. 4.2

Cuando se desea abrir un archivo es abierto, el sistema operativo lo busca en el directorio hasta que encuentra su nombre. Este, entonces, extrae los atributos y la dirección del disco, directamente desde la entrada del directorio o desde la estructura de datos apuntada, y lo pone en una tabla en memoria. Todas las referencias siguientes al archivo usan la información de la memoria principal.

El número de directorios varía según los sistemas operativos. El diseño más simple es tener un sólo directorio que contenga todos los archivos. Pero el mejor diseño es aquel que agrupa los archivos según un camino lógico. Por ejemplo, un profesor

puede tener una colección de archivos que juntos formen un libro que está escribiendo para un curso, una segunda colección de archivos conteniendo programas de una aplicación, un tercer grupo de archivos conteniendo el código de un compilador, y así sucesivamente.

Existe la necesidad de tener una jerarquía de directorios (árbol de directorios).

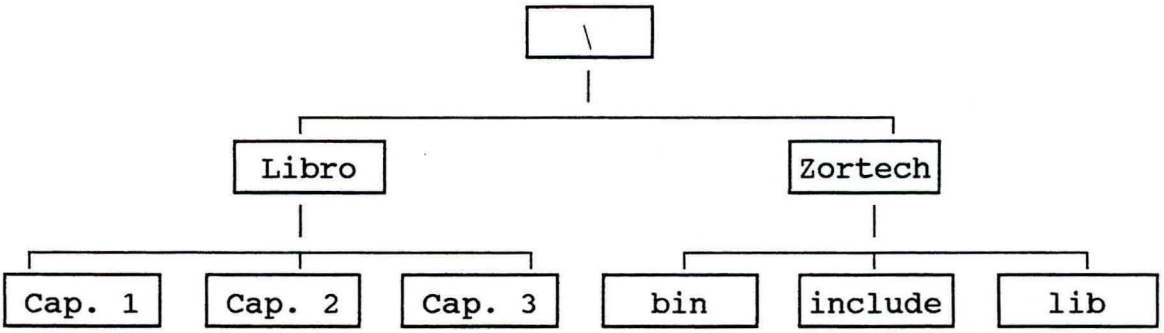


Fig. 4.3 Arbol de Directorios

4.2.2 Nombres de Directorios

Cuando el sistema de archivos está organizado como un árbol de directorios, son necesarios caminos para especificar los nombres de los archivos. Dos métodos son usados.

En el primer método, cada archivo es nombrado con su **camino absoluto**, que consiste en el camino desde la raíz hasta el nombre del archivo. Como por ejemplo, el camino c:\ms\box.txt significa que la raíz del directorio c:\ contiene un subdirectorio ms que tiene un archivo llamado box.txt. El camino absoluto siempre comienza con el directorio raíz y es único. En MS-DOS el separador de directorios es \.

En el segundo método, cada archivo es referenciado con su **camino relativo.** Este es usado en relación con el concepto de **directorio de trabajo** (también llamado **directorio corriente**). Un usuario puede designar un directorio como directorio de trabajo corriente, en el cual todos los caminos no comienzan en el directorio raíz sino que comienzan en el directorio relativo de trabajo. Por ejemplo, si el directorio corriente es c:\ms entonces el archivo cuyo camino absoluto es c:\ms\tesis.txt puede ser referenciado como tesis.txt. En otras palabras el comando en DOS

```
copy c:\ms\tesis.txt c:\ms\tesis.bak
```

y el comando

```
copy tesis.txt tesis.bak
```

hacen lo mismo si el directorio de trabajo es c:\ms.

Algunos programas necesitan acceder a un archivo específico sin conocer cuál es su directorio de trabajo. En este caso, ellos deberían usar siempre el camino absoluto.

También, pueden cambiar su directorio de trabajo al directorio donde se encuentra el archivo que necesitan y luego terminar su ejecución. Así, otros procesos no son afectados por el cambio, pero el sistema operativo cambió de directorio. Si se están usando procesos de librería cambiar de directorio y luego terminar sin volver al directorio anterior puede traer graves incovenientes.

Muchos sistemas operativos que soportan un sistema de directorio jerárquico tienen dos entradas especiales en todos los directorios "." (punto) y ".." (dos puntos). El punto hace referencia al directorio corriente y el dos puntos se refiere a su padre. Por ejemplo, supongamos que el directorio de trabajo sea c:\ms y se quiere copiar en él el archivo c:\wd\prueba.txt

```
copy ..\wd\prueba.txt .
```

El primer argumento le dice al sistema que vaya un directorio hacia arriba (su padre c:\) y luego descienda al directorio wd para hallar el archivo prueba.txt.

El segundo argumento nombra al directorio corriente. Por lo tanto copia el archivo en el directorio corriente.

4.2.3 Operaciones sobre los Directorios

Los sistemas permiten llamadas para administrar los directorios, éstas difieren de uno a otro.

- **Crear:** crea un directorio. Este directorio está vacío, excepto por el punto y dos puntos, los cuales son creados automáticamente por el sistema.

- **Borrar:** borra un directorio. Solamente se puede borrar un directorio vacío. Un directorio que contiene punto y dos puntos es considerado vacío.

- **Abrir Directorio:** los directorios pueden ser leídos. Por ejemplo para listar todos los archivos que contiene. Antes de ser leído un directorio debe ser abierto.

- **Cerrar Directorio:** cuando un directorio ha sido leído se cierra para liberar el espacio.

- **Leer Directorio:** esta llamada retorna la próxima entrada en el directorio abierto.

- **Renombrar:** en algunos sistemas los directorios son considerados archivos, y pueden ser renombrados al igual que éstos.

4.3 Implementación del Sistema de Archivos

Desde el punto de vista de la implementación nos interesa conocer cómo los archivos y los directorios son almacenados, cómo se administra el espacio en disco y cómo hace para trabajar en forma eficiente y confiable.

El sistema operativo debe contar con un módulo que se encargue del manejo de estos aspectos. El Administrador de Infomación posee las siguientes funciones básicas:

- Llevar rastro de toda la información del sistema a través de varias tablas, siendo la más importante el Directorio de Archivos, también llamado Tabla de Contenido del Volumen (VTOC). Estas tablas contienen el nombre, localización y derechos de acceso a la información dentro del sistema.

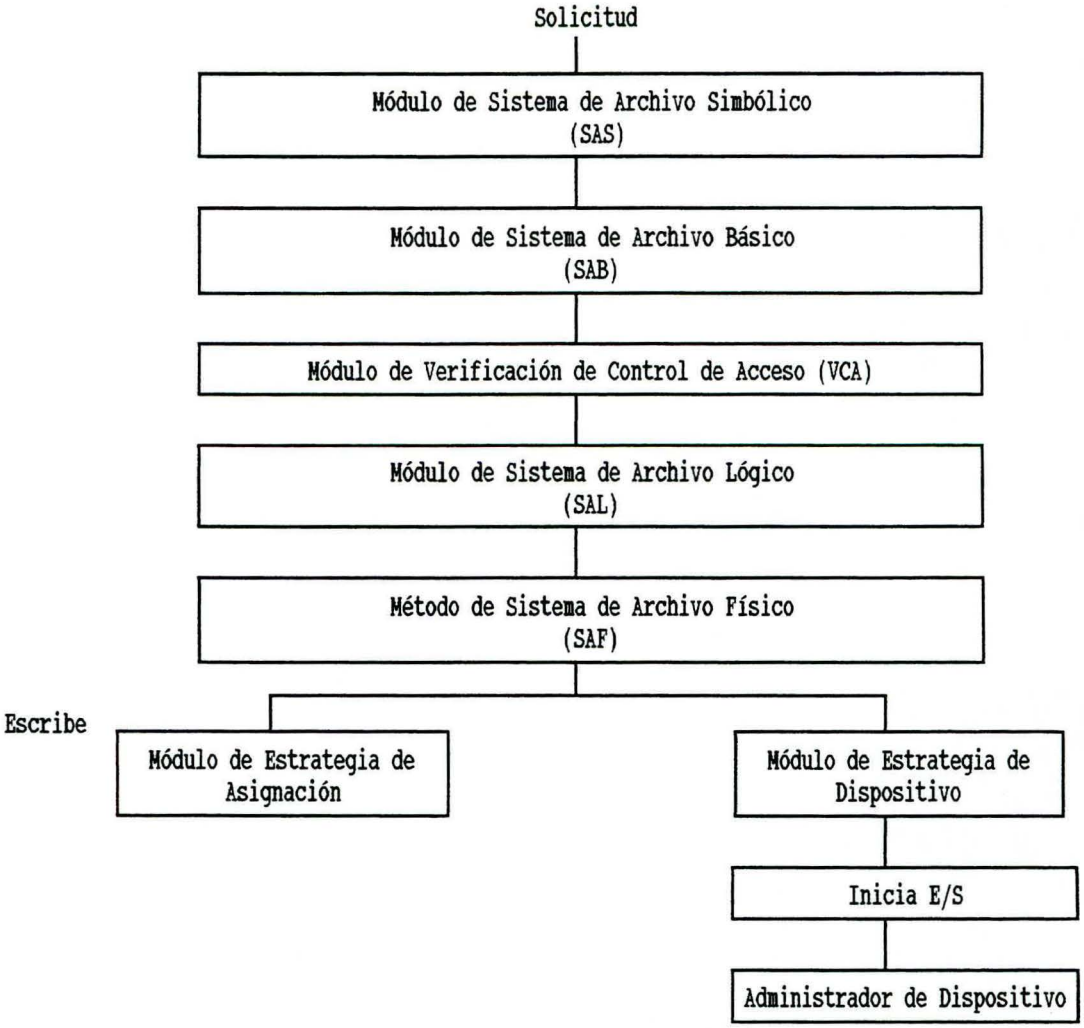
- Decidir qué política es utilizada para determinar dónde y cómo la información es almacenada. Algunos factores que influyen en esta política son la utilización eficiente del almacenamiento secundario, acceso eficiente, flexibilidad a usuarios, protección de los derechos de acceso a la información solicitada.

- Asignación del recurso de la información. Una vez que se toma la decisión de permitir a un proceso acceder a cierta información, los módulos de ubicación de la información deberán encontrarla, hacerla accesible y por último establecer los derechos de acceso.

- Desasignación del recurso de la información. Una vez que la información no se necesita más, las entradas en las tablas asociadas a esta pueden ser eliminadas. Si el usuario ha actualizado la información, la copia original puede ser actualizada para el posible uso de otros procesos.

4.3.1 Modelo General de un Sistema de Archivos

Consideramos que todos los módulos de la implementación se encuentran dispuestos en niveles o capas, de tal forma que dado un módulo, éste sólo depende de aquellos módulos que se encuentran en niveles inferiores a él y sólo efectúan llamadas a éstos módulos. Podemos ver dicho modelo como:



Dependiendo del sistema operativo específico algunos de éstos módulos se fusionan, se desglosan en aún más módulos, o incluso desaparecen. Aún así la estructura subyacente sigue siendo la misma.

4.3.1.1 Estrutura y Mantenimiento del Directorio de Archivos

El **Directorio de Archivos** es una tabla donde cada uno de sus entradas corresponden a un archivo. Definiendo a un **archivo** como una colección de unidades de información relacionadas llamadas **registros.**

El contenido de una entrada en el Directorio de Archivo de nuestro ejemplo contendrá:

- Número de entrada
- Longitud de registro lógico
- Cantidad de registros lógicos
- Longitud de registro físico
- Formato de registros
- Organización
- Dirección al primer bloque físico
- Protección y control de acceso

El número de entrada en la tabla se convertirá en el identificador del archivo dentro del sistema.

Desde el segundo hasta el séptimo campo de la tabla se obtendrá información para poder ubicar y mapear un registro lógico en una verdadera dirección física en el almacenamiento secundario.

La protección y el control de acceso informará qué usuario tiene acceso sobre el archivo y qué tipo de operaciones se pueden realizar sobre el archivo (sólo lectura, lectura/escritura, etc.).

Los comandos CREAR y BORRAR incorporan o eliminan respectivamente una entrada en el Directorio de Archivos.

Si todo el Directorio de Archivos fuera mantenido todo el tiempo en memoria principal, se necesitaría una gran cantidad de memoria sólo para este propósito, por lo que se mantiene el Directorio de Archivos como un archivo dentro de un volumen de almacenamiento (disco, diskette, cinta, etc.).

Solucionado este inconveniente el problema ahora es que la búsqueda de una entrada en dicho directorio puede ser considerablemente larga.

Este problema se soluciona si mantenemos en memoria aquellas entradas del Directorio que pertenecen a *archivos* que fueron referenciados anteriormente. En las llamadas siguientes no habrá desperdicio de tiempo de E/S.

Muchos sistemas operativos constan de pedidos especiales tales como ABRIR y CERRAR un archivo determinado. ABRIR coloca en memoria la entrada en la tabla perteneciente al archivo en cuestión y CERRAR la libera.

4.3.1.2 Sistema de Archivo Simbólico

El primer módulo que es llamado cuando se hace un pedido al Sistemas de Archivos es el módulo denominado Sistema de Archivo Simbólico (SAS). Una llamada puede ser:

SAS(función, nombre de archivo, número de registro, posición)

tal como:

SAS(LEER, "PEPE", 5, 1200)

Donde se pide que se lea el registro lógico número 5 del archivo "PEPE", para colocar su contenido en la dirección 1200 de memoria principal.

El Sistema de Archivo Simbólico usa el nombre del archivo para localizar la única entrada del mismo en el Directorio de Archivos.

En sistemas avazados debe existir la facilidad de tener varios archivos con el mismo nombre y poder llamar al mismo archivo con nombres distintos.

Para implementar esto el Directorio de Archivos se divide en dos partes. Un **Directorio de Archivo Simbólico** (DAS) y un **Directorio de Archivo Básico** (DAB).

El Sistema de Archivo Simbólico debe buscar en el Directorio de Archivo Simbólico la entrada perteneciente al archivo requerido y de esta forma encontrar su único identificador (ID) dentro del sistema, para pasárselo al módulo denominado Sistema de Archivo Básico (BAS).

En el ejemplo anterior se devolverá 6, donde éste es el ID del archivo "PEPE".

Usualmente lo que se hace es copiar en memoria las entradas del DAS que corresponden al archivo en uso (llamados archivos activos o archivos abiertos), de tal forma que estas entradas son usadas para evitar E/S reiteradas sobre la misma zona en el periférico. Esta tabla se mantiene en memoria principal con las entradas del DAS correspondientes a archivos abiertos, denominándose **Tabla de Nombres Activos** (TNA).

El módulo SAS es responsable de:

- El Directorio de Archivo Simbólico.
- La Tabla de Nombres Activos.
- La comunicación con el módulo SAB.

4.3.1.3 Sistema de Archivo Básico

El segundo módulo que es llamado en la secuencia se denomina Sistema de Archivo Básico (SAB).

SAB(función, ID de archivo, número de registro, localización)

En el ejemplo:

SAB(LEER, 6, 5, 1200)

Todos los parámetros son iguales a la llamada del módulo SAS, a excepción del segundo parámetro que constituye el identificador que le pasa el SAS.

El SAB se vale del identificador del archivo (ID) para la localización de la entrada correspondiente en el Directorio de Archivo Básico.

Dicha entrada es mantenida en memoria principal con el objeto de ahorrar posteriores E/S buscando la misma entrada. La tabla que contiene las entradas del Directorio de Archivo Básico de los archivos abiertos recibe el nombre de **Tabla de Archivos Activos** (TAA). Esta tabla se genera y mantiene

en memoria principal a diferencia del directorio básico (DAB) y del directorio simbólico (DAS). Su entrada consta de la siguiente información:

Identificación del archivo

Longitud de registro lógico

Longitud de registro físico

Formato

Organización

Permisos

Concurrencia

Lista de procesos que lo usan

En la próxima etapa Verificación de Control de Acceso (VCA) se usa en lugar del ID del archivo, su entrada en la TAA, la que contiene la información del archivo cuyo ID es 6.

El módulo SAB se encarga de :

- El Directorio de Archivo Básico.
- La Tabla de Archivos Activos.
- La comunicación con el módulo VCA.

4.3.1.4 Verificación de Control de Acceso

Este módulo es invocado de la siguiente forma:

VCA(función, entrada de TAA, número de registro, localización)

En el ejemplo:

VCA(LEER, 2, 5, 1200)

Donde el segundo parámetro es la segunda entrada de la TAA que corresponde al archivo PEPE y los demás son los mismos parámetros de la llamada al módulo anterior (SAB).

Este módulo actúa como control entre el Sistema de Archivo Básico y el Módulo de Sistema de Archivo Lógico. Verifica si la función que se quiere realizar sobre el archivo está indicada en la entrada a la Tabla de Archivos Activos.

Si no está permitido realizar la operación se produce un error, y el pedido es abortado. Si por el contrario, se puede realizar la operación el control pasa directamente al módulo de Sistema de Archivo Lógico.

4.3.1.5 Sistema de Archivo Lógico

Una llamada a este módulo (SAL), tiene la misma sintaxis que la llamada al módulo VCA.

SAL(función, entrada de TAA, número de registro, localización)

En el ejemplo:

SAL(LEER, 2, 5, 1200)

El Sistema de Archivo Lógico convierte el pedido de un registro lógico en el pedido de una secuencia de bytes lógicos, la cual se entrega al Sistema de Archivo Físico (SAF).

Para el SAF un archivo es una secuencia de bytes sin ningún tipo de formato.

Para nuestro ejemplo, vamos a suponer un formato de registro lógico de longitud fija, la información que necesitamos es tomada de la entrada en la tabla TAA.

> **Dirección del byte lógico (dbl)** = (número de registro - 1) * longitud del registro lógico = 4 * 500 = 2000
>
> **Longitud de secuencia de byte lógicos (lsb)** = longitud del registro lógico = 500

Teniendo ya calculado el comienzo de la secuencia de bytes y su longitud, el SAL invoca al módulo de *Sistema* de Archivo Físico (SAF):

SAF(LEER, 2, 2000, 500, 1200) donde dbl = 2000 y lsb = 500

El tecer y cuarto parámetro corresponden al dbl y lsb y el resto de los mimos son iguales que los de la llamada anterior.

Resumiendo, el Sistema de Archivo Lógico es responsable de:

- Convertir la solicitud de un registro a solicitud de secuencia de bytes.

- Comunicarse con el Sistema de Archivo Físico.

4.3.1.6 Sistema de Archivo Físico

El Sistema de Archivo Físico (SAF) tiene por función determinar en qué bloque físico del dispositivo de almacenamiento se encuentra la secuencia de bytes lógicos pasados por el SAL, para lo cual usa la entrada en la tabla TAA más el dbl y el lsb, calculados por el módulo anterior.

Una llamada típica es

SAF(función, entrada de TAA, dirección de bytes, longitud de bytes, localización)

El bloque es entonces leído y colocado en un buffer preasignado en memoria principal. A partir de este buffer es extraída la secuencia de bytes pedidos y colocados en el área del buffer del usuario.

Para realizar estos cálculos el SAF debe saber las funciones de mapeo y longitud del bloque físico que utiliza cada dispositivo de almacenamiento. Si éstas fueran iguales para todo tipo de periférico podrían estar dentro de las mismas rutinas del SAF, pero tal cosa no sucede pues existen diferencias de un periférico a otro.

Esto se resuelve manteniendo tal información en el mismo volumen de almacenamiento, de tal forma que cuando se inicializa el sistema toda esta información puede ser leída en una entrada de la Tabla de Archivos Activos.

Si se hace un pedido de escritura y el bloque sobre el cual se quiere escribir no está asignado previamente entonces el Sistema de Archivo Físico invoca al Módulo de Estrategia de Asignación (MEA) para que este le proporcione un bloque libre en memoria secundaria sobre el cual pueda efectuar la escritura.

Resumiendo, el Sistema de Archivo Físico es responsable de:

- Conversión de la solicitud de secuencia de bytes lógicos a número de bloque físico y desplazamiento.

- Comunicación con el módulo de Estragia de Asignación y el módulo de Estrategia del Dispositivo.

4.3.1.7 Módulo de Estrategia de Asignación

Se encarga de llevar un registro del espacio libre disponible en el almacenamiento secundario. Esta información aparecerá reflejada en la TAA.

4.3.1.8 Módulo de Estrategia de Periférico

Este módulo convierte el número de bloque físico en el formato adecuado para el periférico en cuestión (por ejemplo bloque 7 = pista 3, sector 23). Además, inicializa los comandos adecuados de E/S para el tipo de periférico sobre el cual se va a realizar la operación.

Todos los módulos anteriores son totalmente independientes de cualquier tipo de periférico con excepción de este último. De aquí en adelante el control pasa a manos del Administrador de E/S.

4.3.1.9 Resumen de Módulos

Sistema de Archivos Simbólicos (SAS)	Trasforma el nombre del archivo en el identificador único del Directorio de archivos. Utiliza la Tabla de Nombres Activos (TNA) y el Directorio de Archivos Simbólico (DAS).
Sistema de Archivos Básico (SAB)	Copia la entrada de la VTOC en memoria. Utiliza el Directorio de Archivo Básico (DAB) y la Tabla de Archivos Activos (TAA).
Verificación de Control de Acceso (VCA)	Transforma el pedido lógico en un hilo de bytes lógicos.
Sistema de Archivo Físico (SAF)	Calcula la dirección física.
Módulo de Estrategia de Asignación (MEA)	Consigue espacio disponible en el periférico (en caso de grabación).
Módulo de Estrategia de Periférico	Transforma la dirección física según las características exactas del periférico requerido.

4.3.2 Implementación de Directorios

Al hablar del Sistema de Archivo Básico dijimos que opera sobre el Directorio de Archivo Básico (DAB). Usualmente este directorio se encuentra almacenado en algún periférico del sistema que posee alta velocidad.

Es conveniente estructurar los directorios en forma jerárquica para proveer una mejor y más óptima forma de acceso.

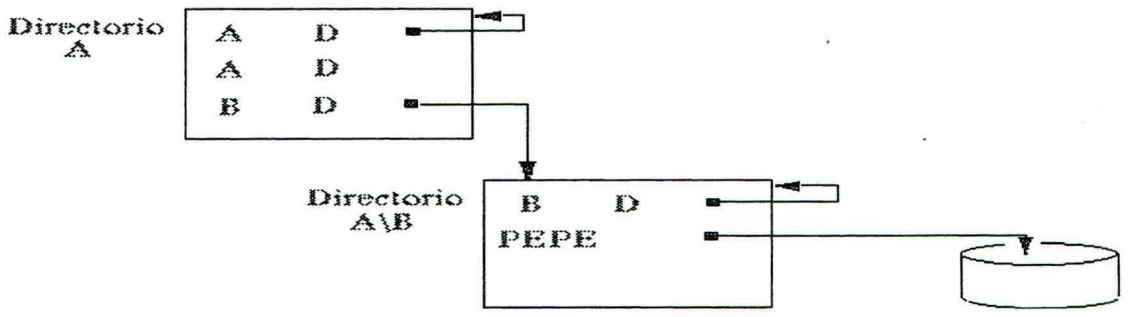


Fig. 4.4

En este caso la cantidad de accesos para llegar al archivo dependerá de la profundidad de niveles del directorio.

En forma genérica se puede decir que para llegar a una entrada en el nivel **n** se requerirán no menos de **n+1** accesos a disco, debido a que es necesario siempre recorrer los **n** niveles que apuntan a dicho archivo para finalmente acceder al archivo en sí.

En el ejemplo para llegar al archivo PEPE se deberá ingresar al sistema por el directorio A, de allí pasar al segundo nivel en el directorio AB y de allí acceder al archivo.

En las estructuras de este tipo es necesario tener presente que en las entradas de los directorios debe existir algún campo que permita diferenciar si la información contenida en esa entrada corresponde a un directorio o a un archivo (en el ejemplo la entrada correspondiente al directorio AB y la entrada correspondiente al archivo PEPE).

En MS-DOS la entrada en el directorio es de 32 bytes y contiene la siguiente información: nombre de archivo, atributos, y el número del primer bloque físico. El primer número de bloque es usado como un índice de una tabla del tipo al de la figura 4.5, que encadena todos los bloques que componen el archivo.

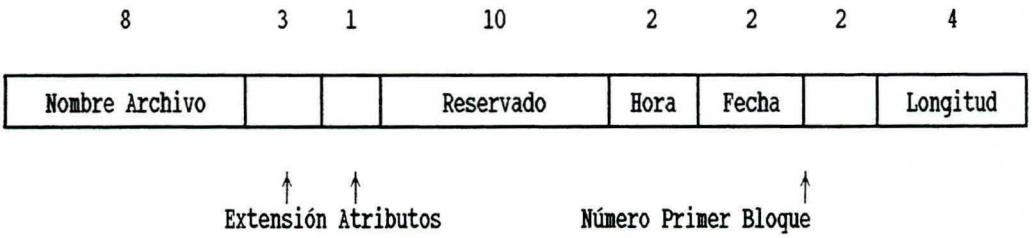


Fig. 4.5 Entrada en el Directorio MS-DOS

En MS-DOS los directorios pueden contener otros directorios, permitiendo de esta forma tener sistema de archivo de estructura jerárquica.

4.3.3 Estructuras de Archivos

Existen tres posibilidades para estructurar los archivos.

En MS-DOS un archivo es una secuencia de bytes sin estructura, en donde el sistema operativo no conoce el contenido del archivo, para él sólo son bytes.

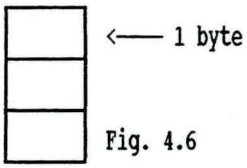


Fig. 4.6

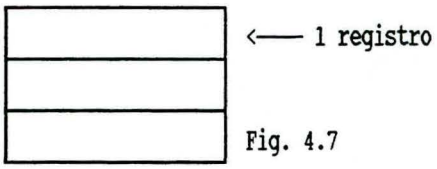


Fig. 4.7

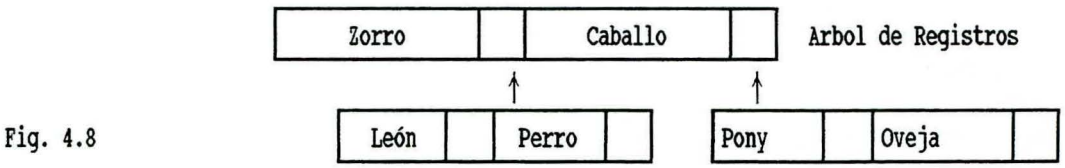


Fig. 4.8

Este esquema provee el máximo de flexibilidad porque para el sistema operativo los archivos no son otra cosa mas que una secuencia de bytes. Los usuarios de programas pueden poner lo que desean en los archivos, con el nombre que crean conveniente.

El primer paso en estructura de archivos está dado por la figura 4.7. En este modelo un archivo es una secuencia de registros de longitud fija. La idea central es que las operaciones de lectura-escritura se hacen sobre un registro.

La tercera clase de estructura de archivos consiste en un árbol de registros (figura 4.8), no necesariamente de la misma longitud, cada uno contiene un campo clave en una posición fija del mismo. El árbol es ordenado por el campo clave para permitir el acceso rápido a un registro en particular.

La operación básica aqui no es obtener el próximo registro, aunque también esto es posible, sino obtener el registro con la clave específica. Por ejemplo, en el archivo "zoo", se puede pedir al sistema el registro cuya clave es "pony" sin saber su posición exacta en el archivo. Además nuevos registros pueden ser sumados al archivo, sin que el usuario decida dónde ponerlo, ya que el sistema operativo se encarga. Esta organización es usada en grandes computadoras mainframe.

El Sistema de Achivo Lógico mapea la estructura de los registros lógicos a una secuencia de bytes. En especial debe convertir una solicitud para un registro a una solicitud para una secuencia de bytes, para lograr esto debe tener conocimiento de la estructura de archivo que se está usando.

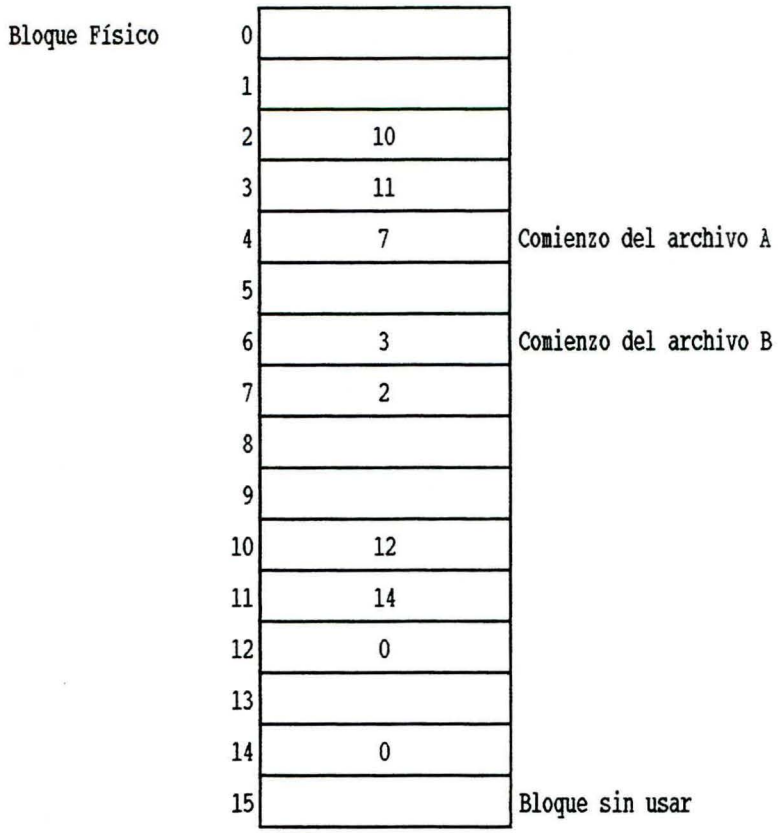
4.3.4 Implementación de Archivos

La clave en la implementación del almacenamiento de archivos es el mantenimiento de la información relativa a los bloques del disco en que reside. Varios métodos son usados en diferentes sistemas, en esta sección se detallará el método utilizado por MS-DOS.

MS-DOS asigna bloques no contigÜos en el disco y mantiene una tabla o índice en memoria con punteros a cada bloque del disco.

Usando la tabla de la figura, podemos comenzar a recorrer el archivo A con el bloque 4, el segundo bloque es el 7, el tercero es el 2 y así seguir el encadenamiento hasta llegar al bloque 12 en donde el archivo finaliza. De la misma forma se puede acceder al archivo B.

Lista de almacenamiento usando una tabla en memoria



El uso de esta organización permite que un bloque entero esté disponible para almacenar datos de un archivo y que todos los bloques del disco sean usados; evita la fragmentación del disco. Con la asignación de bloques no contigÜos es fácil aumentar dinámicamente el tamaño de un archivo durante la ejecución de un programa.

El acceso al azar requiere del seguimiento de la cadena de bloques que lo componen para hallar la porción del archivo buscada, el encadenamiento está completo en la memoria, por lo tanto puede ser seguido sin realizar lecturas de referencia al disco.

Es suficiente guardar en la entrada de directorio un entero que referencia al número del primer bloque.

La desventaja principal de este método es que la tabla entera debe permanecer en memoria todo el tiempo en que se esté trabajando. Con un disco de 500.000 bloques de un 1K (500M), la tabla tendría 500.000 entradas, cada una de las cuales necesita como mínimo de 3 bytes a 4 bytes. Por lo tanto, ocuparía 1,5 o 2 megabytes todo el tiempo.

El Sistema de Archivo Físico se basa en esta implementación para convertir las direccines de bytes lógicos a direcciones de bloques físicos.

4.3.5 Estructuras de Control de Acceso

El control de acceso de un archivo está dado por una serie de permisos que son controlados por el módulo de Verificación de Control de Acceso (VCA).

Tales permisos se almacenan conjuntamente con la información del directorio básico o alguno de sus niveles y pueden constituir, asimismo, archivos de permisos. Estos datos serán copiados por el Sistema de Archivo Básico en la TAA y consultados por el VCA para autorizar el acceso.

Una vez liberado el archivo y, en caso de haber ocurrido algún cambio en los permisos, la información deberá ser devuelta a su lugar de almacenamiento en memoria secundaria.

MS-DOS no guarda información acerca de permisos de acceso, por lo tanto este módulo no existe en él.

4.3.6 Estrategia de Asignación

Existen varias técnicas para la asignación de bloques libres del disco.

Una de ellas es la llamada **Mapa de Archivos Libres**, en la cual los espacios libres se tratan como archivos que poseen entradas en la tabla de directorios. Esta técnica trabaja bien cuando existen grandes áreas libres. De haber muchas áreas libres pequeñas el directorio crece y se torna ineficiente.

Una solución a este problema es utilizar un mapa especial de archivos que contenga una lista de todos los bloques disponibles. Siempre que deba asignarse un bloque se saca de esta lista. Siempre que se libere uno o más bloques (por ejemplo, que se borre un archivo) se agregan estos bloques al mapa de archivos libres.

Otro enfoque de asignación, **cadena libre**, es mantener una lista encadenada de bloques libres, donde cada bloque libre tenga un puntero al siguiente bloque libre. En este caso no se realizan búsquedas. Cuando se necesita un bloque, se toma el primer bloque de la cadena. Recíprocamente, cuando se libera un bloque se coloca al principio de la cadena.

Otro método popular utiliza una sucesión de bits, llamada **mapa de bits**, para indicar el estado de cada uno de los bloques. El 1 indica que el bloque correspondiente está usado y el 0 significa que el bloque está libre. Su principal ventaja es que se puede mantener en memoria debido a que puede ser pequeño. De esta manera es posible efectuar la asignación a gran velocidad.

11100000110011

4.3.7 Estrategia de Dispositivo

Las funciones básicas de estas rutinas son las siguientes:

1- Mapear la dirección física a una dirección del dispositivo (por ejemplo, la dirección física del

registro 2 del archivo PEPE, es el bloque físico 5, que es la dirección: pista 1, registro 3 del dispositivo.

2- Crear el programa de canal, o sea un programa que haga:

a- Bucar la pista 1.

b- Buscar en la pista 1 el registro 3.

c- Esperar hasta que el disco lo encuentre.

d- Leer.

3- Solicitar la E/S, es decir, llamar al administrador de E/S para que la programe.

4- Administrar las interrupciones y condiciones de error de E/S que puedan ocurrir.

5
PROGRAMAS RESIDENTES

5.1 Introducción

Un programa se dice residente cuando es cargado en memoria para su ejecución y una vez que esta finaliza se mantiene en memoria sin liberar toda el área que le fuera asignada.

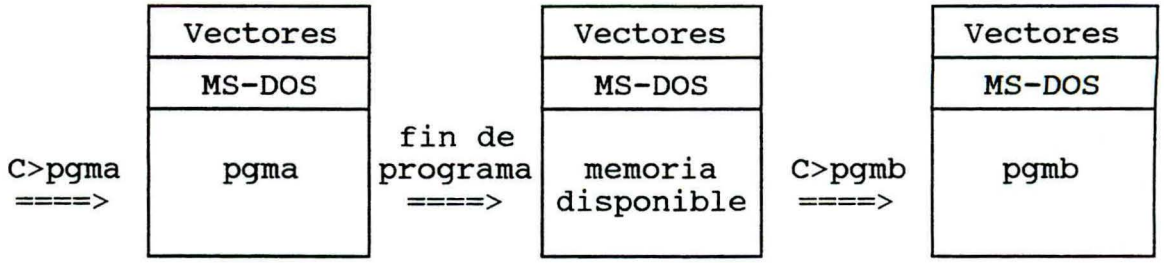
Los términos **TSR** y **Pop Up** significan la misma cosa. Cuando un programa está ejecutando se convierte en una extensión del DOS. Este constantemente monitorea el teclado esperando que la tecla presionada sea la misma combinación de teclas declaradas en el programa. Esta combinación de teclas llamadas "hotkey" es la señal para que el programa salte a la vida (pop up).

El término TSR proviene del hecho que cuando se invoca desde el prompt del DOS el programa puede terminar como un programa normal o puede permanecer residente en memoria (Terminar and Stay Resident)

La diferencia entre un programa llamado **Pop Up** y uno **background** es fundamental. El programa Pop Up es activado solamente cuando el usuario lo invoca con la combinación **hotkey**, mientras que un programa background es invocado por el procesador aproximadamente 18 veces por segundo

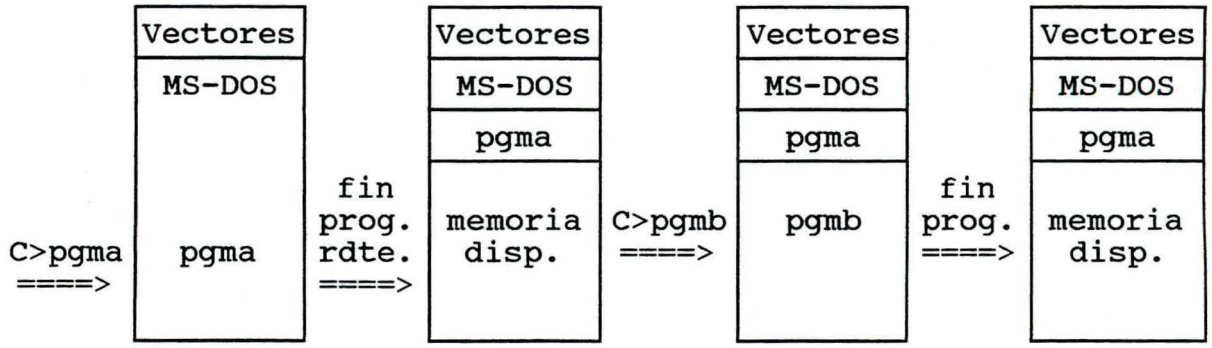
5.2 Memoria reservada

Cuando un programa termina su ejecución, la memoria es liberada y queda disponible para cargar un nuevo programa. Veamos un ejemplo con dos programas (pgma y pgmb) que son invocados en forma sucesiva.



Para indicar que un programa quedará residente en memoria se utiliza una función especial, pasándole como parámetros el código de retorno y la cantidad de memoria que el programa se reserva. El siguiente programa a ejecutar tendrá como memoria disponible, la que deje libre el programa que queda residente.

Veamos el ejemplo anterior, suponiendo que el programa pgma queda residente en memoria:



Se pueden cargar tantos programas residentes como se quiera, en la medida en que se tenga suficiente memoria para que se puedan ejecutar el resto de los programas que se invoquen.

5.3 Cómo trabaja un TSR

Cuando se ingresa el nombre de un programa TSR en el prompt del DOS éste lo trata como un programa cualquiera. Claro, a esta altura eso es lo que es. DOS ubica memoria para éste y carga el archivo .exe o .com. Pasando luego el control a la primera instrucción en el programa.

Como DOS no es un sistema operativo multitarea, asigna todos los recursos del computador al programa.

Cuando el programa decide convertirse en un TSR, abandona un mundo seguro, controlado por DOS, cambiándolo por un mundo hostil en el cual cada programa está por sí mismo. Por lo tanto deber hacer ciertas cosas para protegerse.

Primero debe interceptar ciertas interrupciones. Por ejemplo, si se desea monitorear el teclado, para chequear por una combinación de teclas, debería capturar la interrupción 9h. La interrupción 9h es un evento que ocurre cada vez que se presiona o libera una tecla. Si se desea tener la ventaja de programar backgroumd, se debe capturar también la interrupción 28h.

5.4 Qué es una interrupción

En una PC hay 256 interrupciones (de 0 a 255). Cada interrupción tiene asociado un vector en memoria. Un vector consiste de cuatro bytes que contienen la dirección de una rutina que se ejecuta cuando la interrupción asociada se produce. Así cuando se presiona una tecla, una interrupción 9h se genera. Como resultado la CPU congela cualquier cosa que esté haciendo, levanta la dirección que se encuentra en el vector asociado y pasa el control a esta rutina. Cuando este llamado de rutina retorna, el procesador continua donde había dejado.

Capturar una interrupción consiste en colocar la dirección del programa en el vector de interrupción. Si se captura la interrupción 9h, por ejemplo, y se apunta ésta a una sección del programa, el mismo automáticamente será llamado cada vez que alguien presione una tecla. Si el programa permite removerse a sí mismo, obviamente tiene primero que guardar el contenido del vector, así podrá retornarlo a su estado original.

Lo próximo que un TSR debe hacer es decidir la cantidad mínima de memoria que requiere. Una vez que esto se establece, puede llamar al DOS para terminar y dejar su imagen en memoria. Cuando esto se hace, DOS termina el programa y retorna al prompt. La memoria requerida por el TSR será reservada, y ningún programa subsiguiente podrá cargarse en ella.

A esta altura el programa está en un estado de animación suspendida. Su imagen está en memoria pero no hace nada. Recién cuando la interrupción que ha sido capturada ocurre, el programa se activa.

La interrupción 8h se invoca automáticamente 18 veces por segundo. De este modo, si un TSR también captura esta interrupción, las rutinas seleccionadas también serán activadas 18 veces por segundo. Por lo tanto, si un TSR se escribe pensando en esto, puede convertirse en un programa multi-tarea.

5.5 Qué puede o no puede hacer un TSR

Cuando se da el control a un TSR vía una de las interrupciones capturadas, éste no puede hacer otra cosa hasta que chequé si el MS-DOS y el BIOS están estables. Recordar que MS-DOS no es multi-tarea, así no es sorprendente que cuando ciertas partes de él fueron escritas no se hicieron reentrantes. En otras palabras si MS-DOS está haciendo algo y un TSR se activa y le pide hacer algo más, lo más probable es que se congele el computador.

Esto se resuelve si el programa TSR monitorea ciertos eventos. Por ejemplo, monitorear cualquier código no reentrante y no aceptar que se active cuando está activo. El TSR debe monitorear tales cosas como acceso al disco y operaciones de MS-DOS para detectar si ellas están en uso.

El algoritmo observa algunas cosas como éstas:

1- El TSR ya está activo? ... Si es así terminar

   sino

2- El servicio de disco está ocupado? ... Si es así terminar

   sino

3- Una aplicación gráfica está ejecutando? ... Si es así terminar

sino

4- El DOS está ocupado? ... Si, entonces se está dentro del programa... sino terminar.

Esta última condición necesita una pequeña explicación. Casi todos los TSR se encadenan dentro de un programa, interrupción 28h. Esta es la interrupción usada por programas tales como PRINT. Siempre que DOS se encuentra libre dispara una interrupción 28h. Cuando esto sucede, algún programa que está encadenado a la interrupción 28h tiene la oportunidad de hacer algún proceso.

6
TRANSFERENCIA DE INFORMACION ENTRE ESTACIONES

6.1 Alcances

Para lograr el objetivo establecido se fijan a continuación las bases sobre las que se desarrollará el trabajo de tesis.

Nuevos Comandos

Los comandos que se suman a los ya existentes en NetWare son:

- **PDIR** : Visualiza la información acerca de archivos y directorios del disco de otra estación de trabajo.

- **PCOPIA** : Copia archivos desde el disco local de la estación de trabajo origen al disco local de la estación de trabajo destino o viceversa.

Para el desarrollo de los mismos se considerará el comportamiento de los comandos del sistema operativo DOS: DIR y COPY.

El comando PDIR deberá mostrar por pantalla la siguiente información:

- Nombre del nodo fuente.

- Solicitud realizada de directorio y/o archivos.

- Nombre de directorios y/o archivos que integran el directorio requerido.

- Fecha y hora de creación o actualización según corresponda.

- Tamaño en bytes de cada archivo.

- Total de archivos mostrados.

- Total de bytes del conjunto de archivos visualizados.

Podrán elegirse opciones para ver la información en forma encolumnada y/o paginada.

Podrán ser utilizados los caracteres wildcards ('*', '?') para comodidad del usuario.

Deberán ser respetados los atributos de los archivos (ocultos y del sistema).

El comando PCOPIA copia un archivo desde el disco local de la estación fuente respetando sus atributos (sólo lectura, ocultos y del sistema), a la estación destino.

No reconocerá caracteres wildcards.

Recursos

Los medios necesarios para el desarrollo son:

- Sistema operativo DOS.

- Red Novell NetWare.

- Protocolos de comunicación IPX/SPX.

Protocolo de Comunicacion

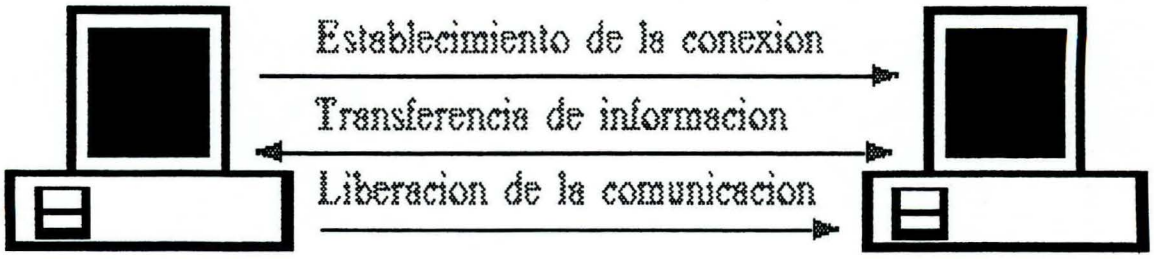
Debido a que se transmitirán archivos entre dos estaciones es necesario contar con un protocolo confiable que garantice la llegada de los paquetes a destino sin errores ni duplicados y en la secuencia correcta. En virtud de lo dicho el desarrollo se basará en el protocolo SPX que cumple con estos requerimientos.

Mediante el empleo de las APIs de éste protocolo se logrará una comunicación punto a punto entre estaciones, no siendo necesario conectarse con el servidor para poder enviar paquetes entre éstas.

Con cada comando se deberá:

- Establecer la comunicación entre la estación de trabajo origen y destino.
- Transferir la información.
- Liberar la comunicación.

Como el usuario se limitará a ejecutar los comandos que se pondrán a su servicio dicha comunicación es transparente para él.



6.2 Ideas Globales de Diseño

6.2.1 Aplicaciones

Desde cada estación de trabajo se podrá ejecutar cualquiera de los comandos mencionados anteriormente.

Cuando un usuario en una estación desea, por ejemplo realizar un PDIR, deberá ingresar el nombre del comando y los parámetros necesarios, en el ejemplo el nodo y directorio fuente. Esta aplicación deberá conectarse con otra residente en la estación del nodo fuente (TSRCOM), quien le enviará la información que se le solicita, en el ejemplo, los archivos y directorios que integran el directorio fuente.

La aplicación que responde al pedido deberá permanecer residente en memoria a la espera de una solicitud de conexión. En cuanto ésta se produce analiza el pedido recibido, ya sea PDIR o PCOPIA y envía la información correspondiente.

Toda estación que necesite la prestación de estos servicios deberá contar con la aplicaciones:

- PDIR.
- PCOPIA.
- TSRCOM.

Las dos primeras corresponden a los nuevos comandos (clientes) y la tercera será la prestataria de los servicios requeridos por las anteriores (servidor).

6.2.2 Reconocimiento de Estaciones

En Novell cada estación de trabajo tiene una dirección de red y una única dirección de nodo que permiten su reconocimiento en forma unívoca.

Para evitar al usuario recordar estas direcciones se crea un archivo de mapeo donde se asocia un nombre a la dirección de red y nodo de cada una. De esta manera, el usuario sólo deberá memorizar el nombre que le fue asignado.

Este nombre se utilizará como parámetro de los comandos.

6.2.3 Sockets

Las aplicaciones correspondientes a los comandos usan un mismo número de socket y éste es dinámico (se cierra cuando termina la aplicación). En cambio, la aplicación residente usa un número de socket distinto, siendo éste permanente (permanece abierto al cerrarse la conexión).

Si todas las aplicaciones usaran el mismo número de socket, los comandos al finalizar su ejecución lo cerrarían, y el residente se quedaría escuchando en un socket cerrado.

6.2.4 Tipos de Paquetes

Se hace indispensable la creación de tres tipos de paquetes:

- Pedido
- Dato
- Error

El **paquete de pedido** es empleado por la aplicación residente para reconocer que la información que tiene es la de un comando.

El **paquete de dato** es destinado a la transmisión de la información acerca de archivos o directorios.

El **paquete de error** indica que se ha producido un error en la otra estación y puede contener un mensaje.

6.3 Diseño

6.3.1 MAPA.TXT

La facilidad en el reconocimiento de las estaciones se logra a través del archivo llamado **mapa.txt**. En él se asocia el nombre dado a la estación con sus direcciones de red y nodo.

Es un archivo ASCII, compuesto por:

Nombre de Nodo	Dirección de Red	Dirección de Nodo

La **dirección de red** deberá corresponder con el número de red sobre la cual el nodo reside. Este número será de 8 dígitos hexadecimales (4 bytes). Si la red tuviera un número con menos dígitos se deberá completar con ceros a la derecha.

El **nombre de nodo** estará compuesto por caracteres alfanuméricos.

La **dirección de nodo** deberá corresponder con la dirección física del nodo. Este número será de 12 dígitos hexadecimales (6 bytes). Si el nodo tuviera un número con menos dígitos se deberá completar con ceros a la derecha.

Cada uno de los componentes deberá estar separado por un caracter blanco, ' ', y los distintos nodos estarán separados por un fin de línea.

Ejemplo de mapa.txt:

```
NODO1 08B10002 0020AF4D3811
NODO2 08B10002 0020AF4D3765
```

Las aplicaciones que utilizan este mapa son PCOPICA y PDIR.

En todas las computadoras personales en donde sean instalados estos comandos deberá instalarse también este archivo.

6.3.2 Recepción de paquetes

Cuando una aplicación quiere establecer una conexión, ya sea porque envía un paquete de "establecer conexión" (API **SPXEstablishConnection**) o porque pone a escuchar un paquete "escuchar establecer conexión" (API **SPXListenForConnection**), debe crear al menos dos paquetes que escuchen ECBs y sean pasados a SPX por medio de **SPXListenForConnection**.

La cantidad de paquetes que se ponen a escuchar establecerá la máxima cantidad de paquetes que SPX podrá recibir simultáneamente (llamada ventana). Cada aplicación tendrá una constante **VENTANA** que indicará dicha cantidad.

Las aplicaciones PCOPIA y PDIR establecen una conexión con la aplicación TSRCOM. Cada una de ellas tendrá una ventana de paquetes que estarán escuchando; se la puede ver como una ventana receptora, correspondiente a los paquetes que está autorizada a recibir. Por lo tanto, cada aplicación estará limitada a enviar la cantidad de paquetes que el otro extremo pueda recibir. Por ejemplo, si la ventana de PCOPIA es de 5 paquetes, TSRCOM sólo le podrá enviar 5 paquetes; y si a su vez la ventana de TSRCOM es de 3 paquetes, PCOPIA sólo le podrá enviar 3 paquetes. La ventana del emisor y la del receptor no necesitan tener la misma cantidad de paquetes.

Los paquetes se pondrán a escuchar de la siguiente forma:

- Se aloca memoria para cada paquete que se pondrá a escuchar (ECB, fragmento de cabecera y datos del paquete).

- Se ponen a escuchar los paquetes generados por medio de **SPXListenForSequencedPacket()** en el socket de la aplicación. SPX suma el ECB recibido a un pool de buffers propio de ECBs que están escuchando paquetes en secuencia sobre el mismo socket.

Las preguntas que nos hacemos son ¿cómo sabe la aplicación que le llegó un paquete? y ¿dónde lo guarda?

Cuando llega un paquete SPX, el protocolo usa uno de los ECBs que están escuchando para recibirlo. Es registrado el valor apropiado en el campo **CompletionCode**, el campo **InUseFlag**

toma el valor 00h (disponible para uso) y se llama a la rutina del servicio de eventos referenciada por el campo del **ECB** llamado **ESRAddress.**

Usamos una rutina de servicio de evento (ESR) para administrar los paquetes que son recibidos, la misma se llamará **ReceiveESR().** Por lo tanto cuando se inicializan los paquetes para escuchar, el campo del **ESRAddress** perteneciente al ECB tomará la dirección de la rutina **ReceiveESR().**

Las aplicaciones para manejar estos paquetes crearán:

Un array de punteros a ECBs (**RcvECB**), que tendrá tantos elementos como indique la ventana. Estos punteros se usarán para apuntar a los ECBs recibidos.

Existirán dos índices para recorrer el array :

- Un índice ,**lRcvECB**, al siguiente elemento del **RcvECB** donde se guardará la dirección del próximo paquete a recibir.

- Un índice, **iRcvECB**, al elemento del **RcvECB** que apunta al paquete a analizar.

La variable, **cant_rcv**, indicará la cantidad de paquetes recibidos que están en el **RcvECB.**

La función **ReceiveESR** (rutina del servicio de evento) guardará el puntero al paquete recibido **ESR_ECB** en el elemento **RcvECB** indicado por **lRcvECB** y aumentará la cantidad de paquetes recibidos (cant_rcv).

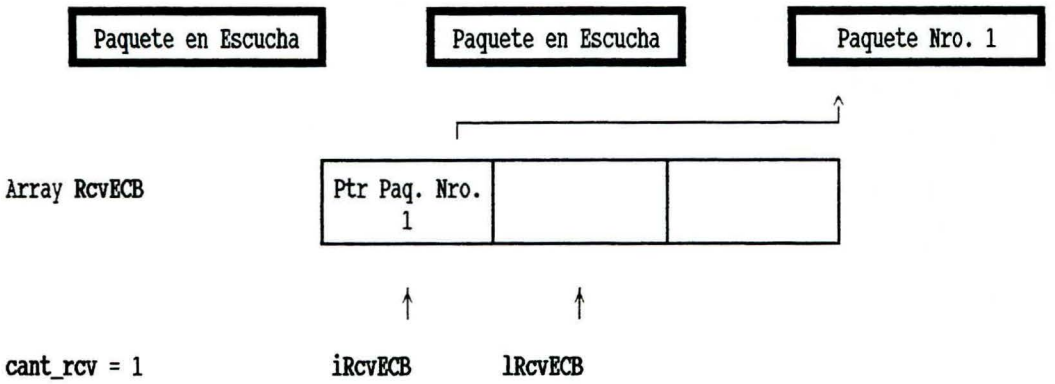
Cuando el índice **lRcvECB** llega al límite de la ventana vuelve a comenzar en la primera posición del array **RcvECB.**

El índice iRcvECB será utilizado por las aplicaciones para saber cuál es el paquete que tiene que analizar. Una vez que lo analiza lo pone nuevamente a escuchar, **SPXListenForSequendPacked().** Cuando llega al final de la ventana vuelve a comenzar.

La variable **cant_rcv** será empleada por las aplicaciones para saber si le llegó algún paquete, o visto de otra forma, si tiene paquetes para analizar.

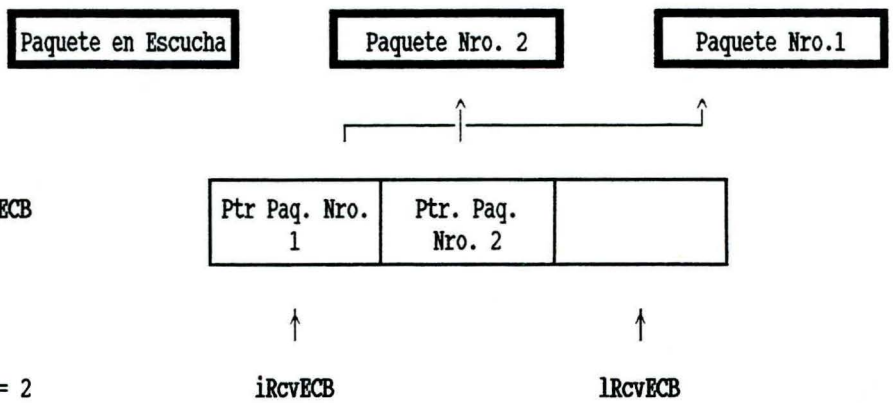
Supongamos que tenemos una VENTANA = 3

Se reservará memoria para 3 paquetes:

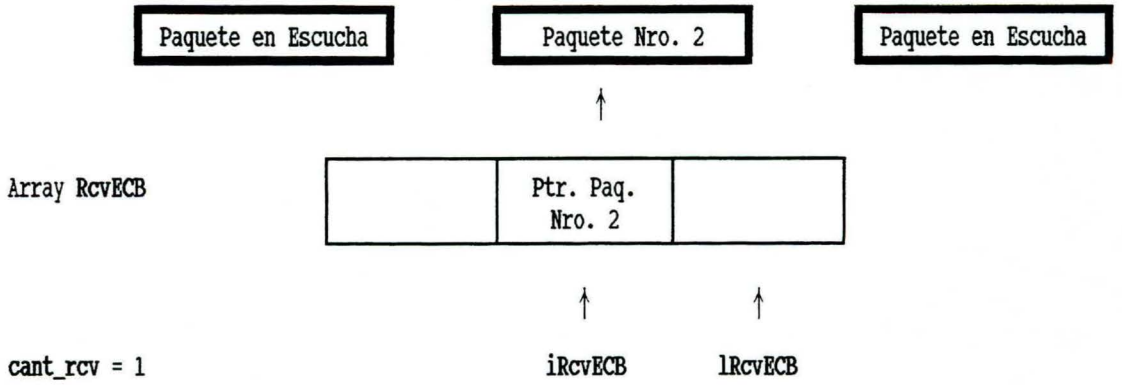


En el ejemplo, cant_rcv = 1, por lo tanto se recibió un paquete. Este paquete está apuntado por el elemento iRcvECB del array RcvECB (aún no fue analizado).

Cuando se recibe otro paquete la función **ReceiveESR()** hace que el elemento lRcvECB de RcvECB apunte al paquete que acaba de llegar. Luego lRcvECB pasa a apuntar al próximo elemento. También la variable cant_rcv es incrementada en uno para anunciar que llegó un paquete, por consiguiente ahora existen dos paquetes que se deben analizar.



Cuando se analiza el paquete apuntado por RcvECB[iRcvECB] (paquete nro. 1) se llama a la función **SPXListenForSequencedPacket()** para poner el paquete a escuchar nuevamente. También se incrementa el índice iRcvECB que apuntará al paquete nro. 2 y se decrementa la variable cant_rcv.



6.3.3 Tipos de Paquetes

Como ya se ha dicho, fue indispensable crear tres tipos de paquetes de acuerdo a la información que se debe enviar y recibir.

El campo **DataStreamType** de la cabecera del paquete SPX indica el tipo de datos que se encuentra en cada paquete. Los valores definidos para ellos son los siguientes:

0x1 Pedido

0x2 Error

0x4 Datos

Pedido

Este paquete contiene el comando que se desea procesar.

El campo **FragmentDescriptor[1].Adrress** del ECB tendrá la dirección de una estructura de tipo COMANDO.

Esta estructura posee dos campos:

comando : tipo de comando a procesar.

origen : directorio/archivos que se desean visualizar, archivo fuente que se desea copiar desde la estación remota o archivo destino que será copiado en la estación remota.

Los tipos permitidos de comandos son los siguientes:

- **PDIR** Mostrará en la pantalla de la estación que ejecuta el comando el directorio/archivos requeridos de la estación remota.

- **RXPCOPIA** Copiará en la estación que ejecuta el comando el archivo que le transmite la estación remota, según el requerimiento que ésta le hiciera.

- **TXPCOPIA** Copiará en la estación remota el archivo que la estación que ejecutó el comando le envía.

Error

Este tipo de paquete se utiliza para avisar a la estación remota que a ocurrido un error al procesar el comando.

Cuando se recibe este tipo de paquete el mensaje apuntado en el campo **FragmentDescriptor[1].Adrress** del ECB se despliega por pantalla para informar el error.

Datos

El tipo de paquete "dato" puede poseer dos estructuras diferentes, dependiendo si lo que contiene es información de un archivo que se copiará, o información de un directorio/archivos que se mostrarán por pantalla.

En el caso que contenga una fracción de un archivo el campo **FragmentDescriptor[1].Adrress** del ECB cotiene la dirección de un array de BYTES. Este array tiene una longitud igual a la máxima permitida para el campo datos en el paquete SPX.

Si lo que contiene es información de un directorio/archivos el campo **FragmentDescriptor[1].Adrress** del ECB apuntará a un array de estructuras. Estas estructuras son de tipo FIND, la cual está definida en **dos.h**.

La estrucutra FIND posee los siguientes campos:

reserved : reservado por DOS.

attribute : atributos del archivo.

time : hora de grabación del archivo.

date : fecha de grabación del archivo.

size : tamaño del archivo.

name : nombre del archivo.

6.3.4 Sintaxis del Comando PDIR

Este programa permite desplegar la información del directorio o archivo deseado que se encuentra almacenada en el disco de una estación remota en la pantalla de la estación local.

La sintáxis del comando PDIR será la siguiente:

PDIR [/p /c] <nodo>:<disco>:\<camino>\<archivo>

Las opciones que se pueden ingresar son:

/p : permite paginar la información mostrada por pantalla.

/c : permite encolumnar la información desplegada por pantalla.

? : despliega la ayuda del comando.

Si no se ingresa ningún argumento también se desplegará el comando.

El argumento que le sigue a las opciones indican la información que se quiere visualizar:

- **nodo** : nombre asignado a la estación remota.

- **<disco>:\<camino>** : es el camino absoluto en donde se encuentra la información a desplegar.

- **<archivo>** : es el nombre del archivo a mostrar.

El nodo es un dato obligatorio, sin él no se sabría en qué estación se encuentra la información. El disco y el camino absoluto también son obligatorios. En cambio el archivo puede ser opcional, pues si no se ingresa se toman todos los archivos (*.*) que se encuentren en el directorio.

Se pueden usar caracteres wildcards '?' o '*' en los archivos.

Se tiene en cuenta que se debe analizar no sólo la sintaxis del comando sino también la de cada uno de los componentes de los argumentos (camino y archivo).

Sólo se podrá visualizar aquellos archivos cuyos atributos lo permitan, es decir, se van a respetar los archivos ocultos y del sistema.

La información se desplegará en forma parecida al comando **"dir"** de DOS.

```
C:\>dir /w

 Volume in drive C is EMILSE
 Volume Serial Number is 1B6C-828C
 Directory of C:\

[DOS]           [VEMM]          [COMPRESS]      [NOR]           [MS]
[UTIL]          [GMOUSE]        [VIRUS]         [ZORTECH]       [CHECKIT]
COMMAND.COM     CONFIG.SYS      AUTOEXEC.BAT    AUTOEXEC.OLD    [CODEVIEW]
CONFIG.BAK      AUTOEXEC.BAK    UTILE.CHT       CONFIG.OLD      [NOVELL]
[BORLANDC]
       21 file(s)      50353 bytes
                     3977216 bytes free
```

```
 Volume in drive C is EMILSE
 Volume Serial Number is 1B6C-828C
 Directory of C:\

DOS            <DIR>      11-08-93   9:51p
VEMM           <DIR>      11-12-93   5:21p
COMPRESS       <DIR>      11-10-93   7:04p
NOR            <DIR>      11-10-93   7:05p
MS             <DIR>      11-10-93   7:10p
UTIL           <DIR>      11-10-93   7:43p
GMOUSE         <DIR>      11-10-93   7:53p
VIRUS          <DIR>      11-12-93   5:59p
ZORTECH        <DIR>      03-26-95  12:03p
CHECKIT        <DIR>      11-14-93  10:51p
COMMAND   COM      47845 04-09-91   5:00a
CONFIG    SYS        102 11-12-95   9:34p
AUTOEXEC  BAT        314 11-12-95   9:30p
AUTOEXEC  OLD        131 11-10-93   7:53p
CODEVIEW       <DIR>      05-21-95   1:47p
CONFIG    BAK        104 11-15-93   8:27p
AUTOEXEC  BAK        118 11-12-93   4:40p
UTILE     CHT       1668 10-13-95   7:05p
DIRW      SCT       2096 11-15-95   7:47p
Press any key to continue . . .
```

6.3.5 Sintaxis del Comando PCOPIA

Permite copiar un archivo desde una estación remota a la estación local y viceversa.

La sintáxis del comando PCOPIA será la siguiente:

"PCOPIA <nodo>:<disco>:\<camino>\<archivo> A <nodo>:<disco>:\<camino>\<archivo>"

La opción que se puede ingresar es:

? : despliega la ayuda del comando.

Si no se ingresa ningún argumento también se desplegará el comando.

Los argumentos que le siguen indican el archivo que se quiere copiar y el lugar en donde se desea copiar.

Se llama fuente al archivo que se quiere copiar. En la misma se indicará:

- **nodo** : nombre asignado a la estación en donde se encuentra el archivo. Es un dato obligatorio.

- **<disco>:\<camino>** : es el camino en donde se encuentra la información a desplegar. Si la copia es desde una estación remota a una estación local el camino debe ser absoluto y obligatorio. Si en cambio, la copia es desde la estación local a una remota el camino no es obligatorio y puede ser relativo al directorio corriente.

- **<archivo>** : es el nombre del archivo a copiar. Este debe ingresarse siempre.

Se llama destino al archivo en que se copiará la fuente. En el mismo se indicará:

- **nodo** : nombre asignado a la estación en donde se quiere copiar la fuente. Es un dato obligatorio.

- **<disco>:\<camino>** : es el camino en donde se encuentra la información a desplegar. Si la copia es desde la estación local a una estación remota el camino debe ser absoluto y obligatorio. Si en cambio, la copia es desde la estación remota a la estación local el camino no es obligatorio y puede ser relativo al directorio corriente.

- **<archivo>** : es el nombre que recibe el archivo copiado. Si no se ingresa se asume que recibe el nombre del archivo fuente.

No se pueden usar caracteres wildcards **'?'** o **'*'** en los archivos.

Se tiene en cuenta que se debe analizar no sólo la sintaxis del comando sino también la de cada uno de los componentes de los argumentos (camino y archivo).

También se tendrá en cuenta si la copia es desde la estación local a la estación remota, que el archivo fuente exista y que no sea un archivo oculto, del sistema o de lectura solamente.

En caso que la copia sea de la estación remota a la estación local, se tendrá en cuenta que el archivo destino no exista, y si existiera con los atributos que permiten sobreescribir le preguntará al usuario si desea o no sobreescribirlo.

Si se tienen problemas cuando se está copiando el archivo, éste deberá borrarse, de manera de volver a la situación anterior a la llamada al comando. De esta forma se respet la integridad del archivo copiado.

Se debe tener en cuenta que los archivos a copiar pueden ser tanto archivos de texto como archivos binarios.

La copia será similar al comando **"copy"** de MS-DOS.

6.3.6 Comando PDIR

El comando PDIR deberá ejecutar las siguientes acciones:

- Verificar que las opciones ingresadas sean correctos.
- Verificar que los argumentos de entrada sean correctos.
- Verificar que los protocolos IPX y SPX de Novell estén instalados.
- Abrir el socket, en forma temporaria (se cierra cuando termina la aplicación).
- Colocar paquetes a escuchar.
- Establecer una conexión.
- Enviar el paquete de "pedido".
- Recibir los paquetes de "datos" y desplegarlos por pantalla según la opción de formato ingresado como argumento. Repetir estas acciones hasta que se recibe un paquete de "terminar conexión".
- En caso en que se produzca algún problema en la estación local abortar la conexión.

- En caso en que se reciba un paquete de "error", desplegar el mensaje contenido en él y abortar la conexión.

6.3.7 Comando PCOPIA

El comando PCOPIA tiene dos acciones posibles:

RXPCOPIA: copia un archivo desde la estación remota a la estación local.

TXPCOPIA: copia un archivo desde la estación local a la estación remota.

El pedido RXPCOPIA deberá ejecutar los siguientes pasos:

- Verificar que los argumentos de entrada sean correctos.
- Abrir el archivo destino.
- Verificar que los protocolos IPX y SPX de Novell estén instalados.
- Abrir el socket, en forma temporaria (se cierra cuando termina la aplicación).
- Colocar paquetes a escuchar.
- Establecer una conexión.
- Enviar el paquete de "pedido".
- Recibir los paquetes de "datos" y copiarlos en el archivo destino. Repetir estos pasos hasta que se recibe un paquete de "terminar conexión".
- En caso de que se reciba un paquete de "error", desplegar el mensaje contenido en él y abortar la conexión.
- En caso en que se produzca algún problema en la estación local abortar la conexión.

- En todo momento controlar el ingreso de Ctrl-Break, en este caso abortar la conexión.

El pedido TXPCOPIA deberá ejecutar los siguientes pasos:

- Verificar que los argumentos de entrada sean correctos.
- Abrir el archivo fuente.
- Verificar que los protocolos IPX y SPX de Novell estén instalados.
- Abrir el socket, en forma temporaria (se cierra cuando termina la aplicación).
- Colocar paquetes a escuchar.
- Establecer una conexión.
- Enviar el paquete de "pedido".
- Armar el paquete de "datos" con un fragmento del archivo fuente y enviarlo a la estación remota. Repetir estos pasos hasta que se termine el archivo y luego enviar un paquete "terminar conexión".
- En caso de que se reciba un paquete de "error", desplegar el mensaje contenido en él y abortar la conexión.
- En caso en que se produzca algún problema en la estación local enviar un mensaje de error a la estación remota y abortar la conexión.
- En todo momento controlar el ingreso de *Ctrl-Break*, en este caso abortar la conexión.

6.3.8 Aplicacion residente

Asignación de memoria

En el momento en que se instala un programa residente se

debe reserva toda la memoria que necesitará para su ejecución. Por esta razón no es posible asignarle más de la prevista una vez que está activado.

Esto significa que no se pueden utilizar instrucciones tales como malloc perteneciente al lenguaje "C". Para subsanar este inconveniente se declaran todas las variables globales que se creen necesarias, como por ejemplo un array de ECBs para alojar a los paquetes en el momento en que son recibidos.

Principal

Antes de quedar residente la aplicación debe ejecutar las siguientes operaciones:

- Verificar que los protocolos IPX y SPX de Novell estén instalados.
- Abrir el socket, que será permanente, ya que continuamente está a la espera de una conexión.
- Colocar paquetes a escuchar.
- Quedar a la espera de una conexión.
- Instalar la aplicación en forma residente.

Residente

La aplicación continuamente debe verificar si se ha establecido una conexión o si ha llegado algún paquete para copiar, lo cual exige que la ejecución sea en forma background (se activa 18 veces por segundo).

Los pasos a seguir son los siguientes:

- Verificar si está copiando un archivo enviado por la estación con la que estableció la conexión previamente.
- Si no está copiando un archivo, verificar si se estableció una conexión. Si no es así, seguir escuchando. Desactivarse.

- Verificar si llegó un paquete.

- Comprobar si el paquete arribó sin problemas.

- Verificar el tipo de paquete.

- Si es un paquete para terminar la conexión y estaba copiando un archivo enviado por la otra estación, cerrar el archivo y terminar la conexión. Quedar escuchando una nueva. Desactivarse.

- Si es un paquete de comando, verificar el tipo de comando.

- Si es un comando **PDIR** procesarlo, terminar la conexión y quedar a la espera de una nueva. Si ocurre algún error durante el proceso abortar la conexión y quedar escuchando una nueva. Desactivarse.

- Si es un comando para transmitir un archivo a otra estación, **"RXPCOPIA"**, procesarlo, teminar la conexión y quedar a la espera de una nueva conexión. Si ocurre algún error durante el proceso abortar la conexión y quedar escuchando una nueva. Desactivarse

- Si es un comando para recibir un archivo de otra estación, **"TXPCOPIA"**, verificar que el archivo no exista y abrirlo para comenzar a copiar. Si ocurre algún error, abortar la conexión y quedar escuchando una nueva. Desactivarse.

- Si es un paquete de datos escribir el paquete en el archivo previamente abierto. Si ocurre algún error, cerrar el archivo destino, borrarlo, abortar la conexión y quedar escuchando una nueva. Desactivarse.

- Si es un paquete de error, significa que ha ocurrido algún problema durante la ejecución de un comando **"TXPCOPIA"** en la otra estación.

Por lo tanto, cerrar el archivo en el que se está grabando, borrarlo, abortar la conexión y quedar escuchando una nueva. Desactivarse.

Comando PDIR

Para procesar este comando primero se debe verificar que el directorio o archivo solicitado exista. Si es así, buscar la información del directorio/archivos (considerando la capacidad del paquete SPX) y enviar el paquete a la estación remota (encargada de desplegarlo por pantalla). Estos últimos pasos se repiten hasta finalizar la búsqueda de la información solicitada.

Comando RXPCOPIA

Este comando es el encargado de enviar un archivo a la estación que lo solicite.

Si el archivo requerido no es oculto o del sistema, lo abre. Luego lee el archivo y forma paquetes con los fragmentos del mismo que serán enviados a la estación opuesta. Controla la ocurrecia de algún error durante el proceso.

Comando TXPCOPIA

Este comando actúa de manera diferente a los anteriores, ya que el residente analiza la llegada de un paquete cada vez que se activa. En cambio en los casos anteriores ejecuta todo el proceso de una sola vez y luego se desactiva.

Lo primero que hace es verificar si el archivo existe. Si no es así, lo abre y se desactiva. Cada vez que se activa y detecte la llegada de un paquete de datos lo graba en el archivo y se desactiva.

El comando finaliza cuando llega un paquete que indica el fin de la conexión, o se detecte un error, ya sea por la llegada de un paquete de error o la ocurrecia de alguno durante la ejecución. En el primer caso se cierra el archivo y la conexión. En el segundo se cierra y borra el archivo, y luego aborta la conexión. En ambos casos se queda esperando la llegada un paquete "establecer conexión".

Mensajes de Error

En todos los casos, en el momento en que se detecta un problema se envía a la estación remota un paquete de tipo "error" explicando cúal es el mismo y queda nuevamente esperando un paquete "establecer conexión". La otra estación despliega el mensaje por pantalla.

Captura de los errores de dispositivo

Siendo ésta una aplicación residente no puede mostrar los mensajes que el sistema emite ante algún error de dispositivo. Por ejemplo, cuando se quiere acceder a una diskketera vacía, cuando ocurre una falla en el disco, etc.

Para evitar que esto suceda se capturan todos errores que se produzcan de este tipo y se envía el mensaje a la estación remota.

6.3.9 Comunicación entre las Aplicaciones

Comunicación entre PDIR y TSRCOM

Supongamos que se quiere ver el directorio **c:** perteneciente a la estación llamada **nodo2** desde la estación **nodo1**.

Los pasos que siguen las aplicaciones en el intercambio de paquetes son los siguientes:

Nro	Nodo1	Red	Nodo2
0			- Pone paquetes a escuchar "paquetes secuenciados" y paquete de "establecer conexión". - Queda residente.
1	Se ejecuta : pdir nodo2:c:\		
2	Envía paquete de "establecer conexión".	————>	- Recibe paquete "establecer conexión". - Espera paquetes de la conexión establecida.
3	- Envía paquete de pedido "PDIR". - Se queda escuchando.	————>	Recibe un paquete de pedido.
4	- Recibe paquete de dato Nro. 1, lo despliega por pantalla. - Pone a escuchar nuevamente el paquete. - Se queda escuchando.	<————	Busca la información para el paquete de dato y envía el paquete de dato Nro. 1.
...	- Recibe paquete de dato Nro. ..., lo despliega por pantalla. - Pone a escuchar nuevamente el paquete. - Se queda escuchando.	<————	Busca la información para el paquete de dato y envía el paquete de dato Nro. ... .

10	Recibe paquete de "terminar conexión" y termina el comando.	<————	- Envía paquete "terminar conexión". - Pone paquetes a escuchar "paquetes secuenciados" (según la ventana) y paquete de "establecer conexión". - Queda residente.

Supongamos que ocurre un error en nodo2, por ejemplo no se encontró el directorio buscado c:\ms\. El intercambio de paquetes es el siguiente:

Nro	Nodo1	Red	Nodo2
0			- Pone paquetes a escuchar "paquetes secuenciados" y paquete de "establecer conexión". - Queda residente.
1	Se ejecuta : pdir nodo2:c:\ms\		
2	Envía paquete de "establecer conexión".	————>	Recibe paquete "establecer conexión". Quedó la conexión establecida.
3	- Envía paquete de pedido "PDIR". - Se queda escuchando.	————>	Recibe un paquete de pedido.
4	- Recibe paquete de error, lo despliega por pantalla. - Aborta la conexión. - Termina el comando.	<————	- No encuentra el directorio, envía paquete de error. - Aborta la conexión. - Pone a escuchar los paquetes recibidos. - Pone a escuchar un "establecer conexión". - Queda residente.

Supongamos que en el nodo1 se presiona Ctrl-Break o *Ctrl-C*. El intercambio de paquetes es el siguiente:

Nro	Nodo1	Red	Nodo2
0			- Pone paquetes a escuchar "paquetes secuenciados" y paquete de "establecer conexión". - Queda residente.
1	Se ejecuta : pdir nodo2:c:\		
2	Envía paquete de "establecer conexión".	————>	Recibe paquete "establecer conexión". Quedó la conexión establecida.
3	- Envía paquete de pedido "PDIR". - Se queda escuchando paquetes.	————>	Recibe un paquete de pedido.
4	- Recibe paquete de dato Nro. 1, lo despliega por pantalla. - Pone a escuchar nuevamente el paquete. - Se queda escuchando.	<————	Busca la información para el paquete de dato y envía el paquete de dato Nro. 1.
...	- Recibe paquete de dato Nro. ..., lo despliega por pantalla. - Pone a escuchar nuevamente el paquete. - Se queda escuchando.	<————	Busca la información para el paquete de dato y envía el paquete de dato Nro. ... .

10	- Se presiona Ctrl-Break. - Aborta la conexión. - Termina el comando.	
11		- Aborta la conexión. - Pone a escuchar los paquetes recibidos. - Pone a escuchar un "establecer conexión". - Queda residente.

Ahora supongamos que se corta la red, entonces el nodo1 recibe un paquete enviado por el protocolo SPX anunciando el problema.

Nro	Nodo1	Red	Nodo2
0			- Pone paquetes a escuchar "paquetes secuenciados" y paquete de "establecer conexión". - Queda residente.
1	Se ejecuta : pdir nodo2:c:\		
2	Envía paquete de "establecer conexión".	————>	Recibe paquete "establecer conexión". Quedó la conexión establecida.
3	- Envía paquete de pedido "PDIR". - Se queda escuchando paquetes.	————>	Recibe un paquete de pedido.

4	- Recibe paquete de dato Nro. 1, lo despliega por pantalla. - Pone a escuchar nuevamente el paquete. - Se queda escuchando.	<————	Busca la información para el paquete de dato y envía el paquete de dato Nro. 1.
...	- Recibe paquete de dato Nro. ..., lo despliega por pantalla. - Pone a escuchar nuevamente el paquete. - Se queda escuchando.	<————	Busca la información para el paquete de dato y envía el paquete de dato Nro. ... .
10	- Recibe paquete de "Falla de conexión". - Aborta la conexión. - Termina el comando.		
11			- Aborta la conexión. - Pone a escuchar los paquetes recibidos. - Pone a escuchar un "establecer conexión". - Queda residente.

Comunicación entre PCOPIA y TSRCOM

El comando PCOPIA se puede usar para:

- RXPCOPIA: copiar un archivo desde la estación remota a la estación local.

- TXPCOPIA: copiar un archivo desde la estación local a la estación remota.

En el primero, RXPCOPIA, los paquetes se intercambian de la misma forma que PDIR y TSRCOM. En cambio en TXPCOPIA se comportan distinto.

Supongamos que se quiere copiar un archivo desde la estación local **nodo1** a la estación remota **nodo2**.

Nro	Nodo1	Red	Nodo2
0			- Pone paquetes a escuchar "paquetes secuenciados" y paquete de "establecer conexión". - Queda residente.
1	Se ejecuta : pcopia nodo1:tesis.doc a nodo2:c:\doc\		
2	Envía paquete de "establecer conexión".	——————>	- Recibe paquete "establecer conexión". - Espera paquetes de la conexión establecida.
3	- Envía paquete de pedido "TXPCOPIA". - Se queda escuchando.	——————>	- Recibe un paquete de pedido.
4	- Recibe paquete de dato	<——————	- Envía paquete anunciando que puede comenzar a copiar.
5	- Toma la primer fracción del archivo y envía paquete de dato Nro. 1. - Pone a escuchar nuevamente el paquete. - Se queda escuchando.	——————>	- Recibe paquete de dato Nro. 1, lo copia en el archivo destino. - Pone a escuchar nuevamente el paquete. - Queda residente.

...	- Toma una fracción del archivo y envía paquete de dato Nro. ... . - Se queda escuchando.	——————>	- Recibe paquete de dato Nro. ..., lo copia en el archivo destino. - Pone a escuchar nuevamente el paquete. - Queda residente.
10	- Envía paquete "terminar conexión" y termina el comando.	——————>	- Recibe paquete de "terminar conexión". - Pone paquetes a escuchar "paquetes secuenciados" (según la ventana) y paquete de "establecer conexión". - Queda residente.

Supongamos que ocurre un error en nodo2, por ejemplo no se puede abrir el archivo destino por error de dispositivo. El intercambio de paquetes es el siguiente:

Nro	Nodo1	Red	Nodo2
0			- Pone paquetes a escuchar "paquetes secuenciados" y paquete de "establecer conexión". - Queda residente.
1	Se ejecuta : pcopia nodo1:tesis.doc a nodo2:c:\doc\		
2	Envía paquete de "establecer conexión".	——————>	- Recibe paquete "establecer conexión". - Espera paquetes de la conexión establecida.
3	- Envía paquete de pedido "TXPCOPIA". - Se queda escuchando.	——————>	- Recibe un paquete de pedido.

4	- Recibe paquete de dato	<————	- Envía paquete anunciando que puede comenzar a copiar.
5	- Recibe paquete de error, lo despliega por pantalla. - Aborta la conexión. - Termina el comando.	<————	- No puede abrir el archivo destino, envía paquete de error. - Aborta la conexión. - Pone a escuchar los paquetes recibidos. - Pone a escuchar un "establecer conexión". - Queda residente.

Supongamos que en el nodo1 se presiona Ctrl-Break o Ctrl-C. El intercambio de paquetes es el siguiente:

Nro	Nodo1	Red	Nodo2
0			- Pone paquetes a escuchar "paquetes secuenciados" y paquete de "establecer conexión". - Queda residente.
1	Se ejecuta : pcopia nodo1:tesis.doc a nodo2:c:\doc\		
2	Envía paquete de "establecer conexión".	————>	- Recibe paquete "establecer conexión". - Espera paquetes de la conexión establecida.
3	- Envía paquete de pedido "TXPCOPIA". - Se queda escuchando.	————>	- Recibe un paquete de pedido.
4	- Recibe paquete de dato	<————	- Envía paquete anunciando que puede comenzar a copiar.

5	- Toma la primer fracción del archivo y envía paquete de dato Nro. 1. - Pone a escuchar nuevamente el paquete. - Se queda escuchando.	——————>	- Recibe paquete de dato Nro. 1, lo copia en el archivo destino. - Pone a escuchar nuevamente el paquete. - Queda residente.
...	- Toma una fracción del archivo y envía paquete de dato Nro. ... . - Se queda escuchando.	——————>	- Recibe paquete de dato Nro. ..., lo copia en el archivo destino. - Pone a escuchar nuevamente el paquete. - Queda residente.
10	- Se presiona Ctrl-Break. - Aborta la conexión. - Termina el comando.		
11			- Aborta la conexión. - Pone a escuchar los paquetes recibidos. - Pone a escuchar un "establecer conexión". - Queda residente.

Ahora supongamos que se corta la red, entonces el nodo1 recibe un paquete enviado por el protocolo SPX anunciando el problema.

Nro	Nodo1	Red	Nodo2
0			- Pone paquetes a escuchar "paquetes secuenciados" y paquete de "establecer conexión". - Queda residente.
1	Se ejecuta : pcopia nodo1:tesis.doc a nodo2:c:\doc\		
2	Envía paquete de "establecer conexión".	——————>	- Recibe paquete "establecer conexión". - Espera paquetes de la conexión establecida.
3	- Envía paquete de pedido "TXPCOPIA". - Se queda escuchando.	——————>	- Recibe un paquete de pedido.
4	- Recibe paquete de dato	<——————	- Envía paquete anunciando que puede comenzar a copiar.
5	- Toma la primer fracción del archivo y envía paquete de dato Nro. 1. - Pone a escuchar nuevamente el paquete. - Se queda escuchando.	——————>	- Recibe paquete de dato Nro. 1, lo copia en el archivo destino. - Pone a escuchar nuevamente el paquete. - Queda residente.
...	- Toma una fracción del archivo y envía paquete de dato Nro. ... . - Se queda escuchando.	——————>	- Recibe paquete de dato Nro. ..., lo copia en el archivo destino. - Pone a escuchar nuevamente el paquete. - Queda residente.

10	- Recibe paquete de "Falla de conexión". - Aborta la conexión. - Termina el comando.	
11		- Aborta la conexión. - Pone a escuchar los paquetes recibidos. - Pone a escuchar un "establecer conexión". - Queda residente.

6.4 Desarrollo

6.4.1 Recursos del Desarrollo

Los medios que se utilizaron para el desarrollo de las aplicaciones son:

- Sistema operativo DOS versión 6.2.
- Red Novell NetWare versión 3.11.
- Protocolos de comunicación IPXODI.
- PCs compatibles con procesador 386 en adelante.
- Compilador C++ versión 3.0.
- Debugging Codeview versión 2.2

6.4.2 Funcion ReceiveESR

Guarda el puntero al paquete recibido **ESR_ECB** en **RcvECB** e indica que existe un paquete más para leer.

Es una rutina del servicio de evento (ESR) que se ejecuta cuando llega un paquete. Un ESR es una rutina de interrupción de servicio.

Esta rutina es invocada por la función externa **ReceiveESRHandler**, reponsable de guardar los registros SS y SP así como de retornar apropiadamente después de la ejecución.

Esta rutina es similar para todas las aplicaciones desarrolladas.

Entradas : - **ESR_ECB**, puntero al paquete recibido.
- **RcvECB**, variable global.
- **cant_rcv**, variable global.
- **lRcvECB**, variable global.

Salidas : - **RcvECB**, array de punteros a ECBs que *fueron* recibidos.
- **cant_rcv**, cantidad de paquetes recibidos.
- **lRcvECB**, indice al elemento del **RcvECB** que apuntará al próximo paquete a recibir.

```
void ReceiveESR(ECB *ESR_ECB)
{
```

Asignación del puntero al paquete recibido **ESR_ECB** en el array de punteros de paquetes recibidos **RcvECB**.

```
    RcvECB[lRcvECB] = ESR_ECB;
```

Se hace apuntar **lRcvECB** al elemento donde se guardará el próximo paquete que llegará.

```
    if (lRcvECB == VENTANA - 1)  /* Si es el último */
       lRcvECB = 0;
    else
       lRcvECB++;
```

Se aumenta la cantidad de paquetes recibidos y no analizados.

```
    cant_rcv++;
}
```

6.4.3 Desarrollo del Manejador de Rutina de Servicio de Eventos (Handrcv.asm)

Una rutina de servicio de eventos es llamada cuando ocurre un evento particular.

Las ESRs (Rutinas de Servicios de Eventos) son llamadas por IPX cuando concluye algún evento. En este desarrollo sólamente se trata el evento producido por la llegada de un paquete. El ECB que está escuchando contine un campo que apunta a la función ESR; cuando ésta es llamada obtiene el puntero de vuelta al ECB.

Antes de llamar a la rutina de servico de eventos, el shell deshabilita las interrupciones.

También, setea el campo InUseFlag del ECB a 00h, registra la información apropiada perteneciente al evento en el ECB, y remueve al ECB de la lista interna de IPX. De esta manera la ESR está libre para manipular el ECB.

Función ReceiveESRHandler

Esta rutina es llamada por IPX cuando se produce la llegada de un paquete. La misma se encarga de llamar a la rutina ReceiveESR, escrita en lenguaje "C" que coloca el ECB recién llegado en un array de ECBs que aún no analizados. Ver secciones 6.4.2 Funcion ReceiveESR y 6.3.2 Recepción de paquetes.

Entradas : - Ninguna.

Salidas : - Ninguna.

Inicializa los segmentos de datos y código.

```
DGROUPgroup_DATA,_BSS
     assumecs:_TEXT,ds:DGROUP
_DATAsegment word public 'DATA'
_DATAends
_BSS segment word public 'BSS'
_BSS ends
_TEXTsegment byte public 'CODE'
     assumecs:_TEXT
     ;*
     ;** Hace la rutina visible
     ;** fuera del módulo assembler
     ;*
     public_ReceiveESRHandler
     ;*
     ;** referencia a la función
     ;** declarada en "C"
     ;*
     extrn_ReceiveESR: near
```

Guarda en el stack los registros ES y SI, que contienen el puntero asociado al ECB recién llegado.

```
_ReceiveESRHandlerprocfar
     movax, DGROUP
     movds, ax        ;* "ds" apunta a DGROUP *
     pushes           ;* guarda "es" en el stack *
```

```
    pushsi          ;* guarda "si" en el stack *
    call_ReceiveESR ;* llama a la rutina *
    addsp, 4        ;* limpia el stack *
    ret              ;* retorna a C *
_ReceiveESRHandlerendp
_TEXT   ENDS
    END
```

6.4.4 Manejo de Control-C

Siempre es difícil manejar las interrupciones de Ctrl-C y Ctrl-Break en MS-DOS. Ellas pueden causar que los programas aborten cuando menos se lo espera, pudiendo producir desastres.

Los pasos a seguir para interceptar esta interrupción son:

- Guardar el puntero de la función que corresponde a la interrupción 1B contenida en el vector de interrupciones.

- Reemplazar esa función por una propia que maneje el Ctrl-C y Ctrl-Break como se desea.

- Una vez que no es más necesaria el Ctrl-Break especial volver a colocar en el vector de interrupciones el puntero a la función original.

Para lograr el manejo de estas interrupciones se recurre a las rutinas facilitadas por el compilador en controlc.h. Estas ejecutan los pasos anteriormente descriptos haciéndolos transparentes al programador.

Estas funciones utilizan la siguiente variable global:

```
void (* _far_cdecl _controlc_handler) (void);
```

Para usarla se instala un puntero a la función propia que maneja el Ctrl-C/Ctrl-Break en la variable global _controlc_handler, y luego se llama a la función cotrolc_open() para activarla. Para desactivarla se invoca a la función controlc_close().

Ejemplo

```
int _far _cdecl myhandler(void)
{
      controc_ocurrio = 1;
}
void main(void)
{
      // asigna el manejador
      _cerror_handler = myhandler;
      // insatala el manejador de Ctrl-C / Ctrl-Break
      controlc_open();
      // insatala el manejador de Ctrl-C / Ctrl-Break
      controlc_close();
}
```

Función cerror_open

Uso

```
#include <controlc.h>

int _cdecl _controlc_open(void);
```

Descripción

Instala el manejador de Ctrl-Break del usuario. Un puntero al manejador ha sido previamente instalado en la variable global **controlc_handler.** El manejador debe ser una función **far** que no tiene argumentos de entrada ni de salida.

Función controlc_close

Uso

```
#include <cerror.h>

int _cdecl _controlc_close(void);
```

Descripción

Remueve un manejador de Ctrl-Break suministrado por el usuario. El manejador previamente ha sido instalado por **controlc_open().**

6.4.5 Manejando Errores Criticos

Los errores críticos pueden ocurrir bajo MS-DOS al intentar acceder a los dispositivos de I/O. Ellos pueden causar el mensaje **Abort, Retry, Ignore, Fail** que aparece en la pantalla en el momento más inoportuno.

Las funciones **cerror_** proveen una solución a este problema.

Estás funciones utilizan la siguiente variable global:

```
int (*_far _cdecl _cerror_handler) (int *ax, int *di);
```

Para usarlas, se instala un puntero al manejador en la variable **_cerror_handler**, y luego se llama a **cerror_open()** para activarlo.

En el presente desarrollo, estas funciones se utilizan en un programa residente (ver sección 6.4.9 Desarrollo de la Aplicación Residente (programa tsrcom.c)), para evitar que los mensajes de error crítico del sistema se emitan en la pantalla mientras el usuario está ejecutando otra aplicación.

Ejemplo

```
int _far _cdecl myhandler(int *ax, int *di)
{
     ...
}
void main(void)
{
    // asigna el manejador
    _cerror_handler = myhandler;
    // insatala el manejador de errores críticos
    cerror_open();
    // remueve el manejador de errores críticos
```

```
    cerror_close();
}
```

Función cerror_open

Uso

```
#include <cerror.h>

int _cdecl _cerror_open(void);
```

Descripción

Instala el manejador de errores críticos del usuario. Un puntero al manejador ha sido previamente instalado en la variable global **_cerror_handler**. El manejador debe ser una función **far** que toma dos argumentos enteros y retorna un entero.

Función cerror_close

Uso

```
#include <cerror.h>

int _cdecl _cerror_close(void);
```

Descripción

Remueve un manejador de errores críticos suministrado por el usuario. El manejador previamente ha sido instalado por **cerror_open()**.

6.4.6 TSR Package

Las rutinas provistas en este paquete permiten escribir programas en Zortech C y C++ que pueden opcionalmente quedar en memoria residente. Tales programas se llaman **TSRs** o **Pop Up**. En suma los programas pueden opcionalmente tener una fracción del tiempo de procesador, permitiendo a un programa cuidadosamente escrito ejecutarse como un proceso background.

El paquete TSR sólo se usa con MS-DOS. No lo soporta el OS/2 o DOS extendido de 16 o 32 bits.

Este paquete se utilizó en el programa TSRCOM.C, que se desarrolla en la sección 6.4.9. Aquí se explicará como trabaja y cuáles son sus funciones principales.

Escribir un TSR o Pop Up

Escribir un TSR puede no ser fácil. Para escribir un programa de esta clase con este paquete se deben considerar los siguientes puntos:

- No usar funciones que asignen memoria (malloc, etc). Se puede usar Page Package para manejar memoria propia si se desea.
- No usar las rutinas tales como fgets, etc, que usan *FILE, reemplazarlas por rutinas como open/read/close.
- No salir con exit de cualquier función, simplemente retornar.

Hacer un Programa Residente

Para dejar el programa residente en memoria se llama a la siguiente función:

```
tsr_install(int argument)
```

Si tiene éxito, la función **no** retorna. Si lo hace, es porque se ha producido un error. No necesariamente el programa se pone inmediatamente residente en memoria. Por ejemplo, se podría tener un menú de opciones para que lo haga residente.

Se puede observar que la rutina antes mencionada tiene un argumento, este puede tener dos alternativas, dependiendo del tipo de programa que se escriba. Estas alternativas son **POPONLY** y **TIMESLICE**.

```
tsr_install(POPONLY);
```

Este argumento convierte al programa en un TSR normal y la función especial popmain sólo es llamada cuando el usuario presiona la combinación de teclas (hotkey) declaradas en el mismo. La otra alternativa es:

```
tsr_install(TIMESLICE);
```

Si se usa éste método, el programa se convierte en background y popmain será llamada reiteradamente hasta un máximo de 18 veces por segundo. También popmain es llamada cuando el usuario presiona la combinación de teclas (hotkey) correctas.

Usualmente se desea saber cuándo popmain es llamada a través de la combinación de teclas o de el algoritmo timeslice. Esta pregunta tiene respuesta examinando la variable global llamada_tsr_timeslice. Si popmain() es llamada por el algoritmo timeslice esta variable se setea a 1, de otro modo se setea a 0.

Si se decide escribir una tarea background, hay que diseñar el programa lo más eficiente que sea posible.

Se puede hacer el programa más compatible con otros TSRs dando a ellos la chance de activarse en el momento más conveniente dentro del propio TSR. Esto se puede hacer con la función tsr_service.

```
void tsr_service(void)
```

Por ejemplo, en lugar de esperar para que se presione una tecla:

```
bioskey(0)
```

Intentar en su lugar:

```
while(bioskey(1)==0)  /* mientras no se presione una
                         tecla */

   tsr_service();     /* dar otra chance */

bioskey(0);           /* obtener tecla normalmente */
```

La función tsr_service simplente dispara una int 28h. Ningún valor en pasado o retornado de tsr_service.

Debugger

Cuando se entra al mundo de la programación de TSRs, se aceptan ciertas cosas que nunca se pueden hacer y otras que se pueden sólo algunas veces. Posiblemente el resultado de hacer alguna de ellas será el bloqueo de la computadora, siendo difícil rastrear estos problemas.

Para ayudar con esta tarea, se incluye una facilidad que puede atrapar y alertar de alguna posible mala operación dentro del programa TSR. Cuando se usa ésta facilidad y alguna acción ilegal se detecta, una ventana se abre con un significativo mensaje dentro de ella. Esto ayuda a rastrear la llamada a la función que está causando los problemas.

Para activar el debugger, simplemente sumar el comando,

```
|TSR_DEBUG
```

al existente tsr_install. Por ejemplo, si se usa la forma:

```
tsr_install (POPONLY);
```

Simplente extenderlo a,

```
tsr_install (POPONLY|TSR_DEBUG);
```

Similarmente se puede usar,

```
tsr_install (TIMESLICE|TSR_DEBUG);
```

Esta facilidad **no** intenta remediar la acción ilegal, sólo alerta al usuario (programador), espera por una tecla y continúa normalmente. Este no evita que el computador quede bloqueado, pero explica por qué se bloquea.

Cuando se invoca las rutinas del debugger se pueden atrapar varias de los problemas comunes que se pueden encontrar. Para cada problema diferente se puede ver un significativo mensaje. Algunos son los siguientes:

Función ODh del DOS Presionar una tecla

Esto significa que alguna función en el programa ha hecho una llamada a la INT 21h con el registro ah seteado un el valor menorqu ODh. Esto es ilegal en un programa TSR.

Para detectar la función que produce este error se podría colocar mensajes en el código. Probablemente sea de la familia del **getch.**

Intenta cerrar un handle std Presionar una tecla

Cada vez que se abre un archivo DOS le asigna un manejador (handle). Este handle es usado cuando se desea leer o escribir. Los handles que DOS asigna comienzan en 5 y se incrementan con cada solicitud de apertura. Los handles de 0000 a 0004 están reservados por DOS para sus dispositivos standard. Estos son tales como el teclado/pantalla/impresora y port com.

Es posible que el programa cierre estos dispositivos reservados, ya sea intencional o por accidente. Si se requiere al DOS cerrar uno de estos handles standard, el debugger asume que se ha incurrido en un error y lo informa.

Asignación de Memoria No dentro de TSR!

La memoria de los programas residentes esta dada por una porción importante de 640k del DOS cuando se hace la transición de los programas normales a TSRs. Si ellos intentan requerir memoria adicional, DOS intentaría forzar y obtener la misma fuente y confundir sinceramente. El debugger detecta un intento de obtener memoria adicional y la ventana anterior aparece.

Si se desea manejar memoria dinámica con un TSR se debe crear un buffer estático y convertirlo en una pila usando **page_initialize**. Se puede usar **page_malloc**, etc. para manejar la memoria.

Hay otro mensaje de error que se puede observar.

Exit detectado Use return en su lugar

En un programa normal probablemente se use la función exit para abortar el programa. Sin embargo se debe recordar que en DOS siempre hay un sólo programa ejecutando. Este asume que la aplicación subyacente (Wordstard etc) y el TSR son una y la misma. Así, si finaliza el programa residente con una llamada a exit, DOS podría interpretar que el program subyacente ha requerido abortar (exit) y lo ejecuta de este modo. Para evitar conflicto se debería sólo retornar, nunca usar **exit**.

Remover el Programa

LLamando a:

```
int tsr_install (void);
```

el programa intentará remover una copia cargada previamente de sí mismo.

Esta función siempre retorna un valor. Los distintos códigos de retorno se muestran más adelante. Esta función puede ser llamada, o desde dentro del TSR cuando está activo, o desde una rutina que se ejecuta cuando el programa es llamado desde el prompt del DOS. Si está dentro del pop up cuando está activo puede remover la copia actual del programa TSR de la memoria. Así una vez que el programa se desactive no será capaz de activarse nuevamente. Si es llamada cuando el programa está ejecutando desde el DOS, éste remueve alguna copia de sí mismo previamente cargado.

Si se desea remover un programa de memoria cuando está activo es conveniente entender como DOS asigna y desasigna memoria.

Cuando un programa es cargado por DOS, esté asigna uno o más segmentos de memoria. Un segmento está arriba de 64k bytes. Un programa inteligente puede rastrear a través de los registros de asignación de memoria del DOS y averiguar el propietario de algún segmento (o parte del segmento) de memoria. Cuando se usa el kit de herramientas TSR, los programas automáticamente tienen está habilidad y ésta es usada cuando se intenta remover el programa de memoria.

Cuando se llama a tsr_uninstall, éste mira en la memoria y retorna al DOS todo segmento que ha sido asignado al programa. También desactivan todas las interrupciones que usen las rutinas TSR. Sin embargo, es importante darse cuenta que porque los segmentos se retornen a DOS, no significa que el programa no esté en memoria, está y puede continuar ejecutando después que tsr_uninstall retorna. Aunque DOS ahora considera la memoria previamente asignada al programa como libre, en realidad ésta todavía contiene la imagen del programa, es por eso que puede continuar ejecutando. Los bloques de memoria liberados sólo son reusados cuando DOS asigna memoria a otro programa.

Considerar la siguiente situación:

1. Se carga un programa TSR, DOS asigna la memoria y retorna al prompt del DOS.

2. Se carga un programa ordinario, Wordstar por ejemplo.

3. El programa TSR se activa cuando Wordstar está ejecutando y contiene la opción para removerse a sí mismo de memoria.

4. Se selecciona la opción para remover el TSR.

En el escenario anterior se ha creado un hueco en la memoria del DOS, este se produce porque la memoria fue asignada de la siguiente forma:

DOS DRIVES

...

El programa TSR

Wordstar

Ahora que DOS ha recuperado los bloques asignados al TSR aparece un espacio. Sin embargo, DOS es perfectamente capaz de manejar tal situación. Puede usar la memoria del espacio libre si es suficiente para sus necesidades, esto sin sobreescribir Wordstar (o cualquier otra aplicación).

Finalmente cuando Wordstar termina, DOS vuelve a recuperar toda la memoria asociada a éste y el espacio libre desaparece. Los valores retornados por **tsr_install** y **tsr_uninstall** son los siguientes:

Valor	Descripción
0	Función exitosa.
1	No puede cargar, el programa ya está cargado.
2	No puede remover, el programa **no** está cargado.
3	No puede remover, otro programa TSR ha sido cargado sobre el programa.

Variables Globales

En el programa fuente se deben especificar ciertas variables que serán utilizadas por las rutinas TSR. El valor que se asigne a las mismas determinará cómo trabajarán las rutinas.

HOTSHIFT y HOTSCAN

Un programa TSR debe tener una secuencia especial de teclas que son reconocidas como una señal para activarlo. Esta se llama usualmente la "combinación

hotkey". Es llamada así porque es una combinación de una o más teclas shift u una tecla ordinaria (usualmente en el rango A-Z). Cuando ésta combinación es presionada el residente toma el control de los recursos del computador y puede ejecutarse como único programa. La manera de espesificar la combinación hotkey en el programa es declarando e inicializando dos variables llamadas **HOTSHIFT** y **HOTSCAN**.

HOTSHIFT es un entero que contiene el valor que representa la tecla shift que se elige. La manera de determinar el valor en este entero es la siguiente:

Elegir las teclas shift (una o más) de entre las siguientes disponibles:

LSHIFT	Tecla shift izquierda.
RSHIFT	Tecla shift derecha.
CTRL	Tecla control
ALT	Tecla alt

Luego simplemente declarar un entero llamado HOTSHIFT e inicializarlo con la hot shift elegida, cono esta:

```
int HOTSHIFT=ALT+RSHIFT;
```

Esto declara la hot shift como la tecla alt + la tecla shift derecha.

También se debe elegir y declarar la tecla que es usada con la shift. Elegir una tecla en el rango A-Z y declara un entero llamado HOTSCAN. Inicializarlo, por ejemplo:

```
int HOTSCAN=SCAN_Q;
```

Esto declara la tecla como Q, así, cuando alguien presione:

```
ALT+RIGHT SHIFT+Q
```

El programa se activará. Notar que se debe inicializar HOTSCAN con un valor scan, no con el caracter en sí mismo.

```
int HOTSCAN='Q';            /* INCORRECTO! */

int HOTSCAN=SCAN_Q;         /* CORRECTO */
```

Todos los valores scan para las teclas A-Z y F1 a F10 están definidas en el archivo **tsr.h**.

Si prefiere no usar un valor scan y sólo usar una tecla shift, se declara HOTSCAN como sigue:

```
int HOTSCAN=NO_SCAN;
```

Con esta instrucción las rutinas TSR ignoran los valores scan y sólo testean los valores de las teclas shift.

tsr_fprint

Esta es una cadena de caracteres, de 20 bytes de longitud. Son usadas por las funciones install y unisntall para determinar si el programa ha sido cargado en memoria.

Ejemplo:

```
char tsr_fprint[20]= "Prog ID";
```

_okbigbuf

Las rutinas TSR tienen que determinar cuánta memoria requiere el programa, así se puede liberar la memoria remanente para que otra aplicación pueda ejecutar. Para que las rutinas tomen una óptima imagen del programa, éste debe contener la siguiente línea antes de la función principal.

```
extern int _okbigbuf = 0;
```

_tsr_timeslice

Si se necesita averiguar si el popmain() fue llamado porque fueron presionadas las hotkey o por el algoritmo timeslice, se puede usar la variable global _tsr_timeslice.

Si se presionaron las hotkey la variable es seteada a cero, si el responsable es el algoritmo es seteada a uno. Usando esto se puede tener una tarea bakground que puede ser activada y configurada de algún modo por el usuario.

La Función Especial popmain

Cuando la combinación de teclas hotkey es presionada las rutinas TSR pasan el control a una función del programa llamada **popmain**. Al escribir esta función se deben grabar algunas áreas de la pantalla que se puedan destruir, también se debe grabar la posición y forma del cursor, recordar que a esta altura se tiene el control de la computadora. Esto se puede hacer con las funciones suministradas por Zortech en el paquete Display.

Cuando se completa el proceso restaurar las áreas de la pantalla que se pudieron sobrescribir y retornar el control de popmain de vuelta a las funciones de TSR que la llamaron, quienes retornan el control a la aplicación que estaba ejecutando.

Limitaciones

Las limitaciones de los programas que usan estas rutinas son las mismas que para cualquier programa TSR. Sólo se puede activar cuando DOS está estable y no se están haciendo accesos al disco. Esto es tomado cuidadosamente por las rutinas TSR. Sin embargo los programas TSR no pueden asignar memoria o hacer llamados a funciones de DOS más alla de 0Dh.

Las funciones que se deben evitar son:

malloc (o alguna otra relacionada con el recurso memoria)

fread, fgets, etc.

Algunas Funciones del Paquete

Estas funciones no están disponibles bajo OS/2 o cuando se usan los modelos de memoria Z, X o P.

tsr_install

Uso

```
#include <tsr.h>

int tsr_install(int flag);
```

Descripción

Hace que un programa termine pero permanezca residente. Instala el manejador de interrupción TSR y setea las hotkey. Si el flag es TIMESLICE, popmain es llamada con la interrupción de reloj, un máximo de 18.2 veces por segundo. Si el flag es POPONLY, popmain se activa sólo cuando se presionan las hotkey. EL valor TSR_DEBUG puede ser sumado al flag de esta forma TSR es instalado con el modo debugging activado.

Valor Retornado

Retorna 1 si no puede instalar porque otra copia ya ha sido instalada. Si la función tiene éxito no retorna.

tsr_service

Uso

```
#include <tsr.h>

int tsr_service(void);
```

Descripción

Causa que se emita una interrupción 0x28, permitiendo a otros TSRs tener una chance para ejecutar mientras el actual TSR está activo.

tsr_uninstall

Uso

```
#include <tsr.h>

int tsr_uninstall(int flag);
```

Descripción

Si se llama desde un TSR (por ejemplo, uno que todavía no ha intentado convertirse en residente), intenta remover una versión previa del programa TSR. Si se llama desde un TSR activo, intenta removerse a sí mismo.

Valor Retornado

0 Exitoso

2 No puede remover, el programa no ha sido cargado.

3 No puede remover, otro TSR está instalado sobre el programa.

6.4.7 Desarrollo del Comando PDIR (programa pdir.c)

Este programa permite desplegar la información del directorio o archivo deseado que se encuentra almacenada en el disco de una estación remota en la pantalla de la estación local.

Para ello establece una conexión entre las estaciones utilizando los protocolos IPX/SPX de Novell. Mediante esta conexión se obtendrá dicha información que se desplegará por pantalla según la opción ingresada.

Recibirá como argumentos de entrada el nombre asignado al nodo de la estación remota y el directorio o archivos que se desean visualizar.

Recordemos la sintáxis del comando:

"PDIR [/p /c] <nodo>:<disco>:\<camino>\<archivo>"

Devolverá OKAY (0) si el comando se pudo ejecutar correctamente y ERROR (-1) si sucedió lo contrario.

Para mayor información sobre las funciones externas usadas consultar las secciones 6.4.10 (ipxspx.c), 6.4.11 (conexion.c), 6.4.12 (dirarch.c).

Includes

Includes provistos por el compilador.

```
#include <stdio.h>
#include <dos.h>
#include <string.h>
#include <direct.h>
#include <ctype.h>
#include <controlc.h>
#include <disp.h>
```

Includes correspondientes a módulos desarrollados especialmente para las aplicaciones.

```
#include "ipxspx.h"
#include "dirarch.h"
#include "conexion.h"
```

Definiciones de Constantes

Se define la cantidad de paquetes que pueden ser recibidos o enviados por la estación.

```
#define VENTANA 5
```

Se define la máxima cantidad de ffblk en el fragmento de datos del ECB.

```
#define MAXINFO SPXDATOS/sizeof(struct FIND)
```

Se definen las opciones que pueden ser ingresadas como parámetros para dar formato a la salida por pantalla.

```
#define SETPAG  0x007    /* Opción de paginar    */
#define SETCOL  0x070    /* Opción de encolumnar */
```

Se definen la máxima cantidad de líneas y columnas que puede tener una pantalla, éstas serán usadas cuando se ingrese alguna opción en los parámetros de entrada.

```
#define MAXLINE 23  /* Máx. cantidad de líneas de pantalla   */
#define MAXCOL  4   /* Máx. cantidad de columnas de pantalla */
```

Se define la sintaxis del comando para usarla en los mensajes de error.

```
#define SINTAX  "PDIR [/p /c] <nodo>:<disco>:\\<camino>\\<archivo>"
```

Prototipos de Funciones Privadas

Función que despliega el help del comando.

```
static void Help (void);
```

Función que verifica si las opciones ingresadas por el usuario son correctas.

```
static int  VerificarOp   (int argc,
                           char *argv[],
                           signed int *opcion);
```

Función que verifica que si los argumentos de nodo, directorio y archivo a mostrar están correctos.

```
static int  VerificarArg  (char *directorio,
                           char *nodo,
                           char *fuente);
```

Función que establece la conexión con el nodo remoto, intercambia la información y cierra la conexión.

```
static int  PDir          (char *nodo,
                           char *fuente,
                           unsigned int opcion);
```

Función que muestra por pantalla la información acerca de directorios y archivos ingresados por el usuario.

```
static void  Mostrar      (char *nodo,
                           char *fuente,
```

```
                    struct FIND ffblk[],
                    long cant,
                    unsigned int opcion);
```

Función que dependiendo de la opción de formato ingresada muestra el total de archivos y la cantidad total de bytes que ocupan.

```
static void  MostrarFin (unsigned int opcion);
```

Función que utilizada al visualizar la información para permitir continuar una página en la pantalla.

```
static void  Continuar (void);
```

Función que permite controlar el Ctrl-Break.

```
static void _cdecl CtrlBreakl(void);
```

Función que preserva los punteros a los paquetes recibidos.

```
static void ReceiveESR (ECB *ESR_ECB);
```

Prototipo de Funciones Externas

Función externa que invoca a la rutina del servicio de evento ReceiveESR.

```
extern void ReceiveESRHandler (); /* en Assembler */
```

Programa Principal

Se encarga de llamar a la rutina que realizan los controles de los argumentos ingresados y a la rutina que ejecuta las acciones necesarias para obtener la información deseada de la estación remota.

Entradas : - **argc**, número de argumentos con los que se invoca el comando en el promt.

- **argv[]**, vector de punteros a cadenas de caracteres que contienen los argumentos con los que fue invocado.

Salidas : - ERROR u OKAY según corresponda.

```
main(int argc, char *argv[])
{

    char        nodo[10];       /* nombre de nodo        */
    char        fuente[MAXPATH]; /* información requerida */
    unsigned int opcion;        /* opción de Formato     */
```

El comando puede tener como máximo 4 argumentos, el primero es el nombre del comando (pdir), el segundo y el tercero las opciones (/p /c) y por último el nodo y directorio/archivos que se desean visualizar. En caso de que el número de argumentos sea mayor se produce un error.

```
    /* Verifica que la cantidad de argumentos sea la que necesita
    ** el comando.
    */
    if (argc > 4) {
          printf("\n%s %s\n", "Error de sintaxis:", SINTAX);
          return ERROR;
    }
```

Si se ingresó sólo el comando o el comando con el argumento '?' se llama a la función (**Help**) que despliega la ayuda del mismo:

```
pdir
pdir ?
```

```
    /* Help del comando */
    if (argc == 1 || (argc == 2 && *argv[1] == '?')) {
          Help();
          return OKAY;
    }
```

Inicializa la variable opcion para que se asuma por defecto que no se ingresó ninguna opción como argumento en la línea de comandos.

```
    /* Inicializa opción */
    opcion = 0x000;
```

Sólo pueden existir opciones de formato cuando se ingresan más de dos argumentos, por lo tanto, en este caso se verifica que los mismos sean correctos a través de la función **VerificarOp()**. Si existe algún error se termina el programa.

```
    /* Verifica opciones */
    if (argc > 2 && VerificarOp(argc, argv, &opcion) ==
                            ERROR)
        return ERROR;
```

También se debe verificar que el nodo y el camino de directorios y archivos no tengan errores (función **VerificarArg()**), si los tiene se termina el programa.

```
    /* Verifica la fuente */
    if (VerificarArg(argv[argc - 1], nodo, fuente) == ERROR)
        return ERROR;
```

Una vez que no hay errores de sintaxis se llama a la función PDir que será la encargada de mostrar la información requerida por pantalla. Si se produce algún error en la misma se termina el programa con error.

```
    /* Muestra la fuente */
    if (PDir(nodo, fuente, opcion) == ERROR)
        return ERROR;
```

Se ejecutó el comando correctamente.

```
    return OKAY;
}
```

Función Help

Se encarga de desplegar la ayuda del comando.

Entradas : - Ninguna.

Salidas : - Ninguna.

No tiene entradas ni salidas.

```
static void Help(void)
{
```

Para visualizar la ayuda se utiliza el Display Package provisto por el compilador. Este permite hacer las I/O de caracteres por pantalla en forma más rápida.

En la ayuda se despliega el modo en que se invoca el comando, una breve explicación de su función y las opciones que se pueden usar.

```
    disp_open();
    disp_move(0,0);
    disp_eeop();
    disp_box(0, DISP_NORMAL, 2, 0, 20, 79);
    disp_move(4, 3);
    disp_printf("Sintaxis:");
    disp_move(6, 3);
    disp_printf("\t%s", SINTAX);
    disp_move(8, 3);
    disp_printf("Descripción:");
    disp_move(10, 0);
    disp_printf("\tMuestra el directorio de otro nodo.\n\n");
    disp_printf("\tTodos argumentos son obligatorios exceptuando las opciones.");
    disp_move(14,3);
    disp_printf("Opciones:");
    disp_move(16,3);
    disp_printf("\t? : Ayuda\n");
    disp_printf("\tp : Paginar\n");
    disp_printf("\tc : Encolumnar\n");
    disp_move(20, 0);
    disp_close();
}
```

La salida por pantalla es:

```
Sintaxis:

     PDIR [/p /c] <nodo>:<disco>:\<camino>\<archivo>

Descripción:

     Muestra el directorio de otro nodo.

     Todos argumentos son obligatorios exceptuando las opciones.

Opciones:

     ? : Ayuda
     p : Paginar
     c : Encolumnar
```

Función VerificarOp

Su objetivo es verifcar que las opciones ingresadas sean correctas.

Entradas : - **argc**, número de argumentos con los que se invoca el comando en el promt.

- **argv[]**, vector de punteros a cadenas de caracteres que contienen los argumentos con los que fue invocado el comando.

Salidas : - **opcion**, contiene las opciones ingresadas en los argumentos de entrada.

- ERROR u OKAY según corresponda.

```
static int VerificarOp(int argc, char *argv[], unsigned int *opcion)
{
    short i;   /* índice a los argumentos */
```

Recorre los argumentos que pueden llegar a ser opciones, el índice 0 es el comando y el último es la información requerida, por lo tanto, 1 <= i < argc -1.

```
    /* Busca las opciones ingresadas */
    for (i = 1; i < argc - 1; i++) {
```

Si la opción es paginar ("/p", "/P") se verifica que anteriormente no se haya ingresado la misma opción, por ejemplo, pdir /p /p. Si esto sucede, hay un error, de lo contrario se setea la variable opcion con el define para paginar.

El seteo de la variable opcion se hace con el operador binario or (|) para respetar la combinación que se pudiera tener de las respectivas opciones.

```
        /* Si la opción es "/p o "/P" */
        if (!strcmp(argv[i], "/p") || !strcmp(argv[i], "/P")) {
            if ((*opcion & SETPAG) == SETPAG) {
                /* La opción se ingresó más de una vez */
                printf("\n%s %s\n", "Error en opciones:", SINTAX);
                return ERROR;
            }
            else {
                /* La opción se ingresó una vez */
```

```
                *opcion |= SETPAG;
            }
        }
```

Si la opción no es paginar, se verifica si es encolumnar ("/c", "/C") y en este caso se controla que anteriormente no se haya ingresado esta opción, por ejemplo, pdir /c /c. Si esto sucede, hay un error, de lo contrario se setea la variable opcion con el define para encolumnar.

Se están controlando todas las posibles combinaciones de duplicación de opciones en los argumentos de entrada.

```
        else if (!strcmp(argv[i], "/c") || !strcmp(argv[i], "/C")){
            /* Si la opción es "/c o "/C" */
            if ((*opcion & SETCOL) == SETCOL) {
                /* La opción se ingresó más de una vez */
                printf("\n%s %s\n", "Error en opciones:", SINTAX);
                return ERROR;
            }
            else {
                /* La opción se ingresó una vez */
                *opcion |= SETCOL;
            }
        }
```

Si no se trata de la opción paginar y/o encolumnar y aún se cumple la condición del ciclo, 1 <= i < argc -1, el argumento es incorrecto por lo que retorna ERROR.

```
        else {
            /* La opción es incorrecta */
            printf("\n%s %s\n", "Error en opciones:", SINTAX);
            return ERROR;
        }
    }
```

Las opciones son correctas y la variable opcion está seteada con ellas.

```
    return OKAY;
}
```

Función VerificarArg

Verifica que la fuente (nodo-disco-directorio-archivo) sea correcta.

Entradas : - **directorio**, compuesto por nodo-disco-directorio-archivo

Salidas : - **nodo**, nombre de la estación remota.

- **fuente**, compuesta por disco-directorio-archivo.
- ERROR u OKAY según corresponda.

```
static int VerificarArg (char *directorio, char *nodo, char *fuente)
{
      char          nodofuente[MAXPATH+10];
      char          *ptr;
      char          drive[MAXDRIVE];
      char          dir[MAXDIR];
      char          file[MAXFILE+1];
      char          ext[MAXEXT+1];
      int           errornodo;
      unsigned int flags;
      BYTE          networkAddress[LONGNET+LONGNODE],
                    dnetwork[LONGNET],  dnodo[LONGNODE];
```

Busca la dirección de red y de nodo de la estación local por medio de la API de IPX.

```
      /* Busca la dirección de la estación local */
      IPXGetInternetworkAddress(networkAddress);
```

Separa el nodo fuente del resto del directorio (lo guarda en la variable nodo), lo distingue buscando la primera aparición del caracter ':', si éste no se encuentra se produce un error.

```
      /* Verifica que el nodo fuente sea un argumento del comando.
      ** Busca los primeros ':'
      */
      ToUpper(directorio, nodofuente);
      ptr = strchr(nodofuente, ':');
      if (ptr) {
            memset(nodo, '\0', 10);
            strncpy(nodo, nodofuente, ptr-nodofuente);
```

```
        strcpy(fuente, ptr+1);
    }
    else {
        printf("\nEl nodo fuente es obligatorio.\n");
        return ERROR;
    }
```

Verifica que el nodo exista en el archivo de mapeo (mapa.txt), si no existe o no se encuentra la tabla de mapeo se consiera error.

```
    /* Verifica que el nodo fuente exista en la tabla de mapeo */
    if ((errornodo = ExisteNode(nodo)) == ERROR) {
        printf("\nEl nodo no existe en el mapa de conexión.\n");
        return ERROR;
    }
    if (errornodo == -2) {
        printf("\nEl archivo de mapeo no existe.\n");
        return ERROR;
    }
```

Busca la dirección de red y la dirección de nodo de la estación remota en el archivo de mapeo, sino lo encuentra se produce error.

```
    /* Verifica si el nodo es el nodo de la estación local */
    if (MapNodeNameToAddress(nodo, dnetwork, dnodo) == ERROR)
        return ERROR;
```

Controla que la estación ingresada sea remota, para esto compara las direcciones de red y nodo de las estaciones en juego.

```
    if (!memcmp((BYTE *) networkAddress,(BYTE *) dnetwork,LONGNET) &&
       !memcmp((BYTE *) networkAddress+LONGNET,(BYTE *) dnodo,
                          LONGNODE)) {
        printf("\nEl nodo debe corresponder a otra estación.\n");
        return ERROR;
    }
```

Divide la fuente ingresada en disco, directorio, archivo y extensión para su validación. En la variable flags se tienen los componentes de la fuente.

```
/* Divide la fuente */
if (fnsplit(fuente, drive, dir, file, ext, &flags ) == ERROR)
      return ERROR;
```

Controla que el disco fuente exista, si así no lo hiciere muestra el mensaje de error.

```
/* Verifica que el drive fuente esté
** en el segundo argumento en el comando
*/
if(!(flags & DRIVE)) {
      printf("\nEl disco fuente es obligatorio.\n");
      return ERROR;
}
```

Valida que el disco sea una letra, sino muestra el mensaje de error.

```
/* Verifica que el drive sea una letra */
if (!isalpha(drive[0])) {
      printf("\nEl disco fuente es incorrecto.\n");
      return ERROR;
}
```

Verifica que se haya ingresado el directorio fuente, sino considera error.

```
/* Verifica que el directorio fuente esté en el segundo argumento
** en el comando
*/
if(!(flags & DIRECTORY)) {
      printf("\nEl directorio fuente es obligatorio.\n");
      return ERROR;
}
```

Analiza la sintaxis del directorio, si es incorrecta muestra mensaje de error.

```
/* Verifica que el directorio no tenga errores de sintaxis */
if (VerifPath(dir) == ERROR) {
      printf("\nEl directorio fuente es incorrecto.\n");
      return ERROR;
}
```

Si no se ingresó ningún nombre de archivo, se considera que se requieren todos los archivos y subdirectorios que componen el directorio fuente (*.*).

```
/* Verifica que el nombre del archivo sea correcto */
if(!(flags & FILENAME)) {
      /* si no se ingresa nombre de archivo se asumen todos
      ** los archivos del directorio
      */
      strcat(fuente, "*.*");
}
```

Si se ingresó el nombre de algún archivo verifica que sea correcto sintáctimente. Considera que puede contener wildcards.

```
else {
      if (VerifNbreArch(file, 1) == ERROR) {
            printf("\nEl nombre del archivo fuente es incorrecto.\n");
            return ERROR;
      }
}
return OKAY;
}
```

Variables Globales para Conexión

La variable cant_rcv acumula la cantidad de ECBs recibidos pero que aún no han sido analizados.

```
int           cant_rcv = 0;
```

Array de punteros a ECBs. La cantidad de ECBs está dada por el define de VENTANA, en esta caso 5. Estos ECBs son aquellos que se ponen a escuchar la llegada de paquetes de una conexión. Ver sección 6.3.2 Recepción de paquetes.

```
ECB           *RcvECB[VENTANA];      /* array de punteros a ECBs */
```

Las variables lRcvECB e iRcvECB son índices al array RcvECB. La primera indica el siguiente lugar de memoria a donde se guardará el ECB que será recibido. La segunda indica el lugar de memoria donde se aloja el siguiente ECB a procesar.

```
int             lRcvECB = 0,     /* Indice a ECBs que llegan */
                iRcvECB = 0;     /* Indice a ECBs leidos     */
```

La variable Socket es el número de socket sobre el cual la aplicación efectuará la comunicación.

La varible dest_socket contiene el número de socket sobre el cual estará escuchando el programa residente en la estación remota.

```
WORD            Socket = INVITE_SOCKET;  /* Socket de la aplicación*/
WORD            dest_socket= CHAT_SOCKET;/* Socket destino */
```

ConnectionID es el número de identificación de la conexión local.

```
WORD            ConnectionID;  /* Identificador de conexión de la
                                  estación local */
```

La variable control vale 1 si se presionó Ctrl_Break y 0 lo contrario.

```
short           control = 0; /* controla Ctrl-Break */
```

Variables para Datos de Información

Las siguientes variables se usan para visualizar la información por pantalla.

La variable linea indica la próxima línea de la pantalla en la que se escribe la información de un archivo o subdirectorio.

La variable col indica la próxima columna de la pantalla en la que se escribe el nombre de un archivo o subdirectorio.

La variable bfiles hace referencia al total de la cantidad de bytes de los archivos desplegados.

Por último la variable nfiles es el total de los archivos mostrados.

```
/** Variables para comando **/
short  linea, col;
long   bfiles, nfiles;
```

Función PDir

Establece la conexión con el nodo y llama a las rutinas que despliegan la información obtenida.

Entradas : - **nodo**, nombre de la estación remota.

- **fuente**, compuesto por disco-directorio-archivo.

- **opcion**, indica las opciones de formato para desplegar la información.

Salidas : - ERROR u OKAY según corresponda.

```
static int PDir(char *nodo, char *fuente, unsigned int opcion)
{
```

La variable terminar indica si la conexión terminó normalmente mientras que problema indica que terminó por algún problema que se presentó.

```
    BYTE        terminar, /* Terminó la conexión normalmente   */
                problem;  /* Terminó la conexión con problemas */
```

PoolSPX es un puntero a la cabecera de un paquete de SPX.

```
    SPXHeader       *PoolSPX;
```

Las variables dest_network y dest_nodo contienen las direcciones de red y nodo de la estación remota.

```
    BYTE            dest_network[LONGNET], dest_nodo[LONGNODE];
```

En la estructura pedido se guardan todos los datos acerca de la información requerida de la otra estación.

```
    COMANDO         pedido;
```

En dato se guardan los mensajes de error que son envíados por la estación remota y que deben ser desplegados por pantalla. La longitud de dato está dada por SPXDATOS, máxima cantidad de bytes que puede contener un paquete SPX como dato.

```
    char            dato[SPXDATOS];
```

Ptr es un puntero al fragmento de datos de un paquete. Permite reconocer los datos no ya como un conjunto de bytes sino como un conjunto de estructuras FIND (que contienen la información de archivos y subdirectorios).

La variable info es un array de la misma estructura y es usada para almacenar dicha información.

```
struct FIND     *Ptr, info[12];
```

La variable elem contiene la cantidad de archivos y subdirectorios enviados. Indicará la cantidad de elementos con información que contendrá el array info.

La variable size hace referencia a la longitud del fragmento de datos del paquete.

```
long            elem, size;
```

Las siguientes variables son usadas por la API SPXInicialize para dar información acerca del release del protocolo, máxima cantidad de conexiones y cantidad de conexiones disponibles por la estación.

```
BYTE            mayor, menor;
WORD            maxconex, dispconex;
```

Verifica si las rutinas de IPX están instaladas.

```
if (IPXInitialize() == 0x0) {
      printf("\nIPX no fue instalado.\n");
      return ERROR;
}
```

Verifica si las rutinas de SPX están instaladas.

```
if (SPXInitialize(&mayor, &menor, &maxconex, &dispconex) == 0x0){
      printf("\nSPX no fue instalado.\n");
      return ERROR;
}
```

Abre el socket de la aplicación como temporario (se cierra cuando termina la aplicación). Si ya fue abierto muestra mensaje de error.

```
/* Abre el socket de la aplicación como temporario */
if (IPXOpenSocket((BYTE *) &Socket, TEMPORARY_SOCKET) != OKAY) {
      printf("\nEl socket no fue abierto.\n");
      return ERROR;
}
```

LLama a la función Escuchar() para que ponga a escuchar en el Socket la cantidad de paquetes dado por VENTANA, en este caso 5. También le envía el puntero a la función ReceiveESRHandler() para que actúe como rutina de servicio de evento cuando algún paquete es recibido. Si se produce algún error termina.

```
/* Inicializa ECBs para escuchar */
if (Escuchar(VENTANA, Socket, ReceiveESRHandler) == ERROR)
      return ERROR;
```

LLama a la función EstablecerConexion() para que envíe un paquete de establecer conexión a la estación remota. Le pasa como parámetros el nombre del nodo, el socket local y el de la estación destino. Si no se puede establecer conexión termina.

```
/* Establece conexión */
if (EstablecerConexion(nodo, Socket, dest_network, dest_nodo,
                                    dest_socket, &ConnectionID) == ERROR)
      return ERROR;
```

Debido a que la conexión está establecida se debe controlar el manejo de Ctrl-C y Ctrl-Break, pues si así no se hiciera el programa abortaría sin terminar la conexión.

La función que pasará a manejar el Ctrl-Break es CtrlBreak1, la variable global controlc_handler apunta a la misma y luego se activa al llamar a la función controlc_open().

```
/* Instala la rutina CtrlBreak1 como
** interrupción de Ctrl-Break
*/
_controlc_handler = CtrlBreak1;
controlc_open();
```

Se inicializa el paquete que contiene el pedido con el comando ("PDIR") y la fuente que se requiere.

```
/* Inicializa paquete de pedido */
problem = OKAY;
strcpy(pedido.comando, "PDIR");
strcpy(pedido.origen,  fuente);
```

Antes de enviar el paquete de pedido se verifica si se presionó Ctrl-Break o Ctrl-C. Si no se presionó se envía el paquete por medio de la función MandarMje(), si la misma no devuelve ERROR el paquete fue enviado.

```
/* Envía el paquete de pedido */
if (control || MandarMje(dest_network, dest_nodo,
       (BYTE*) dest_socket, Socket,
        ConnectionID, &pedido, sizeof(pedido),
        0x1) == ERROR)
    problem = ERROR;
```

Una vez que se estableció la conexión y se envió el comando la aplicación queda en espera de recibir paquetes con la información requerida. Sólo podrá dejar de recibir paquetes cuando se produzca algún problema, se haya presionado Ctrl-Break o Ctrl-C o haya llegado un paquete para terminar la conexión.

```
/* Recibe paquetes hasta que haya algún problema,
** se termine la conexión o se presione ctrl-break
*/
terminar = 0;
while ( !problem && !terminar && !control) {
```

Cada vez que llega algún paquete la API SPXListenForSequencedPacket() llama a la rutina del servicio de evento ReceiveESR() la cual incrementa la variable cant_rcv en uno. Luego, se decrementará cuando el paquete haya sido analizado. Por lo tanto que la variable sea mayor que 0 significa que hay paquetes para analizar.

```
    /* Se recibió paquetes de la otra estación */
    if (cant_rcv > 0 ) {
```

El paquete a analizar está apuntado por el elemento iRcvECB del array de punteros RcvECB.

Se debe controlar el campo CompletionCode del paquete a analizar. Si el valor que tiene es 0x00 el paquete llegó bien. En cambio, si el valor que tiene indica que se produjo

error (0xED u otro), el error es mostrado por pantalla y la variable problem toma el valor de ERROR para luego dejar de escuchar paquetes.

```
/* Verifica que el paquete llegó bien */
switch ( RcvECB[iRcvECB]->CompletionCode) {
case 0x00:
      /* Llegó bien */
      break;
case 0xED:
      printf("\nFalla de conexión.\n");
      problem = ERROR;
      break;
default:
      printf("\nError ocurrido tratando de recibir.\n");
      problem = ERROR;
}
```

Si no hubo problemas y no se presionó Ctrl-Break se continúa analizando el paquete.

```
if (!problem && !control) {
```

Se limpia el array dato, éste guardará el mensaje de error cuando se recibe un paquete de tipo "error".

Se hace apuntar a la variable PoolSPX a la cabecera del paquete analizado.

```
memset(dato, '\0', sizeof(dato));
PoolSPX = (SPXHeader *) RcvECB[iRcvECB] ->
    FragmentDescriptor[0].Address[0];
```

En el campo DataStreamType de la cabecera del paquete se encuentra el tipo de paquete recibido. Este campo indica las acciones a seguir.

```
/* Verifica el tipo de paquete */
switch (PoolSPX->DataStreamType) {
```

Es un paquete del tipo "terminar conexión", se setea la variable terminar en verdadero para luego terminar la conexión.

```
case 0xFE:
     /*Es un paquete para terminar la conexión*/
     terminar++;
     break;
```

Es un paquete del tipo "error", se debe desplegar el error producido en el extremo opuesto de la comunicación. Para ello realiza los siguientes pasos:

- En la variable size se guarda la longitud del mensaje de error, la misma está dada por el resto entre la longitud total del paquete (PoolSPX->PacketLength) y la longitud de la cabecera del paquete (RcvECB[iRcvECB]-> FragmentDescriptor[0].Size).
- En la variable dato se coloca el mensaje de error enviado teniendo en cuenta la longitud (size).
- Se despliega el mensaje de error.
- La varible problem es puesta en verdadero.

```
case 0x2:
     /* Es un paquete con mensaje de error */
     size = IntSwap(PoolSPX->PacketLength) -
       RcvECB[iRcvECB]->FragmentDescriptor[0].Size;
     memcpy(dato, (BYTE *) (RcvECB[iRcvECB]->
          FragmentDescriptor[1].Address[0]),size);
     printf("%s", dato);
     problem = ERROR;
     break;
```

Llega un paquete de tipo "dato". Se debe desplegar la información que contiene por la pantalla, para ello:

- La variable Ptr se apunta al fragmento de datos del paquete.
- En la variable size se guarda la longitud del mensaje de error, la misma está dada por el resto entre la longitud total del paquete (PoolSPX->PacketLength) y la longitud de la cabecera del paquete (RcvECB[iRcvECB]-> FragmentDescriptor[0].Size).
- En la variable info se guardan los datos contenidos en el paquete.
- Se llama a la función Mostrar() para que despliegue la información.

```
                case 0x4:
                        /* Es un paquete de datos */
                        /* Muestra los datos recibidos */
                        Ptr= (struct FIND *) (RcvECB[iRcvECB] ->
                         FragmentDescriptor[1].Address[0]);
                        size = IntSwap(PoolSPX->PacketLength) -
                         RcvECB[iRcvECB]->FragmentDescriptor[0].Size;
                        memcpy(info, Ptr, size);
                        elem = (size/sizeof(struct FIND));
                        Mostrar(nodo, fuente, info, elem, opcion);
```

Se pone el paquete analizado a escuchar y son decrementadas las variables iRcvECB y cant_rcv; la primera es el índice al siguiente paquete a analizar (se tiene en cuenta la ventana) y la segunda es la cantidad de paquetes recibidos que quedan por analizar.

```
                        /* Deja el ECB que llegó y
                        ** fue analizado escuchando
                        */
                        SPXListenForSequencedPacket(RcvECB[iRcvECB]);

                        /* Apunta al próximo ECB a leer */
                        if (iRcvECB == VENTANA - 1)
                                iRcvECB = 0;
                        else
                                iRcvECB++;
                        cant_rcv--;
                        break;
```

Se recibió un tipo de paquete desconocido, por lo tanto se considera como un problema.

```
                default:
                        printf ("\nEl tipo de paquete es incorrecto.\n");
                        problem = ERROR;
                }
        }
    }
}
```

Si hubo algún problema o se presionó Ctrl-Break se desactiva la función que lo maneja, se aborta la conexión y se devuelve ERROR.

```
    /* Hubo problemas o se presionó Ctrl-Break,
    ** aborta la conexión
    */
    if (problem || control) {
          controlc_close();
          SPXAbortConnection(ConnectionID);
          return ERROR;
    }
```

Se muestra el total de los archivos desplegados y el total de bytes que ocupan.

```
    /* Muestra el total de archivos desplegados */
    if (!problem)
          MostrarFin(opcion);
```

Se desactiva la función especial para administrar el Ctrl-Break y se retorna OKAY.

```
    controlc_close();
    return OKAY;
}
```

Función CtrlBreak1

Setea la variable global control en verdadero.

Entradas : - **control**, variable global.

Salidas : - **control**, indica si se presionó Ctrl-Break o Ctrl-C.

```
static void _cdecl CtrlBreak1(void)
{
    control = 1;
}
```

Función Mostrar

Despliega la información traída en los paquetes por pantalla teniendo en cuenta la opción de formato que se desea.

Entradas : - **nodo**, nombre de la estación en que se encuentra la fuente.

- **fuente**, compuesto por disco-directorio-archivo que se quiere desplegar.
- **ffblk**, array con los datos de directorio y archivos recibidos desde el nodo.
- **opcion**, opción de formato.
- **nfiles**, variable global.
- **bfiles**, variable global.
- **col**, variable global.
- **linea**, variable global.

Salidas : - **nfiles**, cantidad de archivos desplegados.

- **col**, próxima columna en donde desplegar el archivo.
- **bfiles**, cantidad de bytes de los archivos desplegados.
- **linea**, próxima línea en donde desplegar el archivo.

```
static void Mostrar(char *nodo,
                    char *fuente,
                    struct FIND ffblk[],
                    long cant,
                    unsigned int opcion)
{
    long          i;     /* índice para recorrer ffblk */
    char          drive[MAXDRIVE];
    char          dir[MAXDIR], direct[15];
    char          name[MAXFILE];
    char          ext[MAXEXT];
    short         hora;
    char          uso;
    char          bytes[25];
    static short  first = 1;  /* Primer paquete con información */
    unsigned int  flags;
```

Cuando es el primer paquete a desplegar por pantalla se inicializan todas las variables que sumarizan datos y aquellas que hacen a su visualización.

```
/* Es el primer paquete que despliega */
if (first) {
      nfiles = 0,
      bfiles = 0;
      col    = 0;
      printf("Nodo %s\nDirectorio de %s\n\n", nodo, fuente);
      linea = 3;
      first = 0;
}
```

Recorre el array que contiene los datos ffblk teniendo en cuenta la cantidad de elementos que lo componen cant.

```
/* Despliega archivos */
for (i = 0 ; i < cant; i++ ) {
```

Si la opción de paginar está activada llama a la rutina Continuar() para que si es necesario pagine.

```
      /* Pagina los archivos */
      if ((opcion & SETPAG) == SETPAG) {
            Continuar();
      }
```

La opción de encolumnar no está activa.

```
      /* No encolumna los archivos */
      if ((opcion & SETCOL) != SETCOL) {
            linea++;
```

Con la función fnsplit() se divide el nombre del archivo en sus componentes disco-camino-nombre-extensión.

```
            /* Divide el nombre del archivo en sus componentes */
            (void) fnsplit(ffblk[i].name, drive, dir, name, ext,
                        &flags);
```

Si se trata de un directorio se desplegará el nombre, la extensión y la indicación de que se trata de un directorio (<DIR>).

```
/* Es un directorio */
if (ffblk[i].attribute == FA_DIREC) {
      printf("%-8.8s %-3.3s  <DIR>  %10.10s  ", name,
               ext+1, " ");
}
else {
```

Si no es un archivo se mostrará el nombre, extensión y tamaño en bytes.

```
      printf("%-8.8s %-3.3s  %5.5s  ",name, ext+1, " ");
      ltoa(ffblk[i].size, bytes, 10);
      printf("%10s  ", bytes);
}
```

Se muestra la última fecha de actualización con el formato mm-dd-nn.

```
printf("%0.2d-",  (ffblk[i].date << 7) >> 12 );/*mes*/
printf("%0.2d-",  (ffblk[i].date << 11) >> 11);/*día*/
printf("%u  ",    (ffblk[i].date >> 9) + 80);  /*año*/
```

Se despliega la hora de última actualización del archivo teniendo en cuenta si es pasado el meridiano (p) o antes del meridiano (a).

```
hora = ffblk[i].time >> 11;                    /*hora*/

/* Considera la hora para desplegar a(m) o p(m) */
switch (hora) {
case 24 :
      hora -= 12;
      uso  = 'a';
      break;
case 12 :
      uso = 'p';
      break;
default :
      if ( hora > 12 && hora < 24) {
            hora -= 12;
            uso = 'p';
      }
      else
            uso = 'a';
}
```

Muestra los minutos correspondientes a la hora de última actualización.

```
            printf("%2d:", hora);
            printf("%0.2d%c\n", (ffblk[i].time << 5) >> 10,
                    uso);     /* min. */
      }
```

La opción de encolumnar está activa.

```
      else {
            /* Encolumna los archivos */
```

Si se trata de un directorio se desplegará el nombre entre corchetes.

```
            /* Es un directorio */
            if (ffblk[i].attribute == FA_DIREC) {
                  sprintf(direct, "[%s]", ffblk[i].name);
                  printf("%-14.14s", direct);
            }
```

Si no es un directorio lo desplegará sin corchetes.

```
            else
                  printf("%-14.14s", ffblk[i].name);
```

Si se llegó a la máxima cantidad de columnas se debe continuar la siguiente línea y comenzar en la primer columna. Si no los nombres de archivos y directorios serán separados con un tab.

```
                  if (col == MAXCOL) {
                            printf("\n");
                            col = 0;
                            linea++;
                   }
                   else {
                            printf("\t");
                            col++;
                   }
            }
```

Se suman la cantidad que corresponde al tamaño de los archivos mostrados y la cantidad de archivos.

```
            /* Total de archivos y de bytes que ocupan */
            bfiles += ffblk[i].size;
            nfiles++;
        }
}
```

Función Continuar

Considera si se llegó a la cantidad máxima de líneas de la pantalla, si esto ocurrió espera que se presione una tecla para continuar.

Entradas : - **linea**, variable global.

Salidas : - **linea**, próxima.

```
static void Continuar(void)
{
    if (linea == MAXLINE) {
            linea = 0;
            printf("Presione una tecla para continuar\n");
            while (!bioskey(0)) ;
    }
}
```

Función MostrarFin

Muestra el total de archivos desplegados y la cantidad de bytes que ocupan.

Entradas : - **opcion**, opción ingresada.

- **nfiles**, variable global, cantidad de archivos.
- **bfiles**, variable global, cantidad de bytes.

Salidas : - Ninguna.

```
static void MostrarFin(unsigned int opcion)
{
```

Si se ingresó la opción de paginar se llama a la función Continuar() para considerarla.

```
    if ((opcion & SETPAG) == SETPAG)
        Continuar();
```

Se imprime la cantidad total de archivos desplegados y su total en bytes.

```
    printf("\n\t%ld Archivo(s)\t%ld bytes", nfiles, bfiles);
}
```

6.4.8 Desarrollo del Comando PCOPIA (programa pcopia.c)

Permite copiar un archivo desde una estación remota a la estación local y viceversa.

Para ello establece una conexión entre las estaciones utilizando los protocolos IPX/SPX de Novell. Mediante esta conexión se obtendrá la información que contiene dicho archivo y se copiará en el disco que corresponda.

Recibirá como argumentos de entrada el nombre asignado al nodo de la estación remota, el camino absoluto del archivo y el nombre completo del mismo; también se deberá ingresar el nombre de la estación local y opcionalmente el camino y nombre del archivo, si éstos últimos no son ingresados se copia en el directorio corriente con el nombre que posee en la estación remota.

Recordemos la sintáxis del comando:

"PCOPIA <nodo>:<disco>:\<camino>\<archivo> A
<nodo>:<disco>:\<camino>\<archivo>"

Devolverá OKAY (0) si el comando se pudo ejecutar correctamente y ERROR (-1) si sucedió lo contrario.

Para mayor información sobre las funciones externas usadas consultar las secciones 6.4.10 (ipxspx.c), 6.4.11 (conexion.c) y 6.4.12 (dirarch.c).

Includes

Includes provistos por el compilador.

```
#include <stdio.h>
#include <dos.h>
#include <string.h>
#include <direct.h>
#include <bios.h>
#include <controlc.h>
#include <disp.h>
#include <stat.h>
```

Includes correspondientes a módulos desarrollados especialmente para las aplicaciones.

```
#include "ipxspx.h"
#include "dirarch.h"
#include "conexion.h"
```

Definiciones de Constantes

Se define la cantidad de paquetes que pueden ser recibidos o enviados por la estación.

```
#define VENTANA         5  /* Cantidad de paquetes que puede contener
                              la ventana */
```

Se definen las opciones de copias posibles.

```
#define NADA            0  /* La copia es indefinida */
#define DEOTRONODO      1  /* La copia es de otro nodo a este */
#define DEESTENODO      2  /* La copia es de este nodo a otro */
```

Se define la cantidad máxima de paquetes recibidos o enviados (según corresponda) por línea que serán mostrados por pantalla.

```
#define PQ              50 /* Cantidad de paquetes por línea */
```

Se define la sintaxis del comando para usarla en los mensajes de error.

```
#define SINTAX          "PCOPIA <nodo>:<disco>:\\<camino>\\<archivo> A
                         <nodo>:<disco>:\\<camino>\\<archivo>"
```

Prototipos de Funciones Privadas

Función que despliega el help del comando.

```
static void  Help (void);
```

Función que verifica si los argumentos ingresados en el comando son correctos.

```
static int  VerificarArg  (char  **argv,
                           short *opcion,
                           char  *nodo1,
                           char  *nodo2,
                           char  *fuente,
                           char  *destino);
```

Función que copia un archivo desde una estación remota a la estación local.

```
static int  CopiarArchivoRx(char *nodo,
                           char *fuente,
                           char *destino);
```

Función que copia un archivo desde la estación local a la estación remota.

```
static int  CopiarArchivoTx(char *nodo,
                           char *fuente,
                           char *destino);
```

Función que arma los paquetes con la información contenida en un archivo del disco local y lo envía a la estación remota.

```
static int  EjecutarCopy  (char path[],
                           long *larch,
                           BYTE Socket_dest[LONGSOCK],
                           BYTE Network_dest[LONGNET],
                           BYTE Nodo_dest[LONGNODE],
                           WORD Socket,
                           WORD ConnectionID);
```

Función que cierra el archivo que se está copiando en la estación local y lo borra.

```
static void Problem (char *destino);
```

Función que permite controlar el Ctrl-Break.

```
static void _cdecl CtrlBreak1(void);  /*Handler ctrl_break
                                       sin conexión*/
static void _cdecl CtrlBreak2(void);  /* Handler ctrl_break
                                       con conexión */
static void _cdecl CtrlBreak3(void);  /* Handler ctrl_break
                                       con conexión */
static void CtrlBreak(void);
```

Función que pone a escuchar al paquete que ya fue analizado.

```
static void EscucharPaquete(ECB *RcvECB);
```

Función que preserva los punteros a los paquetes recibidos.

```
void ReceiveESR(ECB *ESR_ECB);
```

Prototipo de Funciones Externas

Función externa que invoca a la rutina del servicio de evento ReceiveESR.

```
extern      void    ReceiveESRHandler ();  /* en Assembler */
```

Programa Principal

Se encarga de llamar a las rutinas que realizan los controles de los argumentos ingresados y la rutina que ejecuta las acciones necesarias para copiar el archivo desde una estación remota a la estación local o viceversa.

Entradas : - **argc**, número de argumentos con los que se invoca el comando en el promt.

- **argv[]**, vector de punteros a cadenas de caracteres que contienen los argumentos con los que fue invocado.

Salidas : - ERROR u OKAY según corresponda.

```
main(int argc, char *argv[])
{
      char    nodo1[10], nodo2[10];  /* nombre de nodos */
      char    fuente[MAXPATH+10], destino[MAXPATH+10];
      short   opcion;          /* indica desde dónde es la copia */
```

Si se ingresó sólo el comando o el comando con el argumento '?' se llama a la función (Help) que despliega la ayuda del mismo.

```
pcopia
pcopia ?
```

```
      /* Help del comando */
      if (argc == 1 || (argc == 2 && *argv[1] == '?')) {
            Help();
            return OKAY;
      }
```

El comando sólo puede tener 4 argumentos, el primero es el nombre del comando (pcopia), el segundo la fuente que se desea copiar, el tercero el caracter 'A' perteneciente al formato del comando y el cuarto el destino que se le dará a la fuente. En caso de que el número de argumentos sea distinto se produce un error de sintaxis.

```
      /* La cantidad de argumentos es incorrecta */
      if (argc != 4) {
            printf("\n%s\n %s\n", "Error de sintaxis:", SINTAX);
            return ERROR;
      }
```

Verifica que los argumentos no contengan errores (función VerificarArg()), si los tiene se termina el programa.

```
      /* Verifica la fuente y el destino */
      if (VerificarArg(argv, &opcion, nodo1, nodo2, fuente, destino)
                                          == ERROR)
            return ERROR;
```

Si se quiere copiar un archivo de una estación remota a la estación local se invoca a la función CopiarArchivoRx(), en caso contrario se invoca a la función CopiarArchivoTx(). Si alguna retorna ERROR el programa termina con error.

```
        /* Según los datos ingresados copia el archivo */
        if (opcion == DEOTRONODO) {
              /* Copia el archivo desde la otra estación */
              if (CopiarArchivoRx(nodo1, fuente, destino) == ERROR)
                    return ERROR;
        }
        else {
              /* Copia el archivo desde esta estación */
              if (CopiarArchivoTx(nodo2, fuente, destino) == ERROR)
                    return ERROR;
        }
```

El programa termina correctamente, se pudo efectuar la copia.

```
        return OKAY;
}
```

Función Help

Se encarga de desplegar la ayuda del comando.

Entradas : - Ninguna.

Salidas : - Ninguna.

No tiene entradas ni salidas.

```
static void Help(void)
{
```

Para visualizar la ayuda se utiliza el Display Package provisto por el compilador. Este permite hacer las I/O de caracteres por pantalla en forma más rápida.

En la ayuda se despliega el modo en que se invoca el comando, una breve descripción de su función y la opción que se puede usar.

```
        disp_open();
        disp_move(0,0);
        disp_eeop();
        disp_box(0, DISP_NORMAL, 3, 0, 21, 79);
        disp_move(5, 3);
        disp_printf("Sintaxis:");
        disp_move(7, 1);
```

```
        disp_printf("%s", SINTAX);
        disp_move(9, 3);
        disp_printf("Descripción:");
        disp_move(11, 0);
        disp_printf("\tCopia un archivo de un nodo a otro.\n\n");
        disp_printf("\tEl nombre de los nodos es obligatorio.\n");
        disp_printf("\tEl resto de los argumentos será obligatorio dependiendo de cuál\n
                    \tsea el nodo que envíe o reciba el archivo.\n");
        disp_move(17,3);
        disp_printf("Opción:");
        disp_move(19,3);
        disp_printf("\t? : Ayuda");
        disp_move(21, 0);
        disp_close();
}
```

Se visualiza por pantalla:

```
  Sintaxis:

pcopia <nodo>:<disco>:\<camino>\<archivo> A <nodo>:<disco>:\<camino>\<archivo>

  Descripción:

       Copia un archivo de un nodo a otro.

       El nombre de los nodos es obligatorio.
       El resto de los argumentos será obligatorio dependiendo de cuál
       sea el nodo que envíe o reciba el archivo.

  Opción:

       ? : Ayuda
```

Función VerificarArg

Verifica que la fuente y el destino, formados por nodo - disco - directorio - archivo, sean correctos.

Entradas : - **argv**, argumentos con los que fue invocado.

Salidas : - **opcion**, indica desde qué nodo se toma el archivo fuente.

- **nodo1**, nodo en que se encuentra la fuente.
- **nodo2**, nodo en que se encuentra el destino.
- **fuente**, compuesto por disco - directorio - archivo que se quiere copiar.
- **destino**, compuesto por disco - directorio - archivo en el que se quiere copiar.
- ERROR u OKAY según corresponda.

```
static int  VerificarArg(char  **argv,
                         short *opcion,
                         char  *nodo1,
                         char  *nodo2,
                         char  *fuente,
                         char  *destino)
{
      char    nodofuente[MAXPATH+10], nododest[MAXPATH+10];
      char    rta[2], comando[3];
      char    *ptr;
      char    drive[MAXDRIVE], drivedest[MAXDRIVE];
      char    dir[MAXDIR], dirdest[MAXDIR];
      char    file[MAXFILE+1], filedest[MAXFILE+1];
      char    ext[MAXEXT+1], extdest[MAXEXT+1];
      unsigned int att, flags, flagsdest;
      BYTE    networkAddress[LONGNET+LONGNODE], dnetwork[LONGNET],
              dnodo[LONGNODE];
```

Inicializa la variable opcion.

```
      *opcion = NADA;
```

Controla que se haya ingresado como tercer argumento "A", sino considera error.

```
      /* Verifica si "A" está en el comando */
      ToUpper(argv[2], comando);
      if (strcmp(comando, "A")) {
            printf("\n%s\n %s\n", "Error de sintaxis:", SINTAX);
            return ERROR;
      }
```

Busca la dirección de red y de nodo de la estación local por medio de la API de IPX.

```
/* Obtiene la dirección de la estación local */
IPXGetInternetworkAddress(networkAddress);
```

Separa el nodo fuente del resto del primer argumneto (lo guarda en la variable nodo1 y el resto en la variable fuente). Lo distingue buscando la primera aparición del caracter ':', si éste no se encuentra se produce un error, pues es obligatorio.

```
/* Verifica que el nodo fuente sea un argumento
** en el comando
*/
ToUpper(argv[1], nodofuente);
ptr = strchr(nodofuente, ':');
if (ptr) {
      memset(nodo1, '\0', 10);
      strncpy(nodo1, nodofuente, ptr-nodofuente);
      strcpy(fuente, ptr+1);
 }
 else {
      printf("\nEl nodo fuente es obligatorio.\n");
      return ERROR;
 }
```

Busca la dirección de red y la dirección de nodo de la estación fuente en el archivo de mapeo, si no la encuentra se produce error.

```
/* Verifica si el nodo fuente es el nodo de la estación
** sobre la que se está trabajando.
** La copia que se realizará es de este nodo a otro.
*/
if (MapNodeNameToAddress(nodo1, dnetwork, dnodo) == ERROR)
      return ERROR;
```

Si las direcciones de red y nodo del nodo *fuente (nodo1)* coinciden con las direcciones de la estación local: se desea copiar de éste nodo a un nodo remoto. Por lo tanto la varible opcion toma el valor que le corresponde DEESTENODO.

```
if (!memcmp((BYTE *) networkAddress,(BYTE *) dnetwork, LONGNET)
   && !memcmp((BYTE *) networkAddress+LONGNET,(BYTE *) dnodo,
                LONGNODE))
      *opcion = DEESTENODO;
```

Separa el nodo destino del resto del cuarto argumento (lo guarda en la variable nodo1 y lo que resta en la varible destino). Lo distingue buscando la primera aparición del caracter ':', si éste no se encuentra se produce un error, pues es obligatorio.

```
/* Verifica que el nodo destino sea un argumento en el comando */
ToUpper(argv[3], nododest);
ptr = strchr(nododest, ':');
if (ptr) {
      memset(nodo2, '\0', 10);
      strncpy(nodo2, nododest, ptr-nododest);
      strcpy(destino, ptr+1);
 }
 else {
      printf("\nEl nodo destino es obligatorio.\n");
      return ERROR;
 }
```

Busca la dirección de red y la dirección de nodo de la estación destino en el archivo de mapeo, si no la encuentra se produce error.

```
/* Verifica si el nodo destino es el nodo de la estación
** sobre la que se está trabajando
** La copia que se realizará es de otro nodo a éste.
*/
if (MapNodeNameToAddress(nodo2, dnetwork, dnodo) == ERROR)
      return ERROR;
```

Compara las direcciones de red y nodo de la estación destino (nodo2) con las direcciones de la estación local, si coinciden, entonces se desea copiar desde la estación remota a ésta. Por lo tanto la varible opcion debe tomar el valor que le corresponde DEOTRONODO. Pero si la variable opcion ya tiene el valor DEESTENODO, significa que el nodo1 y el nodo2 son iguales, se produce un error.

```
if (!memcmp((BYTE *) networkAddress,(BYTE *) dnetwork, LONGNET)
   && !memcmp((BYTE *) networkAddress+LONGNET,(BYTE *) dnodo,
                    LONGNODE)) {

      if (*opcion == DEESTENODO) {
            printf("\nEl nodo fuente y destino son iguales.\n");
            return ERROR;
      }
```

```
        else if (*opcion == NADA) {
                *opcion = DEOTRONODO;
        }
}
```

Si después de comparar las direcciones de ambos nodos la variable opcion sigue teniendo el valor con que se la inicializó, entonces ninguno de los nodos ingresados corresponde a la estación local, por lo tanto se produce un error.

```
if (*opcion == NADA) {
        printf("\nLos nodos son incorrectos.\n");
        return ERROR;
}
```

Divide la fuente ingresada en disco, directorio, archivo y extensión para su validación. En la variable flags se tienen los componentes de la fuente.

```
/** Comienza Verificación de la FUENTE **/

/* Divide la fuente */
if (fnsplit(fuente, drive, dir, file, ext, &flags ) == ERROR)
        return ERROR;
```

Controla que no se hayan ingresado caracteres wildcards ('?' o '*'), si esto pasa se produce un error.

```
if (flags & WILD) {
        printf("\nNo se puede usar wildcards.\n");
        return ERROR;
}
```

Si se quiere copiar desde la estación remota a la estación local el disco fuente es obligatorio, por lo tanto si *no se* ingresa se considera error.

```
/* Verifica que el drive fuente este en el segundo argumento
** en el comando
*/
if((*opcion == DEOTRONODO) && !(flags & DRIVE)) {
        printf("\nEl disco fuente es obligatorio.\n");
        return ERROR;
}
```

Cuando es ingresado el disco se controla que sea una letra.

```
else if (flags & DRIVE) {
      /* Si se ingresa drive verifica que
      ** comience con una letra
      */
      if (!isalpha(drive[0])) {
            printf("\nEl disco fuente es incorrecto.\n");
            return ERROR;
      }
}
```

Al querer copiar desde el nodo remoto el directorio fuente es un dato obligatorio, por lo tanto el hecho de no ingresarlo es considerado error. Si por el contrario se ingresó se controla su sintaxis mediante la rutina VerifPath().

```
/* Verifica que el directorio fuente esté
** en el segundo argumento del comando
*/
if (*opcion == DEOTRONODO) {
      /* El directorio fuente está en el otro nodo */
      if (!(flags & DIRECTORY)) {
            printf("\nEl directorio fuente es obligatorio.\n");
            return ERROR;
      }
      else {
            /*Verifica que el directorio no tenga
            ** errores de sintaxis
            */
            if (VerifPath(dir) == ERROR) {
                  printf("\nEl directorio fuente es incorrecto.\n");
                  return ERROR;
            }
      }
}
```

Si se quiere copiar desde el nodo local al nodo remoto y se ingresó el directorio, se verifica que el mismo exista por medio de la función dir_exists().

```
else if (flags & DIRECTORY) {
      /* El directorio fuente está en el nodo.
      ** Se puede verificar si el directorio fuente existe
      */
```

```
        sprintf(fuente, "%s%s", drive, dir);
        if (dir_exists(fuente) == ERROR) {
              printf("\nEl directorio fuente no existe.\n");
              return ERROR;
        }
  }
```

El archivo fuente debe existir, si no se considera error.

```
  /* Verifica que el archivo fuente esté en el segundo argumento
  ** en el comando.  Debe estar en ambas opciones.
  */
  if(!(flags & FILENAME)) {
        printf("\nEl archivo fuente es obligatorio.\n");
        return ERROR;
  }
```

Se verifica que el nombre del archivo no tenga errores de sintaxis invocando a la función VerifNbreArch().

```
  /* Verifica que el archivo no tenga errores de sintaxis */
  if (VerifNbreArch(file, 0) == ERROR) {
        printf("\nEl nombre del archivo fuente es incorrecto.\n");
        return ERROR;
  }
```

Si la copia es desde la estación local a la remota se puede controlar la existencia del archivo, para esto se llama a la función file_exists().

```
  /* Analiza existencia del archivo fuente si
  ** copia desde éste nodo
  */
  sprintf(fuente, "%s%s%s%s", drive, dir, file, ext);
  if (*opcion == DEESTENODO) {
        if (!file_exists(fuente)) {
              printf("\n%s %s\n", "El archivo fuente no existe.",
                    fuente);
              return ERROR;
        }
        else {
```

Se controlan los atributos del archivo para respetarlos, cuando se trata de un archivo del sistema o un archivo oculto no se pueden copiar.

```
                /* Verifica los atributos del archivo */
                (void) dos_getfileattr(fuente, &att);
                if ((att & FA_HIDDEN) || (att & FA_SYSTEM)) {
                        printf("\nNo puede acceder a %s\n", fuente);
                        return ERROR;
                }
        }
}
```

Separa el destino ingresado en disco, directorio, archivo y extensión para su validación. En la variable flagsdest se tienen los componentes de la misma.

```
/** Comienza Verificación del DESTINO **/

/* Divide el destino */
if (fnsplit(destino, drivedest, dirdest,
                filedest, extdest, &flagsdest) == ERROR)
        return ERROR;
```

Controla que no se hayan ingresado caracteres wildcards ('?' o '*'), si esto pasa se produce un error.

```
if (flagsdest & WILD) {
        printf("\nNo se puede usar wildcards.\n");
        return ERROR;
}
```

Si se quiere copiar desde la estación local a la estación remota el disco destino es obligatorio, por lo tanto si no se ingresa se considera error.

```
/* Verifica que el drive destino esté en el segundo argumento
** del comando, si copia desde este nodo.
*/
if(*opcion == DEESTENODO && !(flagsdest & DRIVE)) {
        printf("\nEl disco destino es obligatorio.\n");
        return ERROR;
}
```

Cuando el disco es ingresado se verifica que se lo designe con una letra.

```
else if(flagsdest & DRIVE) {
        /* Verifica que el drive sea una letra */
        if (!isalpha(drivedest[0])) {
```

```
                printf("\nEl disco destino es incorrecto.\n");
                return ERROR;
        }
}
```

Cuando se quiere copiar desde el nodo local el directorio destino es un dato obligatorio, por lo tanto el hecho de no ingresarlo es considerado error. Si por el contrario se ingresó se controla su sintaxis mediante la rutina VerifPath().

```
if (*opcion == DEESTENODO) {

        /* El directorio destino  está en el otro nodo */
        if (!(flagsdest & DIRECTORY)) {
                printf("\nEl directorio destino es obligatorio.\n");
                return ERROR;
        }
        else {
                /* Verifica que el directorio no tenga
                ** errores de sintaxis
                */
                if (VerifPath(dirdest) == ERROR) {
                        printf("\nEl directorio destino es incorrecto.\n");
                        return ERROR;
                }
        }
}
```

Cuando se trata de copiar desde la estación local se controla la existencia del directorio (dir_exists()), si no existe se produce un error.

```
else if( (flagsdest & DIRECTORY)) {
        /* El directorio destino está en este nodo.
        ** Se puede verificar si el directorio destino existe.
        */
        sprintf(destino, "%s%s", drivedest, dirdest);
        if (dir_exists(destino) == ERROR) {
                printf("\n%s %s\n", "El directorio destino no existe.",
                        dirdest);
                return ERROR;
        }
}
```

Cuando el archivo destino no se ingresa recibe el nombre del archivo fuente. Se considera el ingreso de caminos relativos para adjuntar el nombre del archivo.

```
/* Si no se ingresó el archivo destino toma el nombre
** del archivo fuente
*/
if(!(flagsdest & FILENAME)) {
      /* Considera camino relativos */
      if (!strcmp(dirdest, ".") || !strcmp(dirdest, ".."))
            strcat(dirdest, "\\");
      sprintf(destino, "%s%s%s%s", drivedest, dirdest, file, ext);
}
```

Si, por el contrario, se ingresa entonces se analiza si es correcto por medio de la función VerifNbreArch().

```
else {
      /* Si ingresó el nombre del archivo, puede verificarlo */
      sprintf(destino, "%s%s%s%s", drivedest, dirdest, filedest, extdest);
      if (VerifNbreArch(filedest, 0) == ERROR) {
            printf("\nEl nombre del archivo destino es incorrecto.\n");
            return ERROR;
      }
}
```

Si se está copiando desde la estación remota se controla la existencia del archivo (file_exists()).

```
/* Analiza existencia del archivo destino,
** si se copia desde éste nodo.
*/
if (*opcion == DEOTRONODO && file_exists(destino)) {
```

Se controlan los atributos del archivo para respetarlos, cuando se trata de un archivo del sistema, de un archivo oculto, de un directorio o de un archivo con sólo permiso de lectura no se puede sobreescribir.

```
      /* Verifica los atributos del archivo */
      (void) dos_getfileattr(destino, &att);
      if ((att & FA_HIDDEN) || (att & FA_SYSTEM)) {
            printf("\nNo puede acceder a %s\n", destino);
            return ERROR;
      }
      else if (att & FA_DIREC) {
```

```
                printf("\nNo puede copiar el directorio %s\n",
                        destino);
                return ERROR;
        }
        else if (att & FA_RDONLY) {
                printf("\nNo puede sobreescribir el archivo %s\n",
                        destino);
                return ERROR;
        }
        else {
```

Si los atributos del archivo permiten sobreescribir pregunta al usuario si desea hacerlo. Si el usuario reponde con una "A" el programa termina su ejecución. Si, en cambio, presiona una "S" continúa su ejecución para sobreescribir.

```
                /* El archivo ya existe */
                do {
                        printf("\nEl archivo %s ya existe, sobrescribe o aborta?\n[S/A] : ",
                                destino);
                        gets(rta);
                        ToUpper(rta, rta);
                } while ((strcmp(rta, "S")  && strcmp(rta, "A")));

                /* No copia */
                if (!strcmp(rta, "A"))
                        return ERROR;
        }
    }
```

No se produjo ningún error, la función retorna OKAY.

```
    return OKAY;
}
```

Variables Globales

La variable cant_rcv acumula la cantidad de ECBs recibidos que aún no han sido analizados.

```
int             cant_rcv = 0;  /* ECBs recibidos y no analizados */
```

RcvECB es un array de punteros a ECBs. La cantidad de ECBs está dada por el define de VENTANA, en este caso 5. Estos ECBs son aquellos que escuchan la llegada de paquetes de una conexión. Ver sección 6.3.2 Recepción de paquetes.

```
ECB             *RcvECB[VENTANA];   /* array de punteros a ECBs */
```

Las variables lRcvECB e iRcvECB son índices al array RcvECB. La primera indica el siguiente lugar de memoria a donde se guardará el ECB que será recibido. La segunda indica el lugar de memoria donde se aloja el siguiente ECB a procesar.

```
int             lRcvECB = 0,        /* Indice a ECBs que llegan */
                iRcvECB = 0;        /* Indice a ECBs leidos */
```

La variable Socket es el número de socket sobre el cual la aplicación efectuará la comunicación.

La varible dest_socket contiene el número de socket sobre el cual estará escuchando el programa residente en la estación remota.

```
WORD            Socket = INVITE_SOCKET;   /* Socket de
                                             la aplicación*/
WORD            dest_socket= CHAT_SOCKET; /* Socket destino */
```

ConnectionID es el número de identificación de la conexión local.

```
WORD            ConnectionID;       /* Identificador de conexión
                                       de la estación local */
```

Las varibles arch y archdest corresponden al archivo destino; la primera es el handle del archivo y la segunda es el nombre.

```
int             arch;               /* archivo destino */
char            archdest[MAXPATH];  /* Nombre de archivo destino */
```

La variable control toma el valor de 1 si se presionó el Ctrl_Break y 0 en caso contrario.

```
short           control = 0;                /* contralo Ctrl-Break */
```

Función CopiarArchivoRx

Establece la conexión con el nodo remoto y copia en el archivo destino los paquetes que le son transmitidos desde la fuente.

Entradas : - nodo, nombre de la estación remota.

- fuente, camino absoluto del archivo a copiar.

- destino, camino relativo del archivo destino.

Salidas : - ERROR u OKAY según corresponda.

```
static int CopiarArchivoRx(char *nodo,
                           char *fuente,
                           char *destino)
{
```

La variable terminar indica si la conexión terminó normalmente, mientras que problem indica que finalizó por algún problema que se presentó.

```
    BYTE          terminar, /* Terminó la conexión normalmente */
                  problem;  /* Terminó la conexión con problemas */
```

PoolSPX es un puntero a la cabecera de un paquete de SPX.

```
    SPXHeader     *PoolSPX;
```

Las variables dest_network y dest_nodo contienen las direcciones de red y nodo de la estación remota.

```
    BYTE          dest_network[LONGNET],  /* Dirección de red
                                             destino */
                  dest_nodo[LONGNODE];    /* Dirección de nodo
                                             destino*/
```

En la estructura pedido se guarda todo los datos acerca de la información requerida de la otra estación.

```
    COMANDO       pedido;
```

La variable longitud hace referencia a la longitud del fragmento de datos del paquete recibido. La variable numwr

indica la cantidad de bytes escritos en el archivo. Por último la varible cantpaq indica la cantidad de paquetes recibidos y copiados en el archivo destino.

```
int             longitud, numwr, cantpaq;
```

En dato se guardan los mensajes de error que son envíados por la estación remota y que deben ser desplegados por pantalla. La longitud de dato está dada por SPXDATOS, máxima cantidad de bytes que puede contener un paquete como datos.

```
BYTE            dato[SPXDATOS];
```

Las siguientes variables son usadas por la API SPXInicialize para dar información acerca del release del protocolo, máxima cantidad de conexiones y cantidad de conexiones disponibles por la estación.

```
BYTE            mayor, menor;
WORD            maxconex, dispconex;
```

Abre el archivo destino para escritura solamente, si no existe lo crea y si existe lo sobreescribe. En ambos casos lo deja con permiso de lectura y escritura. En caso de error muestra mensaje de error.

```
/* Abre el archivo destino */
strcpy(archdest, destino);
if ((arch = open(archdest, O_WRONLY | O_CREAT | O_TRUNC, S_IREAD
                                    | S_IWRITE)) == ERROR) {
      printf("\nNo puede crear el archivo %s\n", destino);
      return ERROR;
}
```

Como el archivo destino está abierto se debe controlar el manejo de Ctrl-C y Ctrl-Break, pues si así no se hiciera el programa abortaría y éste archivo quedaría vacío.

La función que pasará a manejar el Ctrl-Break es CtrlBreak1, la variable global controlc_handler apunta a esta última y luego se activa al llamar a la función controlc_open(). Para una mejor información ver sección 6.4.7 Desarrollo del Comando PDIR (programa pdir.c).

```
/* Instala la rutina CtrlBreak1 como
** interrupción de Ctrl-Break
*/
_controlc_handler = CtrlBreak1;
controlc_open();
```

Verifica si las rutinas de IPX están instaladas. Si no lo están muestra el mensaje de error, cierra el archivo destino por medio de la rutina Problem() y finaliza con ERROR.

```
if (IPXInitialize() == 0x0) {
      printf("\nIPX no fue instalado.\n");
      Problem(destino);
      return ERROR;
}
```

Verifica si las rutinas de SPX están instaladas. Si no lo están muestra el mensaje de error, cierra el archivo destino por medio de la rutina Problem() y finaliza con ERROR.

```
if (SPXInitialize(&mayor, &menor, &maxconex,&dispconex) == 0x0) {
      printf("\nSPX no fue instalado.\n");
      Problem(destino);
      return ERROR;
}
```

Controla que no se haya presionado Ctrl-Break. Abre el socket de la aplicación como temporario (se cierra automáticamente cuando termina la aplicación). Si ya fue abierto muestra mensaje de error y cierra el archivo destino.

```
/* Abre el socket de la aplicación como temporario */
if (control || IPXOpenSocket((BYTE *) &Socket, TEMPORARY_SOCKET)
                != OKAY) {
      printf("\nEl socket no fue abierto.\n");
      Problem(destino);
      return ERROR;
}
```

Controla que no se haya presionadao Ctrl-Break. LLama a la función Escuchar() para que ponga a escuchar en el Socket la cantidad de paquetes dado por VENTANA, en este caso 5. También le envía el puntero a la función

ReceiveESRHandler() para que actúe como rutina de servicio de evento cuando algún paquete es recibido. Si se produce algún error cierra el archivo y termina.

```
/* Inicializa paquete para escuchar */
if (control || Escuchar(VENTANA, Socket, ReceiveESRHandler) ==
                        ERROR) {
      Problem(destino);
      return ERROR;
}
```

Verifica que no se haya presionado Ctrl-Break. LLama a la función EstablecerConexion() para que envíe un paquete de establecer conexión a la estación remota. Le pasa como parámetros el nombre del nodo, el socket local y el socket de la estación destino. Si no se puede establecer conexión cierra el archivo destino y termina.

```
/* Establece conexión */
if (control || EstablecerConexion(nodo, Socket, dest_network,
                                  dest_nodo, dest_socket,
                                  &ConnectionID) == ERROR) {
      Problem(destino);
      return ERROR;
}
```

Como el archivo destino está abierto se debe controlar el manejo de Ctrl-C y Ctrl-Break, pues si así no se hiciera el programa abortaría y éste archivo quedaría vacío.

Debido a que la conexión está establecida ya no basta con sólo cerrar el archivo destino cuando se presiona Ctrl-Break o Ctrl-C, ahora también se debe cerrar la conexión. Por esta razón se desactiva (controlc_close()) la función CtrlBreak1 y se activa la función CtrlBreak2, la que pasará a manejarlo. Ver sección 6.4.4 Manejo de Control-C.

```
/* Instala la rutina CtrlBreak2 como
** interrupción de Ctrl-Break
*/
controlc_close();
_controlc_handler = CtrlBreak2;
controlc_open();
```

Se inicializa el paquete que contiene el pedido con el comando "RXPCOPIA", recibe copia de otro nodo y la fuente que se requiere.

```
/* Inicializa paquete de pedido */
problem = OKAY;
strcpy(pedido.comando, "RXPCOPIA");
strcpy(pedido.origen,  fuente);
```

Antes de enviar el paquete de pedido se verifica si se presionó Ctrl-Break o Ctrl-C. Si no se presionó se envía el paquete por medio de la función MandarMje(), si la misma no devuelve ERROR el paquete fue enviado.

```
/* Envía el paquete de pedido */
if (control || MandarMje(dest_network, dest_nodo,
                         (BYTE *) dest_socket,
                          Socket, ConnectionID,
                          &pedido, sizeof(pedido), 0x1) == ERROR)
     problem = ERROR;
```

Una vez que se estableció la conexión y se envió el comando la aplicación queda en espera de recibir paquetes que contengan al archivo. Sólo podrá dejar de recibir paquetes cuando se produzca algún problema, se haya presionado Ctrl-Break o Ctrl-C o haya llegado un paquete para terminar la conexión.

```
/* Recibe paquetes hasta que haya algún problema,
** se termine la conexión o se presione ctrl-break
*/
terminar = 0;
cantpaq  = 0;
while ( !problem && !terminar && !control) {
```

Cada vez que llega algún paquete la API SPXListenForSequencedPacket() llama a la rutina del servicio de evento ReceiveESR() la cual incrementa la variable cant_rcv en uno. Luego, se decrementará cuando el paquete haya sido analizado. Por lo tanto, que la variable sea mayor que 0 significa que hay paquetes para analizar.

```
     if (cant_rcv > 0 ) {
```

El paquete a analizar está apuntado por el elemento iRcvECB del array de punteros RcvECB.

Se debe controlar el campo CompletionCode del paquete a analizar. Si el valor que tiene es 0x00 el paquete llegó bien. En cambio, si el valor que tiene indica que se produjo

error (0xED u otro), éste es mostrado por pantalla y la variable problem toma el valor de ERROR para luego dejar de escuchar paquetes.

```
/* Verifica que el paquete llegó bien */
switch ( RcvECB[iRcvECB]->CompletionCode) {
case 0x00:
      /* Llegó bien */
      break;
case 0xED:
      printf("\nFalla de conexión.\n");
      problem = ERROR;
      break;
default:
      printf("\nError ocurrido tratando de recibir.\n");
      problem = ERROR;
}
```

Si no hubo problemas y no se presionó Ctrl-Break se continúa analizando el paquete.

```
if (!problem && !control) {
```

Se limpia el array dato, éste guardará los datos enviados en el paquete.

Se hace apuntar a la variable PoolSPX a la cabecera del paquete analizado.

```
memset(dato, '\0', sizeof(dato));
PoolSPX = (SPXHeader *) RcvECB[iRcvECB]->
                  FragmentDescriptor[0].Address[0];
```

En el campo DataStreamType de la cabecera del paquete se encuentra el tipo de paquete recibido. Este campo indica las acciones a seguir.

```
/* Verifica tipo de paquete */
switch (PoolSPX->DataStreamType) {
```

Es un paquete del tipo "terminar conexión", se setea la variable terminar en verdadero para luego terminar la conexión.

```
case 0xFE:
      /* Es un paquete para terminar la conexión */
      terminar++;
      break;
```

Es un paquete del tipo "error", se despliega el error producido en el extremo opuesto de la comunicación. Para ello realiza los siguientes pasos:

- En la variable longitud se guarda la longitud del mensaje de error, la misma está dada por el resto entre la longitud total del paquete (PoolSPX->PacketLength) y la longitud de la cabecera del paquete (RcvECB[iRcvECB]-> FragmentDescriptor[0].Size).
- En la variable dato se coloca el mensaje de error enviado, éste está en el fragmento descriptor de dato, teniendo en cuenta la longitud (longitud).
- Se despliega el mensaje de error.
- La varible problem es puesta en verdadero.

```
case 0x2:
      /* Es un paquete con mensaje de error */
      longitud = IntSwap(PoolSPX->PacketLength) -
          RcvECB[iRcvECB]->FragmentDescriptor[0].Size;
      memcpy(dato, (BYTE *) (RcvECB[iRcvECB]->
                  FragmentDescriptor[1].Address[0]),
                  longitud);
      printf("%s", dato);
      problem = ERROR;
      break;
```

Llega un paquete de tipo "dato". Se copia en el archivo destino la fracción del archivo fuente contenida en el paquete, para ello ejecuta los siguientes pasos:

- En la variable longitud se guarda la longitud de los datos, la misma está dada por el resto entre la longitud total del paquete (PoolSPX->PacketLength) y la longitud de la cabecera del paquete (RcvECB[iRcvECB]-> FragmentDescriptor[0].Size).
- En la variable dato copia el fragmento de descriptor de dato del paquete.
- Copia dato en el archivo destino.

- Indica en la pantalla la copia de un paquete, cada línea de caracteres '.' corresponden a 50 paquetes.

```
case 0x4:
      /* Es un paquete de datos.
      ** Escribe el paquete de datos en el disco
       */
      longitud = IntSwap(PoolSPX->PacketLength) -
       RcvECB[iRcvECB]->FragmentDescriptor[0].Size;
      memcpy(dato, (BYTE *) (RcvECB[iRcvECB]->
                  FragmentDescriptor[1].Address[0]),
                  longitud);
      if ((numwr = write(arch, dato,
             longitud)) != longitud) {
            printf("\nError al intentar escribir en el disco.\n");
            problem = ERROR;
      }
      /* Imprime por pantalla que
      ** recibe un paquete
      */
      if (!problem) {
            if (!cantpaq)
                  printf("%s\n\t", destino);
            else if ((cantpaq % PQ) == 0)
                  printf("\n\t");
            printf(".");
            cantpaq++;
      }
```

Si no se produjo ningún problema ni se presionó Ctrl-Break se llama a la función EscucharPaquete() que pondrá el paquete analizado a escuchar nuevamente y decrementará el índice al siguiente paquete a analizar y la cantidad de paquetes recibidos que quedan por analizar.

```
      /* Deja el ECB, que llegó y
      ** fue analizado, escuchando
      */
      if (!problem && !control)
            EscucharPaquete(RcvECB[iRcvECB]);
      break;
```

Se recibió un tipo de paquete desconocido, por lo tanto se considera como un problema.

```
                default:
                    printf ("\nEl tipo de paquete es incorrecto.\n");
                    problem = ERROR;
                }
            }
        }
    }
```

Si hubo algún problema aborta la conexión y borra el archivo que estaba copiando.

```
    if (problem) {
        /* Si hubo algún problema aborta la conexión */
        SPXAbortConnection(ConnectionID);
        Problem(destino);
        return ERROR;
    }
```

Si se presiona Ctrl-Break llama a la función CtrlBreak() que de acuerdo al momento en que se produce cerrará el archivo y la conexión o solamente cerrará el archivo.

```
    else if (control) {
        CtrlBreak();
        return ERROR;
    }
```

Si todo funcionó sin problemas se cierra el archivo y se retorna OKAY.

```
    close(arch);
    return OKAY;
}
```

Función CopiarArchivoTx

Establece la conexión con el nodo remoto y le transmite el archivo en paquetes para que lo copie en el archivo destino.

Entradas : - **nodo**, nombre de la estación remota.

- **fuente**, camino absoluto del archivo a copiar.
- **destino**, camino relativo del archivo destino.

Salidas : - ERROR u OKAY según corresponda.

```
static int CopiarArchivoTx(char *nodo,
                           char *fuente,
                           char *destino)
{
```

Las variables locales de esta función son similares a la función CopiarArchivoRx() explicada en el punto anterior.

```
    BYTE          terminar,  /* Terminó la conexión normalmente */
                  problem;   /* Terminó la conexión con problemas*/
    SPXHeader     *PoolSPX;
    BYTE          dest_network[LONGNET], /* Dirección de red
                                            destino */
                  dest_nodo[LONGNODE];   /* Dirección de nodo
                                            destino*/
    BYTE          dato[SPXDATOS];
    BYTE          mayor, menor;
    WORD          maxconex, dispconex;
    COMANDO       pedido;
    long          longitud,    /* Longitud del paquete */
                  larch;       /* Cantidad de bytes a copiar del
                                  archivo */
```

La variable first se utiliza para indicar que se está enviando el primer paquete perteneciente al archivo. Es inicializado en verdadero.

```
    short         first = 1;  /* Es el primer paquete que se envía
                                 del archivo */
```

Abre el archivo fuente para lectura solamente, y permite que otros procesos puedan leerlo. Si no existe o no se puede abrir se produce un error.

```
    /* Abre el archivo fuente */
    if ((arch = open(fuente, O_RDONLY, S_IREAD)) == ERROR) {
          printf("\n%s %s\n", "No se encontró o no puede abrir", fuente);
          return;
    }
    /* Posiciona al principio del archivo */
    lseek(arch, 0L, SEEK_SET);
```

Es necesario controlar que el archivo no esté vacío antes de establecer la conexión.

```
/* Toma la longitud del archivo,
** verifica si está vacío
*/
larch=filesize(fuente);
if (larch == 0) {
      printf("\n%s %s\n", "Archivo vacio,", fuente);
      close(arch);
      return;
}
```

Verifica si las rutinas de IPX están instaladas. Si no lo están muestra el mensaje de error y retorna ERROR.

```
if (IPXInitialize() == 0x0) {
      printf("\nIPX no fue instalado.\n");
      return ERROR;
}
```

Verifica si las rutinas de SPX están instaladas. Si no lo están muestra el mensaje de error y finaliza con ERROR.

```
if (SPXInitialize(&mayor, &menor, &maxconex, &dispconex) == 0x0) {
      printf("\nSPX no fue instalado.\n");
      Problem(destino);
      return ERROR;
}
```

Abre el socket de la aplicación como temporario (se cierra automáticamente cuando termina la aplicación). Si ya fue abierto muestra mensaje de error y finaliza con error.

```
/* Abre el socket de la aplicación como temporario */
if (IPXOpenSocket((BYTE *) &Socket, TEMPORARY_SOCKET) != OKAY) {
      printf("\nEl socket no fue abierto.\n");
      return ERROR;
}
```

Controla que no se haya presionadao Ctrl-Break. LLama a la función Escuchar() para que ponga a escuchar en el Socket la cantidad de paquetes dados por VENTANA, en este caso 5. También le envía el puntero a la función ReceiveESRHandler() para que actúe como rutina de servicio de evento cuando algún paquete es recibido. Si se produce algún error termina.

```
/* Inicializa ECBs para escuchar */
if (Escuchar(VENTANA, Socket, ReceiveESRHandler) == ERROR) {
      return ERROR;
}
```

Se invoca a la función EstablecerConexion() para que envíe un paquete de establecer conexión a la estación remota. Le pasa como parámetros el nombre del nodo, el socket local y el socket de la estación destino. Si no se puede establecer conexión cierra el archivo destino y termina.

```
/* Establece la conexión */
if (EstablecerConexion(nodo, Socket, dest_network,
                       dest_nodo, dest_socket,
                       &ConnectionID) == ERROR) {
      return ERROR;
}
```

Debido a que la conexión está establecida se debe controlar el manejo de Ctrl-C y Ctrl-Break, pues si así no se hiciere el programa abortaría sin terminar la conexión.

La función que pasará a manejar el Ctrl-Break es CtrlBreak3, la variable global controlc_handler apunta a la misma y luego se activa al llamar a la función controlc_open().

```
/* Instala la rutina CtrlBreak3 como
** interrupción de Ctrl-Break
*/
_controlc_handler = CtrlBreak3;
controlc_open();
```

Se inicializa el paquete que contiene el pedido con el comando "TXPCOPIA" y el destino en el que se copiará el archivo.

```
/* Inicializa paquete de pedido */
problem = OKAY;
strcpy(pedido.comando, "TXPCOPIA");
strcpy(pedido.origen,  destino);
```

Antes de enviar el paquete de pedido se verifica si se presionó Ctrl-Break o Ctrl-C. Si no se presionó se envía el paquete por medio de la función MandarMje(), si la misma no devuelve ERROR el paquete fue enviado.

```
/* Envía el paquete de pedido */
if (control || MandarMje(dest_network, dest_nodo,
                    (BYTE *) dest_socket,
                    Socket, ConnectionID,
                    &pedido, sizeof(pedido), 0x1) == ERROR) {
      problem = ERROR;
}
```

Una vez que se estableció la conexión y se envió el comando la aplicación queda en espera de recibir algún paquete que le indique que puede comenzar a enviar el archivo. Mientras está enviando paquetes puede llegar a recibir paquetes con mensajes de error del otro extremo o paquetes de error de la red. Sólo podrá dejar de enviar y recibir paquetes cuando se produzca algún problema, se haya presionado Ctrl-Break o Ctrl-C, haya llegado un paquete para terminar la conexión o la estación remota haya abortado.

```
/* Recibe y envía paquetes hasta que haya algún problema,
** se termine la conexión o se presione ctrl-break
*/
terminar = 0;
while ( !problem && !terminar && !control) {
```

Cada vez que llega algún paquete la API SPXListenForSequencedPacket() llama a la rutina del servicio de evento ReceiveESR() la cual incrementa la variable cant_rcv en uno. Luego, se decrementará cuando el paquete haya sido analizado. Por lo tanto, que la variable sea mayor que 0 significa que hay paquetes para analizar.

```
      if (cant_rcv > 0 ) {
```

El paquete a analizar está apuntado por el elemento iRcvECB del array de punteros RcvECB.

Se debe controlar el campo CompletionCode del paquete a analizar. Si el valor que tiene es 0x00 el paquete llegó bien. En cambio, si el valor que tiene indica que se produjo error (0xED u otro), éste es mostrado por pantalla y la variable problem toma el valor de ERROR para luego dejar de escuchar paquetes.

```
            /* Se recibió paquetes de la otra estación.
            ** Verifica que el paquete llegó bien.
            */
            switch ( RcvECB[iRcvECB]->CompletionCode) {
```

```
case 0x00:
      /* Llegó bien */
      break;
case 0xED:
      printf("\nFalla de conexión.\n");
      problem = ERROR;
      break;
default:
      printf("\nError ocurrido tratando de recibir.\n");
      problem = ERROR;
}
```

Si no hubo problemas y no se presionó Ctrl-Break se continúa analizando el paquete.

```
if (!problem && !control) {
```

Se limpia el array dato, éste guardará el mensaje de error si fuese enviado un paquete de tipo "error".

Se hace apuntar a la variable PoolSPX a la cabecera del paquete analizado.

```
memset(dato, '\0', sizeof(dato));
PoolSPX = (SPXHeader *) RcvECB[iRcvECB]->
                        FragmentDescriptor[0].Address[0];
```

En el campo DataStreamType de la cabecera del paquete se encuentra el tipo de paquete recibido. Este campo indica las acciones a seguir.

```
/* Verifica tipo de paquete */
switch (PoolSPX->DataStreamType) {
```

Es un paquete del tipo "terminar conexión", se setea la variable terminar en verdadero para luego salir del while y terminar la conexión.

```
case 0xFE:
     /* Es un paquete para terminar la conexión */
     terminar++;
     break;
```

Es un paquete del tipo "error", se despliega el error producido en el extremo opuesto de la comunicación. Para ello realiza los siguientes pasos:

- En la variable longitud se guarda la longitud del mensaje de error, la misma está dada por el resto entre la longitud total del paquete (PoolSPX->PacketLength) y la longitud de la cabecera del paquete (RcvECB[iRcvECB]-> FragmentDescriptor[0].Size).
- En la variable dato se coloca el mensaje de error enviado, el cual está contenido en el fragmento descriptor de dato, teniendo en cuenta la longitud (longitud).
- Se despliega el mensaje de error.
- La varible problem es puesta en verdadero.

```
case 0x2:
      /* Es un paquete con mensaje de error */
      longitud = IntSwap(PoolSPX->PacketLength) -
       RcvECB[iRcvECB]->FragmentDescriptor[0].Size;
      memcpy(dato, (BYTE *) (RcvECB[iRcvECB]->
                  FragmentDescriptor[1].Address[0]),
                  longitud);
      printf("%s", dato);
      problem = ERROR;
      break;
```

Llega un paquete de tipo "dato". La estación remota abrió el archivo destino sin problemas y está en condiciones de recibir y copiar los paquetes conteniendo el archivo que se le envíe. Los pasos a seguir son:

- Enviar un fragmento del archivo por medio de la función EjecutarCopy(). Si la misma retorna error, la variable problem toma este valor.
- Controlar si se envío el archivo completo o no. Para esto analiza la variable larch que contiene la cantidad de bytes que aún faltan enviar. Si se envío todo el archivo la variable terminar toma el valor de verdadero.
- Se indica en la variable first que ya se envío el primer paquete conteniendo una fracción del archivo fuente.

```
case 0x4:
      /* El extremo opuesto pudo abrir
      ** el archivo destino y está listo
      ** para recibir paquetes conteniendo
      ** el archivo fuente.
```

```
                    ** Envía el primer paquete.
                    */
                    problem = EjecutarCopy( fuente,
                                            &larch,
                                            PoolSPX->Source.Socket,
                                            PoolSPX->Source.Network,
                                            PoolSPX->Source.Node,
                                            Socket, ConnectionID);
                    if (larch == 0)
                          terminar++ ;
                    first = 0;
```

Si no se produjo ningún problema ni se presionó Ctrl-Break se llama a la función EscucharPaquete( ) que pondrá el paquete analizado a escuchar nuevamente y decrementará el índice al siguiente paquete a analizar y la cantidad de paquetes recibidos que quedan por analizar.

```
                    /* Deja el ECB que llegó y
                    ** fue analizado escuchando
                    */
                    if (!problem && !control)
                          EscucharPaquete(RcvECB[iRcvECB]);
                    break;
```

Se recibió un tipo de paquete desconocido, por lo tanto se considera como un problema.

```
               default:
                    printf ("\nEl tipo de paquete es incorrecto.\n");
                    problem = ERROR;
               }
          }
     }
```

Mientras no se reciban paquetes, se envían los paquetes que contienen las fracciones del archivo fuente siempre y cuando, se haya enviado el primer paquete y aún reste enviar paquetes para completar el archivo.

```
     else {
          /* Se deben enviar los paquetes que restan del
          ** archivo
          */
```

Controla que se haya enviado el primer paquete y la cantidad de bytes, larch, que restan enviar sea mayor que 0.

```
if (!first && larch > 0) {
```

Envía un fragmento del archivo por medio de la función EjecutarCopy(). Si la misma retorna error, la variable problem toma este valor.

```
problem = EjecutarCopy( fuente,
                        &larch,
                        PoolSPX->Source.Socket,
                        PoolSPX->Source.Network,
                        PoolSPX->Source.Node,
                        Socket, ConnectionID);
```

Si se envío todo el archivo la variable terminar toma el valor de verdadero.

```
                /* Se terminó el archivo,
                ** termina la conexión
                */
                if (larch == 0)
                      terminar++ ;
            }
        }
    }
```

Si hubo algún problema o se presionó Ctrl-Break se cierra el archivo fuente y se aborta la conexión devolviendo ERROR.

```
/* Hubo problemas o se presionó Ctrl-Break,
** aborta la conexión
*/
if (problem || control) {
      close(arch);
      SPXAbortConnection(ConnectionID);
      return ERROR;
}
```

Si todo salió bien se cierra el archivo fuente y se envía un paquete de "terminar conexión" y retorna OKAY.

```
    close(arch);
    TerminarConexion(Socket, ConnectionID);
    return OKAY;
}
```

Función Problem

Función que cierra el archivo enviado y lo borra.

Entradas : - **destino**, nombre del archivo.
- **arch**, variable global que indica el handle del archivo.

Salidas : - Ninguna

```
static void Problem(char *destino)
{
    close(arch);
    remove(destino);
}
```

Función CtrlBreak1

Setea la variable global control en 1. Este valor indica que se presionó Ctrl-Break o Ctrl-C después de abrir el archivo destino.

Entradas : - **control**, variable global.

Salidas : - **control**, indica si se presionó Ctrl-Break o Ctrl-C.

```
static void _cdecl CtrlBreak1(void)
{
    control = 1;
}
```

Función CtrlBreak2

Setea la variable global control en 2. Este valor indica que se presionó Ctrl-Break o Ctrl-C después de abrir el archivo destino y establecer la conexión con la estación remota.

Entradas : - **control**, variable global.

Salidas : - **control**, indica si se presionó Ctrl-Break o Ctrl-C.

```
static void _cdecl CtrlBreak2(void)
{
     control = 2;
}
```

Función CtrlBreak3

Setea la variable global control en 3. Este valor indica que se presionó Ctrl-Break o Ctrl-C después de abrir el archivo fuente y establecer la conexión con la estación remota.

Entradas : - **control**, variable global.

Salidas : - **control**, indica si se presionó Ctrl-Break o Ctrl-C.

```
static void _cdecl CtrlBreak3(void)
{
     control = 3;
}
```

Función CtrlBreak

Función que de acuerdo a cuándo se presiona Ctrl-Break (según el valor de la variable control) cierra el archivo, lo borra y aborta la conexión.

Entradas : - **control**, variable global, indica cuándo se presionó Ctrl-Break.

Salidas : - Ninguna

```
static void CtrlBreak(void)
{
      switch(control) {
      case 1:
            Problem(archdest);
            break;
      case 2:
            Problem(archdest);
            SPXAbortConnection(ConnectionID);
            break;
      case 3:
            SPXAbortConnection(ConnectionID);
            break;
      }
}
```

Función EjecutarCopy

Envía un paquete a la estación remota conteniendo una fracción del archivo fuente.

Entradas : - **path**, archivo fuente.
- **larch**, cantidad de bytes que faltan enviar.
- **Socket_dest**, socket destino.
- **Network_dest**, dirección de red destino.
- **Nodo_dest**, dirección de nodo destino.
- **Socket**, socket de la aplicación.
- **ConnectionID**, identificador de conexión.

Salidas : - **larch**, cantidad de bytes que faltan enviar.
- ERROR u OKAY según corresponda.

```
static int EjecutarCopy ( char      path[],
                          long      *larch,
                          BYTE      Socket_dest[LONGSOCK],
                          BYTE      Network_dest[LONGNET],
                          BYTE      Nodo_dest[LONGNODE],
                          WORD      Socket,
                          WORD      ConnectionID)
{
      BYTE          dato[SPXDATOS];/* Contiene los datos a enviar */
```

```
int             problem,        /* Se produjo algún problema */
                numread;/* Indica la cantidad de bytes leídos */
static short    cantpaq = 0; /* Cantidad de paquetes enviados */
```

Inicializa la varible problem indicando que no hay problemas (OKAY) y limpia el array de caracteres dato.

```
problem = OKAY;
/* Deja un fin de cadena para reconocer la longitud */
memset(dato, '\0', SPXDATOS);
```

Lee el próximo bloque de caracteres del archivo asociado con el descriptor de archivo arch según la longitud indicada por SPXDATOS (máxima longitud del campo de fragmento de datos de un paquete SPX). Deja el bloque en el buffer dato. En numread se indica la cantidad de bytes que fueron leídos, el que puede llegar a ser menor que SPXDATOS.

Si numread tiene el valor -1 (ERROR) entonces se produjo un error al intentar leer el archivo:

- En la variable dato se indica el mensaje de error.
- Se despliega el error por pantalla.
- Se envía al nodo remoto el tipo de paquete "error" por medio de la función MandarMje().
- Se indica en la variable problem que hubo un error.

```
/* Lee el archivo, según la longitud del paquete */
if((numread = read(arch, dato, SPXDATOS)) == ERROR) {
      sprintf((char *) dato, "\n%s %s\n", "Error leyendo el archivo",path);
      printf("%s", dato);
      /* Manda el paquete con error */
      (void) MandarMje(Network_dest, Nodo_dest, Socket_dest,
                       Socket, ConnectionID,
                       dato, numread, 0x2);
      problem++;
}
```

El bloque del archivo se leyó correctamente:

- Se decrementa la cantidad de bytes que faltan por enviar.
- Se envía al nodo remoto la fracción del archivo como tipo de paquete "dato", para ello se utiliza la función MandarMje().

- Si al enviar el paquete se produce alguna clase de error se indica en la variable problem que existe algún inconveniente.

```
        else {
                *larch = *larch - numread;
                /* Manda el paquete con datos */
                if (MandarMje(Network_dest, Nodo_dest, Socket_dest, Socket, ConnectionID,
                                dato, numread, 0x4) == ERROR)
                        problem++;
```

Si no se produjeron problemas, se imprime por pantalla el envío del paquete de dato poniendo un caracter '.'. Se despliega el envío de PQ (50) paquetes por línea.

```
                /* Imprime por pantalla que envía un paquete */
                if (!problem) {
                        if (!cantpaq)
                                printf("%s\n\t", path);
                        else if ((cantpaq % PQ) == 0)
                                printf("\n\t");
                        printf(".");
                        cantpaq++;
                }
        }
        return problem;
}
```

Función EscucharPaquete

Deja el paquete que fue analizado escuchando e indica el próximo paquete a analizar.

Entradas : - **RcvECB**, puntero al paquete (ECB) analizado.
- **cant_rcv**, variable global.
- **iRcvECB**, variable global.

Salidas : - **cant_rcv**, cantidad de paquetes recibidos que faltan analizar.
- **iRcvECB**, índice al próximo paquete a ser analizado.

```
static void EscucharPaquete(ECB *RcvECB)
{
```

Se pone el paquete analizado a escuchar y son decrementadas las variables iRcvECB y cant_rcv; la primera es el indice al siguiente paquete a analizar (se tiene en cuenta la ventana) y la segunda es la cantidad de paquetes recibidos que quedan por analizar.

```
        /* Deja el ECB que llegó y fue analizado escuchando */
        SPXListenForSequencedPacket(RcvECB);
        /* Apunta al próximo ECB a leer */
        if (iRcvECB == VENTANA - 1)
              iRcvECB = 0;
        else
              iRcvECB++;
        cant_rcv--;
}
```

6.4.9 Desarrollo de la Aplicación Residente (programa tsrcom.c)

Este programa es el encargado de responder a los comandos (PDIR y PCOPIA) que se ejecutan en una estación de trabajo remota.

El mismo permanece residente en memoria para poder dar respuesta, en cualquier momento, a las solicitudes que le hacen las estaciones remotas.

Utiliza los protocolos IPX/SPX de Novell para establecer una conexión entre las estaciones. Sobre esta conexión transmitirá la información solicitada, o recibirá la enviada por la estación que ejetute el comando.

Si el usuario desea removerlo de memoria deberá ingresar como argumentos de entrada "/r" o "/R".

Devolverá OKAY (0) si el proceso se ejecutó con éxito o ERROR (-1) en caso contrario.

Includes

Includes provistos por el compilador.

```
#include <disp.h>
#include <dos.h>
#include <bios.h>
#include <string.h>
#include <io.h>
#include <stdio.h>
#include <stat.h>
#include <cerror.h>
#include <tsr.h>
```

Includes correspondientes a módulos desarrollados especialmente para las aplicaciones.

```
#include "ipxspx.h"
```

Definiciones de Constantes

Se define la cantidad de paquetes que pueden ser recibidos o enviados por la estación.

```
#define VENT      3
```

Se define la máxima cantidad de caracteres que puede poseer el path.

```
#define MAXPATH   80
```

Se define la máxima cantidad de ffblk en el fragmento de datos del ECB.

```
#define MAXINFO SPXDATOS/sizeof(struct FIND)
```

Se define OK para indicar que se está en condiciones de copiar un archivo.

```
#define OK        1
```

Prototipos de Funciones Privadas

Función que despliga la presentación de la aplicación residente.

```
void Inicio (void);
```

Función que verifica si un archivo ya existe.

```
int  file_exists (char *filename);
```

Función que cierra y borra un archivo.

```
void Problem (char *destino);
```

Función que setea un paquete para conexión y lo coloca a escuchar.

```
void EscucharConexion (WORD Socket);
```

Función que termina una conexión preestablecida.

```
int  TerminarConexion (WORD Socket,
                       WORD ConnectionID);
```

Función que lee y envía un archivo por la red a una estación remota, luego de verificar que no está vacío y que tiene acceso a él.

```
static int  EjecutarCopia  (char path[MAXPATH],
                            BYTE Socket_dest[LONGSOCK],
                            BYTE Network_dest[LONGNET],
                            BYTE Nodo_dest[LONGNODE]);
```

Función que envía un paquete a otra estación de trabajo.

```
int  MandarMje     (BYTE *Socket_dest,
                    BYTE *Network_dest,
                    BYTE *Nodo_dest,
                    void *dato,
                    int  longitud,
                    BYTE tipo);
```

Función que obtiene información perteneciente a un directorio o archivo, si éste existe, arma paquetes con ella y los envía a otra estación.

```
int  EjecutarDir   (char path[MAXPATH],
                    BYTE Socket_dest[LONGSOCK],
                    BYTE Network_dest[LONGNET],
                    BYTE Nodo_dest[LONGNODE]);
```

Función que copia la información obtenida de un directorio o archivo en una estructura.

```
void CopiarInfo    (struct FIND info[],
                    struct FIND *ffblk);
```

Función que registra la llegada de un ECB, colocando su puntero en un array de ECBs todavía no analizados.

```
void ReceiveESR       (ECB *ESR_ECB);
```

Función que intercepta un error de dispositivo dándole el control al programa.

```
int  _far _cdecl myhandler (int *ax,
                            int *di);
```

Funciones Externas

Función externa que maneja el llamado a la rutina ESR cuando se recibe un paquete.

```
extern void ReceiveESRHandler ();  /* en Assembler */
```

Variables Globales

La variable _tsr_timeslice pertenece al TRS Package brindado por el compilador. Mediante su uso el programa se convierte en una tarea background, ingresando a la función popmain 18 veces por segundo.

```
extern  unsigned _tsr_timeslice;
```

La variable _okbigbuf, también, pertenece al TRS Package. Esta recibe el valor 0 para que el programa residente tome sólo la memoria que realmente necesita, liberando el resto.

```
extern  int      _okbigbuf = 0;
```

Representan la combinación especial de teclas llamada hotkey.

```
unsigned        TSR_HOTSHIFT = RSHIFT+LSHIFT;
char            TSR_HOTSCAN  = NO_SCAN;
```

Contendrá los atributos de un archivo.

```
unsigned        att;
```

Es otra de las variables pertenecientes al TRS Package, utilizada por las funciones install y uninstall para determinar si el programa está cargado en memoria.

```
char            tsr_fprint[20] = "tsrcom.v1";
```

Cuando se produzca un error, mje_error contendrá el mensaje de error que se enviará a la otra estación.

```
char            mje_error[MAXPATH+50];
```

En data se almacenará los datos que contiene cada ECB. Se declara un array del tamaño SPXDATOS por cada ECB que puede contener la ventana.

```
char            data[VENT][SPXDATOS];
```

En nom_arch se tendrá el nombre del archivo a copiar o enviar.

```
char            nom_arch[MAXPATH];
```

La variable cant_rcv llevará la cantidad de ECBs recibidos que aún no han sido analizados.

```
short           cant_rcv = 0;
```

Las variables lRcvECB e iRcvECB son índices al array RcvECB. La primera indica el siguiente lugar de memoria a donde se guardará el ECB que será recibido. La segunda indica el lugar de memoria donde se aloja el siguiente ECB a procesar.

```
short               lRcvECB  = 0,
                    iRcvECB  = 0;
```

La variable copia indica si se está procesando un comando TXPCOPIA.

```
short               copia    = 0;
```

La variable elem es el índice de la estructura que contendrá la información del directorio en caso de tratarse de un comando PDIR.

```
short           elem;
```

Socket es el número del socket local.

```
WORD            Socket = CHAT_SOCKET;                   /* número de socket local */
```

ConnectionID contendrá el número de identificación de la conexión local.

```
WORD            ConnectionID;
```

Las variables maxconex y dispconex indicarán respectivamente, las máximas conexiones soportadas por SPX, y la cantidad de conexiones disponibles para la aplicación.

```
WORD            maxconex, dispconex;
```

El protocolo SPX devolverá en ellas la versión y el release instalado.

```
BYTE            mayor, menor,
```

La variable fulldir contendrá las direcciones de red, nodo y número de socket.

```
                fulldir[LONGNET+LONGNODE+LONGSOCK],
```

La variable dato almacenará el dato que se envíe a la otra estación.

```
                dato[SPXDATOS];
```

La variable comando apuntará al comando que envía la otra estación.

```
COMANDO         *comando;
```

La variable PoolSPX apuntará a la cabecera de un paquete SPX.

```
SPXHeader       *PoolSPX,
```

ConnectionSPX se utilizará como cabecera de un paquete para el establecimiento de la conexión.

```
            ConnectionSPX;
```

HeadSPX almacenará las cabeceras de los paquetes de SPX que contiene cada ECB de la ventana.

```
SPXHeader       HeadSPX[VENT]; /* array de SPXs Headers */
```

ConnectionECB es el ECB que se utilizará para el establecimiento de la conexión.

```
ECB             ConnectionECB,
```

Se utilizará PoolECB para almacenar en memoria los paquetes que arriban.

```
            PoolECB[VENT], /* array de ECBs */
```

RcvECB es un array de punteros a los ECBs que se encuentran en la ventana.

```
            *RcvECB[VENT];          /* array de punteros a ECBs */
int         longitud,               /* longitud del dato enviado */
            dest,                   /* file handler */
            problema,               /*indica si ocurrió un problema */
            numwr;                  /* cantidad de caracteres escritos */
```

La variable ctrl_disp es utilizada para detectar los errores de dispositivo.

```
volatile int    ctrl_disp = 0;
```

Programa Principal

Se encarga de instalar o remover la aplicación residente, según corresponda. Para ello utiliza las rutinas del TSR Package de Zortech.

Previo a la instalación deja ECBs escuchando para el posterior establecimiento de una conexión.

Sólo en caso de remoción tiene argumentos de entrada.

Entradas : - **argc**, número de argumentos con los que se invoca el comando en el promt.

- **argv[]**, vector de punteros a cadenas de caracteres que contienen los argumentos con los que fue invocado.

Salidas : - ERROR u OKAY según corresponda.

```
int main(int argc, char **argv)
{
    int     i;
    short   j;
```

Inicializa el Display Package. El tipo de display y la dirección de memoria de la pantalla son determinados en esta llamada.

```
    disp_open();
```

Para remover el residente de memoria basta con ingresar en el promt:

- TSRCOM /R ó
- TSRCOM /r

```
    if((strncmp(argv[1],"/R",2) == 0) || strncmp(argv[1],"/r",2)==0))
    {
```

Para remover el programa de memoria utiliza la función tsr_uninstall(), que retorna un valor según el resultado de la operación:

i = 0 El programa ha sido removido con éxito.

i = 2 No se puede remover porque el programa no ha sido previamente cargado en memoria.

i = 3 No se puede remover, otro programa está cargado sobre él en memoria.

```
        /* trata de remover el programa de memoria */
        i = tsr_uninstall();

        if(i==0)
        {
            /* remueve con éxito */
            IPXCloseSocket(Socket);
            disp_printf("\n%s\n", "Programa removido");
```

```
                disp_flush();
                return OKAY;
        }
        /* ocurrió un error, no puede remover */
        if(i==2)
        {
                disp_printf("\n%s\n",
                        "No se puede remover, el Programa no ha sido cargado!");
                disp_flush();
        }
        if(i==3)
        {
                disp_printf("\n%s\n",
                        "No se puede remover, otro programa cargado sobre él!");
                disp_flush();
        }
        return ERROR;
}
```

Si las rutinas de IPX o SPX no están cargadas no se instala el residente.

```
/* Verifica que IPX esté instalado */
if (IPXInitialize() == 0x0)
{
        disp_printf("\nIPX no fue instalado.\n");
        disp_flush();
        disp_printf("\nTSRCOM no quedará residente en memoria\n");
        disp_flush();
        return ERROR;
}
/* Verifica que SPX esté instalado */
if (SPXInitialize(&mayor, &menor, &maxconex, &dispconex) == 0x0)
{
        disp_printf("\nSPX no fue instalado.\n");
        disp_flush();
        disp_printf("\nTSRCOM no quedará residente en memoria\n");
        disp_flush();
        return ERROR;
}
```

Si al abrir el socket se produce un error, esto puede ser porque:

- Se produjo realmente un error al abrirlo ó

- El socket ya fue abierto porque anteriormente el programa fue cargado en memoria.

```
/* Abre el socket */
if (IPXOpenSocket((BYTE *) &Socket, PERMANENT_SOCKET) != OKAY)
{
      disp_printf("\nOcurrió un error al abrir el socket, o");
      disp_flush();
      disp_printf("\nel programa ya fue cargado en memoria");
      disp_flush();
      disp_printf("\ny el socket abierto.\n");
      disp_flush();
      return ERROR;
}
```

Coloca los ECBs de la ventana a escuchar.

```
/*
** Inicializa ECBs para escuchar
** Utiliza arrays de SPXHeader y ECBs para no usar
** malloc y calloc ya que será residente
*/
for (j =0 ; j < VENT; j++)
{
```

El campo ESRAddress del ECB se inicializa con la dirección de la rutina del servicio de interrupción de evento pasada como parámetro ESR. Esta rutina será invocada cada vez que llega un paquete.

```
      IPXGetProcAddress(ReceiveESRHandler, PoolECB[j].ESRAddress);
```

El campo ECBSocket se setea con el Socket de la aplicación.

```
      PoolECB[j].ECBSocket = Socket;
```

El paquete "escuchar paquete" debe tener inicializado el campo FragmentCount en 2, pues contendrá la información de la cabecera y los datos de los paquetes que llegan.

```
      PoolECB[j].FragmentCount = 2;
```

La función IPXGetDataAddress() asigna la dirección de la cabecera del paquete "escuchar paquete" al campo descriptor del fragmento de la cabecera (correspondiente al ECB).

```
        IPXGetDataAddress((WORD *) &HeadSPX[j],
                          (WORD *) &PoolECB[j].FragmentDescriptor[0].Address);
```

Asigna la longitud de la cabecera del paquete al campo longitud de descriptor del fragmento correspondiente.

```
        PoolECB[j].FragmentDescriptor[0].Size = sizeof(SPXHeader);
```

Se asigna la dirección de los datos del paquete "escuchar paquete" y su longitud máxima (SPXDATOS) al campo descriptor del fragmento de datos (correspondiente al ECB).

```
        IPXGetDataAddress((WORD *) data[j],
                          (WORD *) &PoolECB[j].FragmentDescriptor[1].Address);
        PoolECB[j].FragmentDescriptor[1].Size = SPXDATOS;
```

El tipo de paquete (PacketType) indica el tipo de servicio ofrecido o requerido por el paquete. El tipo 5 significa que el paquete es del protocolo SPX.

```
        HeadSPX[j].PacketType = '5';
```

Se invoca a la API SPXListenForSequencedPacket().

```
        SPXListenForSequencedPacket(&PoolECB[j]);
    }
```

Coloca un paquete "escuchar conexión", a la espera de que alguna estación quiera establecer una conexión con él.

```
    /* Escucha por una conexion */
    EscucharConexion(Socket);
```

Despliega la presentación del residente.

```
    disp_close();
    /* Muestra presentación */
    Inicio();
```

Instala el residente en forma background utilizando la rutina tsr_install(), esto se logra pasando como parámetro la constante "TIMESLICE" definida en tsr.h. La aplicación se activará 18 veces por segundo.

```
    /* Instala el programa residente */
    i = tsr_install(TIMESLICE|TSR_DEBUG);
```

Si retorna del llamado a esta rutina quiere decir que la instalación no tuvo éxito.

```
    /* Si retorna aquí, ha ocurrido un error */
    disp_open();
    if(i != 0)
    {
          disp_printf("\n%s %i\n", "Falla al instalar, error",i);
          disp_flush();
    }
    disp_close();
    return ERROR;
}
```

Función popmain

Es la función que queda residente en memoria.

Si se estableció una conexión, verifica el tipo de comando recibido. De acuerdo a esto lo procesa, enviando a la estación remota la información requerida, o recibiendo la enviada.

Entradas : - **copia**, variable global, indica si se está ejecutando un comando TXPCOPIA.

- **cant_rcv**, variable global, cantidad de ECBs recibidos que aún no han sido analizados.

- **RcvECB**, variable global, vector de punteros a los paquetes que llegan.

- **iRcvECB**, variable global, indica el paquete que se analiza.

Salidas : - **copia**, indica si se está ejecutando un comando TXPCOPIA.

- **cant_rcv**, cantidad de ECBs recibidos que aún no han sido analizados.

- **iRcvECB**, indica el próximo paquete que se analizará.

```
void popmain(void)
{
    /*
    ** POPMAIN es una "palabra reservada" especial, función,
    **  a la cual las rutinas TSRs pasan el control.
    */
```

Si no está copiando un archivo enviado por la otra estación, entonces está escuchando un "establecer conexión".

```
    if (!copia)
    {
```

Cada vez que se activa verifica si se ha establecido alguna conexión. Si no es así sigue escuchando.

```
        /* No está procesando un comando TXPCOPIA */
        if(ConnectionECB.InUseFlag)
            return;   /* No se estableció conexión aún */
```

Verifica el estado en que llegó el paquete analizando el campo CompletionCode.

```
        switch (ConnectionECB.CompletionCode)
        {
```

Si llegó un paquete de conexión sin errores, la misma queda establecida. El número de identificación de conexión que se usará más adelante es guardado en la variable ConnectionID.

```
        case 0x00:
            /* Se estableció conexión */
            memcpy((BYTE *) &ConnectionID, (BYTE *) ConnectionECB.IPXWorkspace, 2);
            break;
```

Si el paquete tiene algún problema lo desecha y continúa esperando que llegue un paquete "escuchar conexión".

```
        default:
            /* El paquete de conexión llegó mal */
            EscucharConexion(Socket);
            return;
        }
    }
```

A esta altura está en condiciones de recibir paquetes de datos, es decir, se estableció la conexión.

Que cant_rcv sea mayor que 0 significa que hay paquetes para analizar.

```
/* Recibe y analiza paquetes */
if (cant_rcv > 0)
{
      /* hay por lo menos un paquete */
      problema = OKAY;
```

Analiza el paquete que arribó. En el campo CompletionCode tiene el estado final del ECB recibido.

```
      switch (RcvECB[iRcvECB]->CompletionCode)
      {

      case 0x00:
            /* El paquete llegó bien */
            PoolSPX = (SPXHeader *) RcvECB[iRcvECB]-> FragmentDescriptor[0].Address[0];
```

El paquete arribó con éxito, ahora debe verificar el tipo de paquete para ejecutar el proceso adecuado en cada caso.

```
            /* Verifica tipo de paquete */
            switch (PoolSPX->DataStreamType)
            {
```

Este tipo de paquete es generado por SPX cuando la estación remota llama a la API SPXTerminateConnection, sólo es enviado cuando el archivo se está copiando en esta estación.

```
            case 0xFE:
                  /*  Es un paquete para terminar la conexión  */
                  if (copia)
```

Se cierra el archivo que se está copiando y se inicializa la variable copia nuevamente en 0.

```
                  {
                        close(dest);
                        copia = 0;
                  }
                  break;
```

El tipo de paquete 0x1 es generado por la estación remota, indica que su contenido es un comando.

```
case 0x1:
```

Si se está recibiendo un archivo de la estación remota (copia = 1), no se puede recibir otro paquete de tipo comando, pues ya está ejecutando uno. Por lo tanto, no lo analiza.

```
    /* Es un comando */
if (copia)
        break;
```

Copia el comando enviado en el paquete en la variable nom_arch.

```
comando = (COMANDO *) RcvECB[iRcvECB]->
        FragmentDescriptor[1].Address[0];
strcpy(nom_arch, comando->origen);
```

El comando es un PDIR.

```
if (!strcmp(comando->comando, "PDIR"))
```

La función EjecutarDir() envía a la estación remota la información del directorio o archivos solicitados, cuyo path se encuentra en nom_arch. Si ocurre algún error lo indica en la variable problema.

```
problema = EjecutarDir(nom_arch,
                       PoolSPX->Source.Socket,
                       PoolSPX->Source.Network,
                       PoolSPX->Source.Node);
```

El comando es un RXPCOPIA, transmite un archivo para que la estación remota lo copie.

```
else if (!strcmp(comando->comando, "RXPCOPIA"))
```

La función EjecutarCopia() envía a la estación remota el archivo solicitado en tantos paquetes como sean necesarios. Si ocurre algún error lo informa en la variable problema.

```
problema = EjecutarCopia(nom_arch,
                         PoolSPX->Source.Socket,
                         PoolSPX->Source.Network,
                         PoolSPX->Source.Node);
```

El comando es un TXPCOPIA, copia el archivo transmitido por la estación remota.

```
else if (!strcmp(comando->comando, "TXPCOPIA"))
{
```

En este caso se recibe un archivo enviado por la estación remota. Cada vez que se activa recibe un paquete de datos que grabará.

Activa la función que captura los errores de dispositivo, ya que no puede permitir que se muestren los mensajes del sistema por pantalla (recordar que es un programa residente).

```
/* Maneja los errores de
** dispositivo
*/
_cerror_handler = myhandler;
cerror_open();
```

Si se produce un error, ya sea de dispositivo o cualquier otro, lo almacena en la variable mje_error para transmitirlo a la estación remota.

```
memset(mje_error, '\0', sizeof(mje_error));
```

Verifica que el archivo no exista, si existe envía el mensaje de error a la estación remota.

```
if (file_exists(nom_arch) || ctrl_disp)
{
      ctrl_disp = 0;
      sprintf(mje_error, "\n%s %s\n",
            "Error de dispositivo o Achivo ya existe",
             nom_arch);
      (void) MandarMje(PoolSPX->Source.Socket,
                       PoolSPX->Source.Network,
                       PoolSPX->Source.Node,
                       mje_error,
                       strlen(mje_error),
```

```
                                        0x2);
                problema = ERROR;
        }
```

Si no puede abrir el archivo para escritura envía el mensaje de error a la estación remota.

```
        else if ((dest = open(nom_arch,O_WRONLY | O_CREAT,
                S_IREAD | S_IWRITE )) == ERROR || ctrl_disp)
        {
                ctrl_disp = 0;
                sprintf(mje_error, "\n%s %s\n",
                        "No puede crear el archivo destino", nom_arch);
                (void) MandarMje(PoolSPX->Source.Socket,
                                PoolSPX->Source.Network,
                                PoolSPX->Source.Node,
                                mje_error,
                                strlen(mje_error),
                                0x2);
                problema = ERROR;
        }
```

Abrió el archivo para escritura, entonces envía a la estación remota un paquete de aviso para que comience a transmitir la información.

```
        else
        {
                /*
                ** Abrió el archivo sin
                ** problemas.
                ** Avisa a la otra estación que
                ** puede comenzar a enviar la
                ** información.
                */
                problema = MandarMje(PoolSPX->Source.Socket,
                                PoolSPX->Source.Network,
                                PoolSPX->Source.Node,
                                mje_error,
                                strlen(mje_error),
                                0x4);
```

Cierra el manejador de errores de dispositivo, ya que se desactivará cuando devuelva el recurso del procesador y quedará a la espera de un paquete.

```
                    if (!problema)
                    {
                        cerror_close();
                        /*
                        ** A partir de ahora comenzará
                        ** a copiar
                        */
```

Indica en la variable copia que se espera paquetes de datos que contengan fragmentos del archivo fuente.

```
                        copia = OK;
                    }
                }
            }
            break;
```

Este tipo de paquetes (0x4) sólo se recibe cuando se está ejecutando un comando PCOPIA en la estación remota, la copia es de un archivo almacenado en la estación remota a la estación en la está el residente (TXPCOPIA).

Graba el paquete que ha llegado, en el archivo previamente abierto. En caso de error lo reporta a la otra estación. Aquí vuelve a abrir el manejador de errores de dispositivo y lo cierra antes de desactivarse.

```
        case 0x4:
```

Activa la función que captura los errores de dispositivo.

```
            /*
            ** Es un paquete de datos.
            ** Escribe el paquete de datos
            ** en el disco
            */
            _cerror_handler = myhandler;
            cerror_open();
```

En la variable longitud se guarda la longitud del fragmento de datos del paquete, la misma está dada por el resto entre la longitud total del paquete (PoolSPX->PacketLength) y la longitud de la cabecera del paquete (RcvECB[iRcvECB]->FragmentDescriptor[0].Size).

```
            longitud = IntSwap(PoolSPX->PacketLength) -
                RcvECB[iRcvECB]->FragmentDescriptor[0].Size;
```

En la variable dato se coloca el mensaje de error *enviado* contenido en el fragmento descriptor de dato, teniendo en cuenta su longitud (longitud).

```
memcpy(dato,
      (BYTE *) (RcvECB[iRcvECB]->FragmentDescriptor[1].Address[0]),
      longitud);
```

Graba el fragmento del archivo enviado (dato) en el archivo destino (dest). Si ocurre un error, transmite el mensaje de error a la estación remota.

```
if ((numwr = write(dest, dato,longitud))!=longitud || ctrl_disp)
{
      ctrl_disp = 0;
      memset(mje_error, '\0', sizeof(mje_error));
      sprintf(mje_error, "\n %s \n",
            "Error al intentar escribir en el disco.");
      (void) MandarMje(PoolSPX->Source.Socket,
                  PoolSPX->Source.Network,
                  PoolSPX->Source.Node,
                  mje_error,
                  strlen(mje_error),
                  0x2);
      problema = ERROR;
}
```

Pudo grabar bien, cierra el manejador de errores de dispositivo, ya que se desactivará cuando devuelva el recurso del procesador y quedará a la espera de un paquete de datos.

```
else
      cerror_close();
break;
```

La estación remota envía este tipo de paquetes cuando detecta algún error durante el proceso de un comando *TXPCOPIA*.

```
case 0x2:
      /* Es un paquete de error */
      problema = ERROR;
      break;
```

No reconoce otro tipo de paquete que no sea alguno de los antes mencionados.

```
        default:
                break;
        }
```

Mientras la conexión esté activa, continuamente escucha por un paquete. Para ello cada vez que se termina de analizar un paquete lo pone nuevamente a escuchar otro (tiene en cuenta la ventana receptora).

```
        /* pone a escuchar el ECB recién analizado */
        SPXListenForSequencedPacket(RcvECB[iRcvECB]);
        if (iRcvECB == VENT - 1)
                iRcvECB = 0;
        else
                iRcvECB++;
        cant_rcv--;
        break;
```

El paquete llegó mal debido a una falla de conexión.

```
default:
        /* falla de conexión */
        problema = ERROR;
}
```

Si se produjo algún error durante el proceso, aborta la conexión y coloca los paquetes que han sido recibidos nuevamente a escuchar. Luego queda esperando por una nueva conexión y se desactiva.

```
if (problema)
{
```

Si se estaba ejecutando un comando TXPCOPIA borra el archivo en donde estaba copiando.

```
        if (copia)
        {
                /* Estaba copiando un archivo */
                Problem(nom_arch);
                cerror_close();
                copia = 0;
        }
        /* Aborta la conexión */
        SPXAbortConnection(ConnectionID);
```

```
                /* Pone paquetes a escuchar */
                while ( cant_rcv > 0 )
                {
                        SPXListenForSequencedPacket(RcvECB[iRcvECB]);
                        if (iRcvECB == VENT - 1)
                                iRcvECB = 0;
                        else
                                iRcvECB++;
                        cant_rcv--;
                }
                /* Inicializa variables a ECBs */
                lRcvECB = 0;
                iRcvECB = 0;
                /* Escucha por una nueva conexión */
                EscucharConexion(Socket);
        }
```

El paquete recibido se procesó sin que surgieran problemas. Por lo tanto, si no está ejecutando un comando TXPCOPIA, termina la conexión, queda escuchando por una nueva y se desactiva.

```
        else
        {
                /* No se han producido errores */
                if (!copia)
                {
                        /* No está copiando.
                        ** Termina la conexión
                        */
                        TerminarConexion(Socket, ConnectionID);
                        lRcvECB = 0;
                        iRcvECB = 0;
                        cant_rcv = 0;
                        /* Escucha por una nueva conexión */
                        EscucharConexion(Socket);
                }
        }
    }
}
```

Función Inicio

Se encarga de desplegar la presentación del programa residente.

Entradas : - Ninguna.

Salidas : - Ninguna.

No tiene entradas ni salidas.

```
void Inicio (void)
{
```

Para visualizar se utiliza el Display Package provisto por el compilador. Este permite hacer las I/O de caracteres por pantalla en forma más rápida.

```
    disp_open();
    disp_move(0,0);
    disp_eeop();
    disp_box(0, DISP_NORMAL, 3, 0, 11, 79);
    disp_move(5, 3);
    disp_printf("\tTRSCOM (Versión 1.1) Instalado.\n");
    disp_printf("\tDerechos Reservados 1995, EPSB Corp.\n\n");
    disp_printf("\tPara removerlo de memoria, desde la Línea de Comando escriba:\n");
    disp_printf("\tTsrcom /r ó Tsrcom /R.\n");
    disp_move(11, 0);
    disp_close();
}
```

La salida por pantalla es:

```
TRSCOM (Versión 1.1) Instalado.
Derechos Reservados 1995, EPSB Corp.

Para removerlo de memoria, desde la Línea de Comando escriba:
Tsrcom /r ó Tsrcom /R.
```

Función file_exists

Su función es verifcar la existencia de un archivo.

Entradas : - **filename**, nombre del archivo.

Salidas : - 1, si el archivo existe.
- 0, si el archivo no existe.

```
int file_exists (char *filename)
{
    return (access(filename, 0) == 0);
}
```

Función Problem

Cierra y borra un archivo.

Entradas : - **destino**, nombre del archivo.

- **dest**, File handle del archivo.

Salidas : - Ninguna

```
void Problem (char *destino)
{
    close(dest);
    remove(destino);
}
```

Función EscucharConexión

Coloca un paquete a escuchar la llegada de un paquete "establecer conexión".

Entradas : - **socket**, número de socket.

Salidas : - Ninguna.

```
void EscucharConexion (WORD Socket)
{
    /* Inicializa paquete de conexión */
    memset(&ConnectionECB.ESRAddress, 0, 4 );
    ConnectionECB.ECBSocket     = Socket;
    ConnectionECB.FragmentCount = 1;
    IPXGetDataAddress((WORD *) &ConnectionSPX,
                      (WORD *) &ConnectionECB.FragmentDescriptor[0].Address);
    ConnectionECB.FragmentDescriptor[0].Size = sizeof(SPXHeader);
    ConnectionSPX.PacketType    = '5';
```

El watchdog de SPX monitorea una conexión SPX, asegurando que la misma está funcionando apropiadamente cuando no está pasando tráfico a través de ella.

RetryCount (contador de reintentos) utilizado para establecer la cantidad de veces en que SPX reenviará paquetes de confirmación antes de concluir que el nodo destino no está funcionando correctamente.

```
    /*
    ** RetryCount= 0 SPX usa su propio intervalo de tiempo
    ** para reenviar paquetes
    ** watchdog  = 1 habilitado para determinar
    ** si la conexion está bien cuando no hay tráfico
    */
    SPXListenForConnection((BYTE) 0, (BYTE) 1, &ConnectionECB);
}
```

Función TerminarConexión

Termina una conexión preestablecida.

Entradas : - **socket**, número de socket.

- **ConnectionID**, número de identificación de la conexión.

Salidas : - **OKAY**.

```
int TerminarConexion (WORD Socket,
                      WORD ConnectionID)
{
    ECB       TerminateECB;
    SPXHeader TerminateSPX;

    /* Inicializa paquete para terminar conexión */

    memset( &TerminateECB.ESRAddress,0,4 );
    TerminateECB.ECBSocket     = Socket;
    TerminateECB.FragmentCount = 1;
    IPXGetDataAddress((WORD *) &TerminateSPX,
                      (WORD *) &TerminateECB.FragmentDescriptor[0].Address);
    TerminateECB.FragmentDescriptor[0].Size = sizeof(SPXHeader);
    SPXTerminateConnection(ConnectionID, &TerminateECB);

    /* Espera que se complete el Terminate */
```

```
    while ( TerminateECB.InUseFlag )
          IPXRelinquishControl();

    return OKAY;
}
```

Función EjecutarCopia

Lee y envía un archivo por la red hacia otra estación luego de verificar que puede acceder a él y que no está vacío.

En caso de producirse algún problema envía un mensaje de error a la estación remota.

Entradas : - **path**, path del archivo que se transmitirá por la red.

- **Socket_dest**, número de socket destino.

- **Network_dest**, dirección de red destino.

- **Nodo_dest**, dirección de nodo destino.

Salidas : - ERROR u OKAY según corresponda.

```
int EjecutarCopia (char path[MAXPATH],
                   BYTE Socket_dest[LONGSOCK],
                   BYTE Network_dest[LONGNET],
                   BYTE Nodo_dest[LONGNODE])
{
    BYTE    dato[SPXDATOS];
    int     arch, problem, numread;
    long    larch;     /* cantidad de bytes del archivo */
```

Coloca el manejador de errores del dispositivo, recordemos que la varible ctrl_disp toma el valor de verdadero si se produce un error de I/O en el dispositivo.

```
    /* Maneja los errores de dispositivo */
    _cerror_handler = myhandler;
    cerror_open();
    memset(mje_error, '\0', sizeof(mje_error));
```

Este comando actúa en forma semejante al "COPY" del DOS, por lo tanto no puede enviar los archivos ocultos o del sistema.

Para lograr su objetivo, obtiene los atributos del archivo y los verifica.

```
/* Obtiene los atributos del archivo */
dos_getfileattr(path, &att);
```

Si el archivo es oculto o del sistema no puede acceder a él, en este caso envía el mensaje de error a la estación remota.

```
/* Verifica si es un archivo oculto o del sistema */
if (ctrl_disp || (att & FA_HIDDEN) || (att & FA_SYSTEM))
{
      sprintf(mje_error, "\n%s %s\n", "No puede acceder a", path);
      (void) MandarMje(Socket_dest, Network_dest, Nodo_dest, mje_error,
                  strlen(mje_error), 0x2);
      return ERROR;
}
```

Abre el archivo para lectura solamente, si el archivo no existe o se produce un error de dispositivo (ctrl_disp) envía el error al extremo opuesto.

```
/*
** No es un archivo oculto o del sistema.
** Lo abre para lectura.
*/
if (ERROR == (arch=open(path,O_RDONLY)) || ctrl_disp)
{
      ctrl_disp = 0;
      sprintf(mje_error, "\n%s %s\n", "No se encontro o no se pudo abrir", path);
      (void) MandarMje(Socket_dest, Network_dest, Nodo_dest, mje_error,
                  strlen(mje_error), 0x2);
      return ERROR;
}
```

Analiza la longitud del archivo, si no tiene bytes el archivo está vacío, entonces envía un mensaje de error a la estación remota.

```
/* toma la longitud del archivo */
larch = filelength(arch);
if (larch == 0)
{
      sprintf(mje_error, "\n%s %s\n", "Archivo vacio", path);
      (void) MandarMje(Socket_dest, Network_dest, Nodo_dest, mje_error,
                  strlen(mje_error), 0x2);
      close(arch);
      return ERROR;
}
```

Lee del archivo la cantidad de información equivalente a la longitud que permite SPX para los datos (SPXDATOS).

Luego envía esa información a la otra estación. Esto lo hace hasta que se termine el archivo o se produzca algún error.

Mientras no se produzca ningún error y no se haya enviado el archivo completo toma una fracción del archivo que corresponderá al fragmento de datos del paquete a transmitir. Esta fracción tendrá la longitud máxima permitida por SPX, SPXDATOS.

```
/* Posiciona al principio del archivo */
lseek(arch, 0L, SEEK_SET);
problem = 0;
while (!problem && (larch > 0))
{
```

Limpia la variable dato que contendrá la fracción del archivo o el mensaje de error que se transmita.

```
      /* Deja un fin de cadena para reconocer la
      ** longitud
      */
      memset(dato, '\0', SPXDATOS);
      /*
      ** Lee la información del archivo y
      ** la envía a la otra estación
      */
      if((numread = read(arch, dato, SPXDATOS - 1)) == ERROR || ctrl_disp)
      {
```

Se produjo un error al leer el archivo, el mensaje es enviado a la estación remota y la variable problem toma el valor de verdadero.

```
                ctrl_disp = 0;
                sprintf(mje_error, "\n%s %s\n","Error leyendo el archivo", path);
                (void) MandarMje(Socket_dest, Network_dest, Nodo_dest, mje_error,
                                strlen(mje_error), 0x2);
                problem++;
        }
```

El archivo pudo leerse sin problemas. La variable numread contiene la cantidad de bytes que fueron leídos y larch la cantidad de bytes que aún faltan leer.

Si al enviar el paquete con los datos se produce algún problema la variable problem se pone en verdadero.

```
        else
        {
                larch = larch - numread;
                if (MandarMje(Socket_dest, Network_dest, Nodo_dest, dato, numread, 0x4)
                        == ERROR)
                        problem++;
        }
}
```

Cierra el archivo y el manejador de errores del dispositivo.

```
close(arch);
cerror_close();
```

Si se produjo algún problema retorna ERROR, sino retorna OKAY.

```
if (problem)
        return ERROR;
return OKAY;
}
```

Función MandarMje

Envía un paquete hacia otra estación de trabajo.

Entradas : - **Socket_dest**, número de socket destino.

- **Network_dest**, dirección de red destino.

- **Nodo_dest**, dirección de nodo destino.
- **dato**, información que se enviará en el paquete.
- **longitud**, longitud de la información a enviar.
- **tipo**, tipo de paquete.
- **Socket**, número de socket de la estación origen.
- **ConnectionID**, número de identificación de la conexión.

Salidas : - ERROR u OKAY según corresponda.

```
int MandarMje (BYTE *Socket_dest,
               BYTE *Network_dest,
               BYTE *Nodo_dest,
               void *dato,
               int  longitud,
               BYTE tipo)
{
     ECB            SendECB;
     SPXHeader      SendSPX;
     int            TransportTime;

     /* Inicializa paquete de datos */
     memset( &SendECB.ESRAddress,0,4 );
     SendECB.ECBSocket      = Socket;
     SendECB.FragmentCount  = 2;
     IPXGetDataAddress((WORD *) &SendSPX,
                       (WORD *) &SendECB.FragmentDescriptor[0].Address);
     SendECB.FragmentDescriptor[0].Size = sizeof(SPXHeader);
     IPXGetDataAddress((WORD *) dato,
                       (WORD *) &SendECB.FragmentDescriptor[1].Address);
     SendECB.FragmentDescriptor[1].Size = longitud; /*SPXDATOS;*/
     SendSPX.PacketType     = '5';
     SendSPX.DataStreamType = tipo;
     memcpy(SendSPX.Destination.Socket, Socket_dest, LONGSOCK);
     memcpy(SendSPX.Destination.Network, Network_dest, LONGNET);
     memcpy(SendSPX.Destination.Node, Nodo_dest, LONGNODE);
     memcpy(fulldir, SendSPX.Destination.Network, LONGNET+LONGNODE+LONGSOCK);
     IPXGetLocalTarget((BYTE *) &fulldir, (BYTE *)&SendECB.ImmediateAddress,
                       &TransportTime);

     /* Envía paquete a la otra estación */
       SPXSendSequencedPacket(ConnectionID, &SendECB);
```

Espera que se complete el Send.

```
while(SendECB.InUseFlag)
      IPXRelinquishControl();
```

Verifica que el paquete se halla enviado con éxito y retorna OKAY. En caso contrario retorna ERROR.

```
switch(SendECB.CompletionCode)
{
case 0x00:
      /* El paquete fue enviado con éxito */
      return OKAY;
default:
      /* Falló al enviar el paquete */
      return ERROR;
}
}
```

Función EjecutarDir

Obtiene la información perteneciente a un directorio o archivo, arma paquetes con ella y los envía a otra estación de trabajo. Previamente verifica si éste existe.

Si se produce algún problema durante el proceso envía a la otra estación el correspondiente mensaje de error.

Entradas : - **path**, información requerida.

- **Socket_dest**, número de socket destino.
- **Network_dest**, dirección de red destino.
- **Nodo_dest**, dirección de nodo destino.

Salidas : - ERROR u OKAY según corresponda.

```
int EjecutarDir (char path[MAXPATH],
                 BYTE Socket_dest[LONGSOCK],
                 BYTE Network_dest[LONGNET],
                 BYTE Nodo_dest[LONGNODE])
{
```

```
int     pma, long_info;
struct  FIND *ffblk;
struct  FIND info[MAXINFO];
```

Coloca el manejador de errores de dispositivo propio (myhandler).

```
memset(mje_error, '\0', sizeof(mje_error));
/* Maneja los errores de dispositivo */
_cerror_handler = myhandler;
cerror_open();
```

Las funciones findfirst y findnext son usadas para buscar archivos que macheen, en este caso, con path. Estas funciones devuelven un puntero a una estructura estática FIND definida en dos.h, esta estructura contiene la información del archivo o directorio que es encontrado. Si el archivo o directorio no es encontrado devuelven NULL.

La llamada al sistema findnext obtiene el próximo nombre de archivo que machee con el nombre de archivo especificado en la llamada a findfirst. Sólo se puede llamar a findnext después de findfirst.

```
ffblk = findfirst(path, FA_DIREC);
elem = 0;
if (ctrl_disp)
{
```

Se produjo un error de dispositivo, envía el paquete de error a la estación remota.

```
/* Se produjo un error de dispositivo */
      ctrl_disp = 0;
      sprintf(mje_error, "\n%s %s\n","Error al intentar leer del disco", path);
      (void) MandarMje(Socket_dest, Network_dest, Nodo_dest, mje_error,
                  strlen(mje_error), 0x2);
      pma = ERROR;
}
```

Si ffblk es 0 no se encontró ningún archivo que machee con path, por lo tanto envía el paquete de error al extremo opuesto. La variable pma toma el valor de error.

```
else if (ffblk == 0)
{
      /* No encontró el directorio */
      sprintf(mje_error, "\n%s %s\n", "No se encontro", path);
```

```
        (void) MandarMje(Socket_dest, Network_dest, Nodo_dest, mje_error,
                        strlen(mje_error), 0x2);
        pma = ERROR;
    }
    else
        pma = OKAY;
```

Obtiene la información del directorio o archivo y con ella arma paquetes de datos que luego envía a la estación que lo requirió.

```
    while (ffblk && !pma)
    {
```

La cantidad máxima de estructuras FIND que se pueden enviar en un paquete es MAXINFO. La variable elem es el índice al array de estructuras FIND que será transmitido a la estación remota. Cuando elem tiene el valor de MAXINFO se completó la información que puede contener un paquete, entonces un paquete de datos es enviado por medio de la función MandarMje(). La variable elem vuelve a tomar el valor 0.

```
        /* Colecciona la información del directorio */
        if (elem == MAXINFO)
        {
            /* Calcula longitud de la información
            ** a enviar
            */
            long_info = elem * sizeof(info[0]);
            pma = MandarMje(Socket_dest, Network_dest, Nodo_dest, info, long_info, 0x4);
            elem = 0;
        }
```

El paquete de datos se pudo enviar sin inconvenientes.

```
        if (!pma)
        {
```

Copia la información apuntada por ffblk en el array que será transmitido info por medio de la función CopiarInfo().

```
            CopiarInfo(info, ffblk);
            /* busca el próximo archivo del directorio */
```

Busca el próximo archivo que machee con path. Si se produce algún error transmite un paquete de error y pone la variable pma con ERROR.

```
                ffblk = findnext();
                if (ctrl_disp)
                {
                        ctrl_disp = 0;
                        sprintf(mje_error, "\n%s %s\n", "Error al intentar leer del disco",
                                path);
                        (void) MandarMje(Socket_dest, Network_dest, Nodo_dest, mje_error,
                                        strlen(mje_error), 0x2);
                        pma = ERROR;
                }
        }
    }
```

Si no hubo problemas envía el último paquete de datos.

```
    if (!pma)
    {
        /* Calcula longitud de la información a enviar */
        long_info = elem * sizeof(info[0]);
        /* Envía el último paquete */
        pma = MandarMje(Socket_dest, Network_dest, Nodo_dest, info, long_info, 0x4);
    }
```

Cierra el manejador de errores de dispositivo y devuelve el valor de pma.

```
    cerror_close();
    return pma;
}
```

Función CopiarInfo

Copia la información obtenida de un directorio o archivo (ffblk) en el array de estructuras FIND (info).

Entradas : - **info**, estructura en la que se almacena la información obtenida.

- **ffblk**, información que se copiará en la estructura.
- **elem**, variable global, índice al elemento del array info donde se copiará ffblk.

Salidas : - **elem**, índice al próximo elemento a copiar en info.

- **info**, estructura en la que se almacena la información obtenida.

```
void  CopiarInfo (struct FIND info[],
                  struct FIND *ffblk)
{
     strcpy(info[elem].reserved, ffblk->reserved);
     info[elem].attribute = ffblk->attribute;
     info[elem].time = ffblk->time;
     info[elem].date = ffblk->date;
     info[elem].size = ffblk->size;
     strcpy(info[elem].name, ffblk->name);
     elem++;
}
```

Función myhandler

Intercepta un error de dispositivo de I/O, dándole el control al programa.

Entradas : - **di**, código de error.

Salidas : - **ax**, determina la acción a tomar.

```
int _far _cdecl myhandler (int *ax, int *di)
{
     ctrl_disp++;
     /*
     ** Ignora el error.
     ** Da el control al programa.
     */
     *ax = 0x00;
     return 1;
}
```

6.4.10 Desarrollo del Módulo IPXSPX (ipxspx.h - ipxspx.c)

Contiene las definiciones de los paquetes IPX y SPX, así como también del Event Control Block (ECB) necesario para la transmisión de los mismos, y toda otra definición relacionada con ellos.

También contiene las llamadas a las APIs de los servicios de comunicaciones.

La descripción de cada campo de las estructuras, así como también como la interface de cada una de las APIs está explicado exhaustivamente en las secciones 3.7 Estructura del Paquete IPX , 3.8 Estructura del Paquete SPX y 3.12 APIs del Servicio de Comunicación. Por lo tanto aquí se dará un pantallazo general de su implementación.

Include

Definiciones de las estructuras usadas por los protocolos IPX y SPX a través de las APIs

```
/*************************************************************/
/*   IPX/SPX - Header para Funciones IPX/SPX                 */
/*************************************************************/

/* Tipos de Datos */
typedef unsigned char BYTE;
typedef unsigned int  WORD;
typedef unsigned long LONG;

/* Modo de abrir el socket */
#define TEMPORARY_SOCKET (BYTE)0
#define PERMANENT_SOCKET (BYTE)0xFF

/* Numero de Socket */
#define INVITE_SOCKET    0x0150     /* Socket del Cliente */
#define CHAT_SOCKET      0x0250     /* Socket del Servidor */

/* Estado de retorno */
#define OKAY             0
#define ERROR           -1

/* Constantes para definicion de estructuras para comunicacion */
#define SPXDATOS         534  /* longitud de los datos */
#define LONGNET          4    /* longitud de dir. de red */
#define LONGNODE         6    /* longitud de dir. de nodo */
#define LONGSOCK         2    /* longitud del nro. de socket */

/* Archivo de Direccion de Nodos */
#define NODOMAPA "mapa.txt"
```

```
typedef struct
{
     BYTE    Network[LONGNET];        /* high-low */
     BYTE    Node[LONGNODE];          /* high-low */
     BYTE    Socket[LONGSOCK];        /* high-low */
} IPXAddress;

/* Estructura del paquete IPX */
typedef struct IPXPacketStructure
{
     WORD                  PacketCheckSum;  /* high-low */
     WORD                  PacketLength;    /* high-low */
     BYTE                  PacketTransportControl;
     BYTE                  PacketType;
     IPXAddress            Destination;
     IPXAddress            Source;
} IPXPacket;

struct  ECBFragment
{
     WORD          Address[2];        /* offset-segment */
     WORD          Size;              /* low-high */
};

/* Estructura del ECB */
typedef struct ECBStructure
{
     WORD                  Link[2];       /* offset-segment */
     WORD                  ESRAddress[2]; /* offset-segment */
     BYTE                  InUseFlag;
     BYTE                  CompletionCode;
     WORD                  ECBSocket;          /* high-low */
     BYTE                  IPXWorkspace[4];    /* N/A */
     BYTE                  DriverWorkspace[12]; /* N/A */
     BYTE                  ImmediateAddress[6]; /* high-low */
     WORD                  FragmentCount;      /* low-high */
     struct ECBFragment    FragmentDescriptor[2];
} ECB;

/* Estructura del paquete SPX */
typedef struct SPXPacketStructure
{
     WORD                  PacketCheckSum;          /* high-low */
     WORD                  PacketLength;            /* high-low */
     BYTE                  PacketTransportControl;
     BYTE                  PacketType;
```

```
    IPXAddress          Destination;
    IPXAddress          Source;
    BYTE                ConnectionControl;
    BYTE                DataStreamType;
    WORD                SourceConnectionID;      /* high-low */
    WORD                DestinationConnectionID; /* high-low */
    WORD                SequenceNumber;          /* high-low */
    WORD                AcknowledgeNumber;       /* high-low */
    WORD                AllocationNumber;        /* high-low */
} SPXHeader;

/* Estructura del pedido que contiene el comando */
typedef struct SPXpedido {
    char    comando[10];
    char    origen[250];
} COMANDO;

extern int  IntSwap (int num);
```

Prototipos de las llamadas a los servicios brindados por IPX.

```
extern BYTE IPXInitialize (void);
extern int  IPXOpenSocket (BYTE *socketNumber,
                           BYTE socketType);
extern void IPXCloseSocket (WORD socketNumber);
extern int  IPXGetLocalTarget (BYTE *networkAddress,
                               BYTE *immediateAddress,
                               int *transportTime);
extern void IPXSendPacket (ECB *eventControlBlock);
extern void IPXListenForPacket (ECB *eventControlBlock);
extern void IPXScheduleIPXEvent (WORD timeUnits,
                                 ECB *eventControlBlock);
extern int  IPXCancelEvent (ECB *eventControlBlock);
extern int  IPXGetIntervalMarker (void);
extern void IPXGetInternetworkAddress (BYTE *networkAddress);
extern void IPXRelinquishControl (void);
extern void IPXDisconnectFromTarget (BYTE *networkAddress);
extern void IPXGetDataAddress (WORD *data,
                               WORD *addressField);
```

Prototipos de las llamadas a los servicios brindados por SPX.

```
extern int  SPXEstablishConnection (BYTE RetryCount,
                                    BYTE Watchdog,
                                    WORD *connectionID,
                                    ECB *eventControlBloc );
extern void SPXListenForConnection (BYTE retryCount,
                                    BYTE watchDog,
                                    ECB *eventControlBlock);
extern void SPXListenForSequencedPacket (ECB *eventControlBloc);
extern void SPXSendSequencedPacket (WORD ConnectionID,
                                    ECB *eventControlBloc);
extern void SPXTerminateConnection (WORD ConnectionID,
                                    ECB *eventControlBloc);
extern void SPXAbortConnection (WORD connectionID);
```

Include propios del archivo ipxspx.c

```
#include <stdio.h>
#include <dos.h>
#include "ipxspx.h"
```

Función IntSwap

Invierte un número entero.

Entradas : - **int**, número que se desea invertir.

Salidas : - **temp**, el entero invertido.

```
extern int IntSwap (int num)
{
    int temp;

    temp = ( num & 0xFF00 ) >> 8;
    temp += ( num & 0xFF ) << 8;

    return temp;
}
```

int86 e int86x

Todas las llamadas a los servicios de comunicaciones tanto de IPX como de SPX se realizan a través de las funciones **int86** e **int86x**. Por lo tanto se cree oportuno dar una breve explicación de como se manejan.

int86()

No se puede usar int86 para las interrupciones 0x25 o 0x26.

Descripción : Ejecuta una interrupción de software.

Uso : #include <dos.h>
int int86 (int intnum, union REGS *regsin, union REGS *regsout);

Entrada : **intnum**, es un número de interrupción (0..255)

regsin, es un puntero a la estructura que contiene los valores de los registros AX, BX, CX, DX, SI y DI que se pasan a la interrupción.

Salida : **regsout**, es un puntero a una estructura dentro de la cual retorna los valores de los registros que pueden ser escritos.
En el registro AX retorna el valor de terminación de la interrupción.
El estado del carry flag puede ser determinado desde x.cflag en regsout

La estructura REGS está definida en dos.h.

int86x()

No se puede usar int86x para las interrupciones 0x25 o 0x26.

Descripción : Ejecuta una interrupción de software.

Uso : #include <dos.h>
int int86 (int intnum, union REGS *regsin, union REGS *regsout, struct SREGS *segregs);

Entrada : **intnum**, es un número de interrupción (0..255)

regsin, es un puntero a la estructura que contiene los valores de los registros AX, BX, CX, DX, SI y DI que se pasan a la interrupción.

Salida : regsout, es un puntero a una estructura dentro de la cual retorna los valores de los registros que pueden ser escritos.
En el registro AX retorna el valor de terminación de la interrupción.
El estado del carry flag puede ser determinado desde x.cflag en regsout

segregs, es un puntero a la estructura que contiene los valores de los registros de segmentos que se pasan a la interrupción. También es usada para retornar los valores en los registros de segmentos después de que la interrupción termina.

Las estructuras REGS y SREGS están definidas en dos.h.

Función IPXInitialize

Retorna la dirección de las rutinas del servicio de interrupción de IPX.

Entradas : - Ninguna.

Salidas : - **outregs.h.al**, toma el valor 00h si IPX está instalado.

```
extern BYTE IPXInitialize (void)
{
    union REGS inregs,outregs;
    inregs.x.ax = 0x7A;
    int86(0x2F,&inregs,&outregs);
    return(outregs.h.al);
}
```

Función IPXOpenSocket

Abre un socket IPX.

Entradas : - **socketNumber**, número de socket a abrir.
socketTupe, tipo de socket.

Salidas : - **outregs.h.al**, puede tomar los siguientes valores:
00h si abrió el socket,
FFh si el socket ya está abierto,
FEh si la tabla de sockets está llena.

```
int IPXOpenSocket (BYTE socketNumber[2],
                   BYTE socketType)
{
     union REGS inregs,outregs;

     inregs.x.bx = 0;
     inregs.h.dl = socketNumber[0];
     inregs.h.dh = socketNumber[1];
     inregs.h.al = socketType;
     int86(0x7A,&inregs,&outregs);
     return(outregs.h.al);
}
```

Función IPXCloseSocket

Cierra un socket IPX.

Entradas : - **socketNumber**, número de socket a cerrar.

Salidas : - Ninguna.

```
void IPXCloseSocket (WORD socketNumber)
{
     union REGS inregs,outregs;

     inregs.x.bx = 1;
     inregs.x.dx = (int) socketNumber;
     int86(0x7A,&inregs,&outregs);
}
```

Función IPXGetLocalTarget

Retorna la dirección del router más cercano y el tiempo estimado para entregar un paquete al destino.

Entradas : - **networkAddress**, dirección de red + dirección de nodo + número número de socket destino.

Salidas : - **inmediateAddress**, dirección del puente local más cercano.

- **transportTime**, tiempo estimado para entregar un paquete al destino.
- **outregs.h.al**, puede tomar los siguientes valores:

00h exitoso,

FAh no se encontró camino al nodo destino.

```
int IPXGetLocalTarget (BYTE *networkAddress,
                       BYTE *immediateAddress,
                       int  *transportTime)
{
     union REGS inregs,outregs;
     struct SREGS segregs;

     segread(&segregs);
     segregs.es = segregs.ds;

     inregs.x.bx = 2;
     inregs.x.si = (int) networkAddress;
     inregs.x.di = (int) immediateAddress;
     int86x(0x7A,&inregs,&outregs,&segregs);
     *transportTime = outregs.x.cx;
     return(outregs.h.al);
}
```

Función IPXSendPacket

Inicia la transmisión de un paquete IPX.

Entradas : - **eventControlBlock**, apunta al Event Control Block que contiene el paquete.

Salidas : - Ninguna.

```
void IPXSendPacket (ECB *eventControlBlock)
{
     union REGS inregs,outregs;
     struct SREGS segregs;

     segread(&segregs);
     segregs.es = segregs.ds;

     inregs.x.bx = 3;
     inregs.x.si = (int) eventControlBlock;
     int86x(0x7A,&inregs,&outregs,&segregs);
}
```

Función IPXListenForPacket

Prepara a IPX para recibir un paquete.

Entradas : - **eventControlBlock**, apunta al Event Control Block que contendrá el paquete que llegue.

Salidas : - Ninguna.

```
void IPXListenForPacket (ECB *eventControlBlock)
{
     union REGS inregs,outregs;
     struct SREGS segregs;

     segread(&segregs);
     segregs.es = segregs.ds;

     inregs.x.bx = 4;
     inregs.x.si = (int) eventControlBlock;
     int86x(0x7A,&inregs,&outregs,&segregs);
}
```

Función IPXScheduleIPXEvent

Administra un evento de IPX.

Entradas : - **timeUnits**, número de ticks que se difiere un ECB.

- **eventControlBlock**, apunta al Event Control Block que se difiere.

Salidas : - Ninguna.

```
void IPXScheduleIPXEvent (WORD timeUnits,
                          ECB *eventControlBlock)
{
     union REGS inregs,outregs;
     struct SREGS segregs;

     segread(&segregs);
     segregs.es = segregs.ds;

     inregs.x.bx = 5;
     inregs.x.si = (int) eventControlBlock;
     inregs.x.ax = (int) timeUnits;
     int86x(0x7A,&inregs,&outregs,&segregs);
}
```

Función IPXCancelEvent

Cancela un evento definido por un ECB.

Entradas : - **eventControlBlock**, apunta a un Event Control Block.

Salidas : - **outregs.h.al**, puede tomar los siguientes valores:

00h exitoso,

F9h ECB no puede ser cancelado,

FFh ECB no está en uso.

```
int IPXCancelEvent (ECB *eventControlBlock)
{
     union REGS inregs,outregs;
     struct SREGS segregs;

     segread(&segregs);
     segregs.es = segregs.ds;

     inregs.x.bx = 6;
     inregs.x.si = (int) eventControlBlock;
     int86x(0x7A,&inregs,&outregs,&segregs);
     return(outregs.h.al);
}
```

Función IPXGetIntervalMarker

Marca el tiempo de IPX.

Entradas : - Ninguna.

Salidas : - **outregs.h.ax**, indica el intervalo entre eventos.

```
int IPXGetIntervalMarker ()
{
     union REGS inregs,outregs;

     inregs.x.bx = 8;
     int86(0x7A,&inregs,&outregs);
     return(outregs.x.ax);
}
```

Función IPXGetInternetworkAddress

Retorna la dirección de red y de nodo de la estación requerida.

Entradas - Ninguna.
:

Salidas - **networkAddress**, dirección de red y nodo.
:

```
void IPXGetInternetworkAddress (BYTE networkAddress[10])
{
     union REGS inregs,outregs;
     struct SREGS segregs;

     segread(&segregs);
     segregs.es = segregs.ds;

     inregs.x.bx = 9;
     inregs.x.si = (int) networkAddress;
     int86x(0x7A,&inregs,&outregs,&segregs);
}
```

Función IPXRelinquishControl

Devuelve el control a la CPU de una estación.

Entradas : - Ninguna.

Salidas : - Ninguna.

```
void IPXRelinquishControl ()
{
     union REGS inregs,outregs;

     inregs.x.bx = 0xA;
     int86(0x7A,&inregs,&outregs);
}
```

Función IPXDisconnectFromTarget

Informa a un socket específico, que está esperando recibir paquetes, que la aplicación no intenta transmitir ninguno.

Entradas : - **networkAddress**, dirección de la estación.

Salidas : - Ninguna.

```
void IPXDisconnectFromTarget (BYTE networkAddress[12])
{
     union REGS inregs,outregs;
     struct SREGS segregs;

     segread(&segregs);
     segregs.es = segregs.ds;

     inregs.x.bx = 0xB;
     inregs.x.si = (int) networkAddress;
     int86x(0x7A,&inregs,&outregs,&segregs);
}
```

Función IPXGetDataAddress

Obtiene la dirección de un dato.

Entradas : - **data**, dato del que se quiere la dirección.

Salidas : - **addressField**, contendrá la dirección del dato.

```
void IPXGetDataAddress(WORD *data,
                       WORD addressField[2])
{
     struct SREGS segregs;

     segread(&segregs);
     segregs.es = segregs.ds;
```

```
    addressField[1] = (WORD) segregs.ds;
    addressField[0] = (WORD) data;
}
```

Función SPXInitialize

Determina si SPX está instalado sobre el nodo donde la aplicación está corriendo.

Entradas : - Ninguna.

Salidas : - **majorRevisionNumber**, componente del número de versión del protocolo SPX.

- **minorRevisionNumber**, componente del número de versión. Junto con el anterior componen el número de versión del protocolo SPX.
- **maxConnections**, es el número máximo de conexiones soportadas por SPX.
- **availableConnections**, indica cuántas conexiones están disponibles para una aplicación.
- **outregs.h.al**, puede tomar los siguientes valores:

  00h no está instalado,

  FFh está instalado.

```
BYTE SPXInitialize (BYTE *majorRevisionNumber,
                    BYTE *minorRevisionNumber,
                    WORD *maxConnections,
                    WORD *availableConnections)
{
    union REGS inregs,outregs;
    inregs.x.bx = 0x10;
    inregs.h.al = 00;
    int86(0x7A,&inregs,&outregs);
    *majorRevisionNumber  = outregs.h.bh;
    *minorRevisionNumber  = outregs.h.bl;
    *maxConnections       = outregs.x.cx;
    *availableConnections = outregs.x.dx;
    return(outregs.h.al);
}
```

Función SPXEstablishConnetion

Intenta establecer una conexión con un socket que está escuchando.

Entradas : - **retryCount**, indica cuántas veces SPX retransmite los paquetes de confirmación.

- **watchDog**, indica si el proceso watchdog está habilitado o deshabilitado.
- **eventControlBlok**, apunta al ECB que contiene el paquete para establecer la conexión.

Salidas : - **connectionID**, número de indetificación de conexión.

outregs.h.al, puede tomar los siguientes valores:

00h SPX está intentando conectarse con el socket destino,

EFh Tabla de conexiones locales completa,

FDh contador de fragmentos no es 1; el tamaño del buffer no es 42,

FFh socket de transmisión no está abierto.

```
int SPXEstablishConnection (BYTE retryCount,
                            BYTE watchDog,
                            WORD *connectionID,
                            ECB  *eventControlBlock)
{
     union REGS inregs,outregs;
     struct SREGS segregs;
     segread(&segregs);
     segregs.es = segregs.ds;
     inregs.x.bx = 0x11;
     inregs.h.al = retryCount;
     inregs.h.ah = watchDog;
     inregs.x.si = (int) eventControlBlock;
     int86x(0x7A,&inregs,&outregs,&segregs);
     *connectionID = outregs.x.dx;
     return(outregs.h.al);
}
```

Función SPXListenForConnection

Intenta recibir un paquete "establecer conexión".

Entradas : - **retryCount**, indica cuántas veces SPX retransmite los paquetes de confirmación.

- **watchDog**, indica si el proceso watchdog está habilitado o deshabilitado.
- **eventControlBlok**, apunta al ECB que se pondrá a escuchar para establecer una conexión.

Salidas : - Ninguna.

```
void SPXListenForConnection (BYTE retryCount,
                             BYTE watchDog,
                             ECB  *eventControlBlock)
{
     union REGS inregs,outregs;
     struct SREGS segregs;
     segread(&segregs);
     segregs.es = segregs.ds;
     inregs.x.bx = 0x12;
     inregs.h.al = retryCount;
     inregs.h.ah = watchDog;
     inregs.x.si = (int) eventControlBlock;
     int86x(0x7A,&inregs,&outregs,&segregs);
}
```

Función SPXTerminateConnection

Termina una conexión SPX.

Entradas : - **connectionID**, número de indetificación de la conexión.

- **eventControlBlok**, apunta al ECB que *contiene* el paquete de terminación.

Salidas : - Ninguna.

```
void SPXTerminateConnection (WORD connectionIDNumber,
                             ECB  *eventControlBlock)
{
     union REGS inregs,outregs;
     struct SREGS segregs;
     segread(&segregs);
     segregs.es = segregs.ds;
```

```
    inregs.x.bx = 0x13;
    inregs.x.dx = connectionIDNumber;
    inregs.x.si = (int) eventControlBlock;
    int86x(0x7A,&inregs,&outregs,&segregs);
}
```

Función SPXSendSequencedPacket

Envía un paquete SPX.

Entradas : - **connectionID**, número de indetificación de la conexión.

- **eventControlBlok**, apunta al ECB a enviar.

Salidas : - Ninguna.

```
void SPXSendSequencedPacket (WORD ConnectionID,
                             ECB *eventControlBlock)
{
    union REGS inregs,outregs;
    struct SREGS segregs;

    segread(&segregs);
    segregs.es = segregs.ds;

    inregs.x.bx = 0x16;
    inregs.x.dx = ConnectionID;
    inregs.x.si = (int) eventControlBlock;
    int86x(0x7A,&inregs,&outregs,&segregs);
}
```

Función SPXListenForSequencedPacket

Recibe un paquete SPX en secuencia.

Entradas : - **eventControlBlok**, apunta al ECB donde se recibirá el paquete.

Salidas : - Ninguna.

```
void SPXListenForSequencedPacket (ECB *eventControlBlock)
{
    union REGS inregs,outregs;
    struct SREGS segregs;
```

```
    segread(&segregs);
    segregs.es = segregs.ds;

    inregs.x.bx = 0x17;
    inregs.x.si = (int) eventControlBlock;
    int86x(0x7A,&inregs,&outregs,&segregs);
}
```

Función SPXAbortConnection

Aborta una conexión SPX.

Entradas : - **connectionID**, número de indetificación de la conexión.

Salidas : - Ninguna.

```
void SPXAbortConnection (WORD ConnectionID)
{
    union REGS inregs,outregs;

    inregs.x.bx = 0x14;
    inregs.x.dx = ConnectionID;
    int86(0x7A,&inregs,&outregs);
}
```

6.4.11 Desarrollo del Módulo CONEXION (conexion.c)

Módulo de funciones que facilitan el establecimiento de la conexión, intercambio de paquetes y finalización de la conexión. Usa los protocolos SPX/IPX.

Es utilizado por las aplicaciones PCOPIA y PDIR.

Para más detalle sobre las APIs usadas perteneciente al protocolo SPX/IPX consultar las secciones 3.12 APIs del Servicio de Comunicación y 6.4.10 Desarrollo del Módulo IPXSPX (ipxspx.h - ipxspx.c).

Includes

Includes provistos por el compilador.

```
#include <stdio.h>
#include <dos.h>
#include <string.h>
```

Includes correspondientes a módulos desarrollados para realizar la conexión.

```
#include "ipxspx.h"
#include "conexion.h"
```

Función TerminarConexion

Función que termina la conexión indicada en el parámetro ConnectionID y que se encuentra en el socket, Socket.

Entradas : - **Socket**, socket de la aplicación.

- **ConnectionID**, identificador de conexión de la estación local.

Salidas : - ERROR u OKAY según corresponda.

```
extern  int TerminarConexion(WORD Socket,
                             WORD ConnectionID)
{
      ECB       *TerminateECB;
      SPXHeader *TerminateSPX;
```

Solicita memoria para el paquete de terminar conexión, cabecera y ECB. En caso de no alcanzar la memoria muestra el mensaje y termina.

```
      /* Requiere memoria para paquete de terminación
      ** de la conexión.
      */
      if ((TerminateECB = (ECB *) calloc(1,sizeof(ECB))) == NULL) {
            printf("\nMemoria saturada, tratando de alocar para ECB de terminación.\n");
            return ERROR;
      }

      if ((TerminateSPX = (SPXHeader *) calloc(1,sizeof(SPXHeader))) == NULL) {
            printf("\nMemoria saturada, tratando de alocar para SPX de terminación.\n");
            return ERROR;
      }
```

El campo ESRAddress del ECB se inicializa en 0, pues el evento terminar conexión no tiene asociada una rutina de servicio de evento.

```
/* Inicializa el paquete */
memset( &TerminateECB->ESRAddress, 0, 4);
```

El campo ECBSocket se setea con el Socket de la aplicación.

```
TerminateECB->ECBSocket     = Socket;
```

El paquete "terminar conexión" debe tener inicializado el campo FragmentCount en 1, sólo tiene información la cabecera del paquete.

```
TerminateECB->FragmentCount = 1;
```

La función IPXGetDataAddress() asigna la dirección de la cabecera del paquete "terminar conexión" al campo descriptor del fragmento de la cabecera (correspondiente al ECB).

```
IPXGetDataAddress( (WORD *) TerminateSPX,
                   (WORD *) TerminateECB->FragmentDescriptor[0].Address);
```

Asigna la longitud de la cabecera del paquete al campo longitud del descriptor del fragmento.

```
TerminateECB->FragmentDescriptor[0].Size    = sizeof(SPXHeader);
```

LLama a la API de SPX SPXTerminateConnection() para que envíe el paquete "terminar conexión".

```
/* Termina la conexión */
SPXTerminateConnection(ConnectionID, TerminateECB);
```

Cuando el campo InUseFlag toma el valor 0 significa que SPX dejó de usar el ECB. Mientras está en uso, la API IPXRelinquishControl() devuelve el control temporariamente de la estación de trabajo a otros procesos.

```
/* Espera a que se envíe el paquete de terminación */
while ( TerminateECB->InUseFlag )
      IPXRelinquishControl();
```

Libera la memoria usada para el paquete y termina la rutina con OKAY.

```
    /* Libera memoria */
    free((void *) TerminateECB);
    free((void *) TerminateSPX);
    return OKAY;
}
```

Función Escuchar

Función que se encarga de poner la cantidad de paquetes indicada en la variable ventana a escuchar en el Socket.

Entradas : - **ventana**, cantidad de paquetes que se ponen a escuchar.

- **Socket**, socket de la aplicación.
- **ESR**, puntero a la función de servicio de evento.

Salidas : - ERROR u OKAY según corresponda.

```
extern int Escuchar(short ventana, WORD Socket, void (*ESR)())
{
    ECB         *PoolECB; /* Puntero a buffer de ECB */
    SPXHeader   *PoolSPX; /* Puntero a cabecera de paquete */
    int         i;
    char        *data;    /* Puntero a datos de paquete */
```

Ciclo para poner a escuchar la cantidad de paquetes indicado por la variable ventana.

```
    for (i =0 ; i < ventana ; i++) {
```

Solicita memoria para el paquete (ECB, cabecera y dato) que escuchará la llegada de paquetes. En caso de no alcanzar la memoria muestra el mensaje y termina.

```
        /* Requiere memoria para paquete */
        if ((PoolECB = (ECB *) calloc(1, sizeof(ECB))) == NULL ) {
            printf("\nMemoria saturada, tratando de alocar para ECB de escucha.\n");
            return ERROR;
        }
        if ((PoolSPX = (SPXHeader *) calloc(1, sizeof(SPXHeader))) == NULL ) {
            printf("\nMemoria saturada, tratando de alocar para SPX de escucha.\n");
            return ERROR;
        }
        if ((data = (char *) calloc(1, SPXDATOS)) == NULL ) {
```

```
        printf("\nMemoria saturada, tratando de alocar para Datos\n ");
        return ERROR;
}
```

El campo ESRAddress del ECB se inicializa con la dirección de la rutina del servicio de interrupción de evento pasada como parámetro ESR. Esta rutina será invocada cada vez que llega un paquete.

```
/* Inicializa el paquete */
IPXGetProcAddress(ESR, PoolECB->ESRAddress);
```

En el campo ECBSocket se setea con el Socket de la aplicación.

```
PoolECB->ECBSocket = Socket;
```

El paquete "escuchar paquete" debe tener inicializado el campo FragmentCount en 2, pues contendrá la información de la cabecera y los datos de los paquetes que llegan.

```
PoolECB->FragmentCount = 2;
```

La función IPXGetDataAddress() asigna la dirección de la cabecera del paquete "escuchar paquete" al campo descriptor del fragmento de la cabecera (correspondiente al ECB).

```
IPXGetDataAddress( (WORD *) PoolSPX,
                   (WORD *) PoolECB->FragmentDescriptor[0].Address );
```

Asigna la longitud de la cabecera del paquete al campo longitud del descriptor del fragmento correspondiente.

```
PoolECB->FragmentDescriptor[0].Size = sizeof(SPXHeader);
```

Se asigna la dirección de los datos del paquete "escuchar paquete" y su longitud máxima (SPXDATOS) al campo descriptor del fragmento de datos (correspondiente al ECB).

```
IPXGetDataAddress( (WORD *) data,
                   (WORD *) PoolECB->FragmentDescriptor[1].Address );
PoolECB->FragmentDescriptor[1].Size = SPXDATOS;
```

El tipo de paquete (PacketType) indica el tipo de servicio ofrecido o requerido por el mismo. El tipo 5 significa que el paquete es del protocolo SPX.

```
        PoolSPX->PacketType = '5';
```

Se invoca a la API SPXListenForSequencedPacket().

```
        /* Pone paquete a escuchar */
        SPXListenForSequencedPacket(PoolECB);
    }
```

No hubo ningún problema, devuelve OKAY.

```
    return OKAY;
}
```

Función EstablecerConexion

Función que establece la conexión desde el Socket de la estación local al socket dsocket de la estación remota (NodoName).

Devuelve el número de identificación de la conexión local (ConnectionID) y las direcciones de red y nodo de la estación destino.

Entradas : - **NodoName**, nombre del nodo con el que se quiere establecer la conexión.
- **Socket**, socket de la aplicación.
- **dsocket**, socket destino.

Salidas : - **ConnectionID**, identificador de conexión local.
- **dnetwork**, dirección de red destino.
- **dnodo**, dirección de nodo destino.
- ERROR u OKAY según corresponda.

```
extern  int  EstablecerConexion(char *NodoName,
                                WORD Socket,
                                BYTE *dnetwork,
                                BYTE *dnodo,
                                WORD dsocket,
                                WORD *ConnectionID)
{
    ECB             *ConnectionECB;/* Puntero a buffer de ECB */
```

```
SPXHeader       *ConnectionSPX;/*Puntero a cabecera del paquete*/
int             TransportTime, problema;
BYTE            fulldir[LONGNET+LONGNODE+LONGSOCK]; /* dirección completa de estación */
```

Requiere memoria para el paquete (ECB y cabecera) que será enviado para establecer la conexión. En caso de no alcanzar la memoria muestra el mensaje y termina.

```
/* Requiere memoria para paquete de conexión. */
if ((ConnectionECB = (ECB *) calloc(1,sizeof(ECB))) == NULL) {
      printf("\nMemoria saturada, tratando de alocar para ECB de Conexión\n ");
      return ERROR;
}
if ((ConnectionSPX = (SPXHeader *) calloc(1,sizeof(SPXHeader))) == NULL) {
      printf("\nMemoria saturada, tratando de alocar para SPX de Conexión\n ");
      return ERROR;
}
```

El campo ESRAddress del ECB se inicializa en 0, pues el evento establecer conexión no tiene asociada una rutina de servicio de evento.

```
/* Inicializa paquete */
memset( &ConnectionECB->ESRAddress, 0, 4 );
```

El campo ECBSocket se setea con el Socket de la aplicación.

```
ConnectionECB->ECBSocket     = Socket;
```

El paquete "establecer conexión" debe tener inicializado el campo FragmentCount en 1, sólo tiene información la cabecera del paquete.

```
ConnectionECB->FragmentCount = 1;
```

Se asigna la dirección de la cabecera del paquete "establecer conexión" y su longitud al campo descriptor del fragmento de cabecera (correspondiente al ConnectionECB).

```
IPXGetDataAddress( (WORD *) ConnectionSPX,
                   (WORD *)ConnectionECB->FragmentDescriptor[0].Address);
ConnectionECB->FragmentDescriptor[0].Size = sizeof(SPXHeader);
```

El tipo de paquete (PacketType) es 5, pues indica que el paquete es del protocolo SPX.

```
ConnectionSPX->PacketType   = '5';
```

Asigna el número de socket destino al campo Destination.Socket perteneciente a la cabecera del paquete.

```
memcpy(ConnectionSPX->Destination.Socket,(BYTE *) &dsocket, LONGSOCK);
```

Busca la dirección de red y nodo de la estación remota en el archivo de asignación de nombres (mapa.txt) usando la función MapNodeNameToAddress(). Si no existe o hubiera algún error en las direcciones termina con ERROR.

```
/* Busca la dirección del nodo remoto en el archivo mapa.txt
** debe devolver el Destination.Network, Destination.Nodo
** el Destination.Socket= INVITE_SOCKET, usar dest_network,
** dest_nodo
*/
if (MapNodeNameToAddress(NodoName, dnetwork, dnodo) == ERROR)
      return ERROR;
```

Copia las direcciones de red y nodo buscadas anteriormente en los campos que corresponden de la cabecera del paquete.

```
memcpy(ConnectionSPX->Destination.Network, dnetwork, LONGNET);
memcpy(ConnectionSPX->Destination.Node, dnodo, LONGNODE);
```

Se invoca a la función IPXGetLocalTarget(), enviándole la dirección completa del nodo destino (red-nodo-socket) para que retorne la dirección del puente local y el tiempo estimado de la entrega de paquetes.

```
/* Setea la dirección completa del destino Red-Nodo-Socket
** para encontrar la dirección inmediata
*/
memcpy(fulldir, (BYTE *) ConnectionSPX->Destination.Network, LONGNET+LONGNODE+LONGSOCK);

/* Busca la dirección de la estación local */
IPXGetLocalTarget( (BYTE *) fulldir,
                   (BYTE *) ConnectionECB->ImmediateAddress, &TransportTime);
```

Se invoca a la API SPXEstablishConnection() para que establezca la conexión. El primer parámetro que se le pasa es el RetryCount (contador de reintentos) utilizado para establecer la cantidad de veces en que SPX reenviará paquetes de confirmación antes de concluir que el nodo destino no

está funcionando correctamente. El segundo parámetro monitorea la conexión para asegurar que está funcionando apropiadamente cuando no hay tráfico a través de ella.

```
/* RetryCount= 0 SPX usa su propio intervalo de tiempo
** para reenviar paquetes
** watchdog= 1 habilitado para determinar si la conexion
** está bien cuando no hay tráfico
*/
/* Establece la conexión */
SPXEstablishConnection( (BYTE)0, (BYTE)1, ConnectionID, ConnectionECB);
```

Se inicializa la variable problema en falso.

Cuando el campo InUseFlag toma el valor 0 significa que SPX dejó de usar el ECB. Mientras está en uso, la API IPXRelinquishControl() devuelve el control temporariamente de la estación de trabajo a otros procesos.

```
/* Se espera hasta que IPX/SPX deja de usar el ECB.
** IPX/SPX deja de usarlo cuando el campo InUseFlag del
** está en 0
*/
problema = OKAY;
while (ConnectionECB->InUseFlag) {
      IPXRelinquishControl();
}
```

Después de enviar el paquete "establecer conexión" al nodo destino (y reenviarlo un número especificado de veces, si fuese necesario), SPX le da valor al campo CompletionCode del ConnectionECB enviado. Si este campo toma el valor de 0x00 se estableció la conexión, de lo contrario se muestran los mensajes de error.

```
/* Verifica el establecimiento de la conexión */
switch (ConnectionECB->CompletionCode) {
case 0x00:
      /* Se estableció la conexión */
      break;
case 0xED:
      printf("\nNo se estableció la conexión, no se recibió respuesta.\n");
      problema = ERROR;
      break;
default:
```

```
        printf("\nError durante el establecimiento de la conexión.\n");
        problema = ERROR;
    }
```

Se libera la memoria utilizada para el ECB y la cabecera del paquete.

```
    free((void *) ConnectionECB);
    free((void *) ConnectionSPX);
```

Se retorna ERROR u OKAY según la variable problema.

```
    return problema;
}
```

Función EscucharConexion

Función que pone en el Socket un paquete a escuchar la llegada de un paquete de pedido de conexión ("establecer conexión").

Devuelve el número de identificación de la conexión local (ConnectionID).

Entradas : - **Socket**, socket de la aplicación.

Salidas : - **ConnectionID**, identificador de conexión local.
- ERROR y OKAY según corresponda.

```
extern  int  EscucharConexion(WORD Socket,
                              WORD *ConnectionID)
{
    ECB             *ConnectionECB; /* Puntero a ECB */
    SPXHeader       *ConnectionSPX;/*Puntero a cabecera de paquete*/
    int             TransportTime;
```

Solicita memoria para el paquete (ECB y cabecera) que se pondrá a escuchar la llegada de un paquete "establecer conexión". En caso de no alcanzar la memoria muestra el mensaje y termina.

```
    /* Requiere memoria para paquete de escuchar conexión. */
    if ((ConnectionECB = (ECB *) calloc(1,sizeof(ECB))) == NULL) {
        printf("\nMemoria saturada, tratando de alocar para ECB de Conexión\n ");
        return ERROR;
```

```
}
if ((ConnectionSPX = (SPXHeader *) calloc(1,sizeof(SPXHeader))) == NULL) {
      printf("\nMemoria saturada, tratando de alocar para SPX de Conexión\n ");
      return ERROR;
}
```

El campo ESRAddress del ECB se inicializa en 0, pues el evento escuchar conexión no tiene asociada una rutina de servicio de evento.

```
/* Inicializa el paquete */
memset( &ConnectionECB->ESRAddress, 0, 4 );
```

El campo ECBSocket se setea con el Socket de la aplicación.

```
ConnectionECB->ECBSocket     = Socket;
```

El paquete "escuchar conexión" debe tener inicializado el campo FragmentCount en 1, sólo tiene información la cabecera del paquete.

```
ConnectionECB->FragmentCount = 1;
```

Se asigna la dirección de la cabecera del paquete "escuchar conexión" y su longitud al campo descriptor del fragmento de cabecera (correspondiente al ConnectionECB).

```
IPXGetDataAddress( (WORD *) ConnectionSPX,
                   (WORD *) ConnectionECB->FragmentDescriptor[0].Address);
ConnectionECB->FragmentDescriptor[0].Size = sizeof(SPXHeader);
```

El tipo de paquete (PacketType) es 5, pues indica que el paquete es del protocolo SPX.

```
ConnectionSPX->PacketType    = '5';
```

Se invoca a la API SPXListenForConnection() para que escuche un paquete "establecer conexión". El primer parámetro que se le pasa es el RetryCount (contador de reintentos) utilizado para establecer la cantidad de veces en que SPX reenviará paquetes de confirmación antes de concluir que el nodo destino no está funcionando correctamente. El segundo parámetro monitorea la conexión para asegurar que está funcionando apropiadamente cuando no hay tráfico a través de ella.

```
/* RetryCount= 0 SPX usa su propio intervalo de tiempo
** para reenviar paquetes
** watchdog= 1 habilitado para determinar si la conexion
** esta bien cuando no hay trafico
*/
SPXListenForConnection((BYTE) 0, (BYTE) 1, ConnectionECB);
```

Cuando el campo InUseFlag toma el valor 0 significa que SPX dejó de usar el ECB. Mientras está en uso, la API IPXRelinquishControl() devuelve el control temporariamente de la estación de trabajo a otros procesos.

```
/* Se espera hasta que IPX/SPX deja de usar el ECB.
** IPX/SPX deja de usarlo cuando el campo InUseFlag del
** está en 0
*/
while (ConnectionECB->InUseFlag)
    IPXRelinquishControl();
```

Si SPX no puede crear una conexión local porque el socket no fue abierto o la Tabla de Conexiones está llena, SPX escribe en el campo CompletionCode el valor que corresponde. Si SPX no encuentra problemas el campo toma el valor 0x00.

```
/* Verifica que el paquete se queda escuchando */
switch (ConnectionECB->CompletionCode) {
case 0x00:
      memcpy((BYTE *) &ConnectionID, (BYTE *) ConnectionECB->IPXWorkspace, 2);
      break;
default:
      printf("\nLa conexión no ha sido establecida.\n");
      return ERROR;
}
```

Libera la memoria utizada por el ECB y la cabecera del paquete.

```
/* Libera la memoria */
free((void *) ConnectionECB);
free((void *) ConnectionSPX);
```

No hubo problemas, retorna OKAY.

```
return OKAY;
}
```

Función MandarMje

Función que envía un paquete al nodo destino.

Entradas : - **dnetwork**, dirección de red destino.

- **dnodo**, dirección de nodo destino.
- **dsocket**, socket destino.
- **Socket**, socket de la aplicación.
- **ConnectionID**, identificador de conexión local.
- **dato**, datos a enviar.
- **longitud**, longitud de los datos.
- **tipo**, tipo de dato a enviar.

Salidas : - **ConnectionID**, identificador de conexión local.

- ERROR y OKAY según corresponda.

```
extern int MandarMje (BYTE      *dnetwork,
                      BYTE      *dnodo,
                      BYTE      *dsocket,
                      WORD      Socket,
                      WORD      ConnectionID,
                      void      *dato,
                      int       longitud,
                      BYTE      tipo)
{
      ECB             *SendECB; /* Puntero a ECB */
      SPXHeader       *SendSPX; /* Puntero a cabecera de paquete */
      int             TransportTime;
      int             problem;
      BYTE            fulldir[LONGNET+LONGNODE+LONGSOCK];
```

Requiere memoria para el paquete (ECB y cabecera) que se enviará a la estación remota. En caso de no alcanzar la memoria muestra el mensaje y termina.

```
      /* Requiere memoria para paquete */
      if ((SendECB = (ECB *) calloc(1, sizeof(ECB))) == NULL) {
            printf("\nMemoria saturada, tratando de alocar para ECB de Paquete.\n ");
```

```
        return ERROR;
    }
    if ((SendSPX = (SPXHeader *) calloc(1, sizeof(SPXHeader))) == NULL) {
        printf("\nMemoria saturada, tratando de alocar para SPX de Paquete.\n ");
        return ERROR;
    }
```

El campo ESRAddress del ECB se inicializa en 0, pues el evento enviar paquetes en secuencia no tiene asociado una rutina de servicio de evento.

```
/* Inicializa el paquete */
memset( &SendECB->ESRAddress,0,4 );
```

El campo ECBSocket se setea con el Socket de la aplicación.

```
SendECB->ECBSocket      = Socket;
```

El paquete debe tener inicializado el campo FragmentCount en 2, pues contendrá la información de la cabecera y los datos del paquete que enviará.

```
SendECB->FragmentCount = 2;
```

Se asigna la dirección de la cabecera del paquete y su longitud al campo descriptor del fragmento de cabecera (correspondiente al SendECB).

```
IPXGetDataAddress( (WORD *) SendSPX,
                   (WORD *) SendECB->FragmentDescriptor[0].Address );
SendECB->FragmentDescriptor[0].Size    = sizeof(SPXHeader);
```

Se asigna la dirección de los datos del paquete (dato) y su longitud (variable longitud) al campo descriptor del fragmento de datos (correspondiente al SendECB).

```
IPXGetDataAddress( (WORD *) dato,
                   (WORD *) SendECB->FragmentDescriptor[1].Address );
SendECB->FragmentDescriptor[1].Size    = longitud;
```

El tipo de paquete (PacketType) es 5, pues indica que el paquete es del protocolo SPX.

```
SendSPX->PacketType     = '5';
```

En el campo DataStreamType se indica el tipo de dato del paquete (pedido, dato, error). Este valor está contenido en el parámetro de entrada tipo.

```
SendSPX->DataStreamType = tipo;
```

Se le da valor a los campos de dirección de red, nodo y socket destino con los parámetros de entrada.

```
memcpy(SendSPX->Destination.Socket,  dsocket,  LONGSOCK);
memcpy(SendSPX->Destination.Network, dnetwork, LONGNET);
memcpy(SendSPX->Destination.Node,    dnodo,    LONGNODE);
```

Se invoca a la función IPXGetLocalTarget(), enviándole la dirección completa del nodo destino (red-nodo-socket) para que retorne la dirección del puente local y el tiempo estimado de la entrega del paquete.

```
/* Setea la dirección completa del destino Red-Nodo-Socket
** para encontrar la dirección inmediata
*/
memcpy((BYTE *) fulldir, (BYTE *) SendSPX->Destination.Network,
      LONGNET+LONGNODE+LONGSOCK);
/* Busca dirección de estación local */
IPXGetLocalTarget( (BYTE *) fulldir,(BYTE *)SendECB->ImmediateAddress,
                  &TransportTime);
```

Se inicializa la variable problema indicando que no tiene error.

Se invoca a la rutina SPXSendSequencedPacket().

```
problem = OKAY;
SPXSendSequencedPacket(ConnectionID, SendECB);
```

Cuando el campo InUseFlag toma el valor 0 significa que SPX dejó de usar el ECB. Mientras está en uso, la API IPXRelinquishControl() devuelve el control temporariamente de la estación de trabajo a otros procesos.

```
/* Se espera hasta que IPX/SPX deja de usar el ECB.
** IPX/SPX deja de usarlo cuando el campo InUseFlag del
** está en 0
*/
while (SendECB->InUseFlag)
      IPXRelinquishControl();
```

Cuando SPX termina con su intento de enviar el *paquete* deja en el campo CompletionCode un valor para indicar si hubo algún problema. Si lo hubo la variable problem toma el valor de ERROR.

```
/* Verifica si envío el paquete */
switch (SendECB->CompletionCode) {
case 0x00:
      break;
case 0xED:
      printf("\nFalla de conexión.\n");
      problem = ERROR;
      break;
case 0xEC:
      printf("\nLa conexión fue terminada por el otro nodo.\n ");
      problem = ERROR;
      break;
default:
      printf("\nError ocurrido tratando de enviar un paquete.\n");
      problem = ERROR;
      break;
}
```

Se libera la memoria tomada para el ECB y la cabecera del paquete.

```
/* Libera memoria */
free((void *) SendECB);
free((void *) SendSPX);
```

Retorna el valor contenido en la variable problem.

```
return problem;
}
```

Función MapNodeNameToAddress

Función que busca la dirección de red (dir_red) y nodo (dir_nodo) del nombre del nodo (NodoName) en el archivo de asignación de nombres de nodos mapa.txt.

Para recordar el formato del archivo remitirse a la sección 6.3.1.

Entradas : - **NodoName**, nombre del nodo.

- **dsocket**, socket destino.
- **dir_red**, dirección de red.
- **dir_nodo**, dirección de nodo.

Salidas : - ERROR y OKAY según corresponda.

```
extern int MapNodeNameToAddress(char *NodoName,
                                BYTE *dir_red,
                                BYTE *dir_nodo)
{
      FILE  *mapa;          /* Handle del archivo NODONAME */
      short nbre,           /* Indica si se formó el nombre de un nodo */
            encontro,       /* Indica si se encontró el nodo */
            i;              /* Indice de mapanodo */
      char  mapanodo[10]; /*Nombre de los nodos que están en NODONAME*/
      char  red[LONGNET * 2];   /* Dirección de red en caracteres */
      char  nodo[LONGNODE * 2]; /* Dirección de nodo en caracteres */
      int   ch;                 /* Caracter leído del archivo */
```

Abre el archivo mapa.txt para lectura. Si no existe muestra un mensaje de error.

```
      /* Abre el archivo que contiene los nodos y sus direcciones */
      if ((mapa = fopen(NODOMAPA, "r")) == NULL) {
            printf("\nEl archivo %s no existe", NODOMAPA);
            return ERROR;
      }

      /* posisiona al comienzo del archivo */
      fseek(mapa, SEEK_SET, 0);
```

Comienza a buscar el nombre del nodo en el archivo. Mientras no se encuentre (!encontro) y no se llegue al final del archivo se lee un caracter del mismo.

```
      /* Busca nombre del nodo */
      ToUpper(NodoName, NodoName);
      i = 0;
      nbre = encontro = 0;
      memset(mapanodo, '\0', sizeof(mapanodo));
      while (!encontro && (ch = fgetc(mapa)) != EOF ) {
            switch (ch) {
```

Si el caracter leído es un blanco ' ' entonces se formó un nombre de nodo y se setea la variable nbre en verdadero.

```
case ' ' :
     if (!nbre) {
          nbre = 1;
```

Compara el nombre leido del archivo en mapanodo con el nombre del nodo buscado NodoName. Si son iguales se indica en la variable encontro.

```
/* Se encontró el nodo bucado */
if (!strcmp(mapanodo, NodoName)) {
     encontro = 1;
```

Mientras el caracter leído no sea fin del archivo, un fin de línea o un blanco ' ' se leen caracteres del archivo para formar la dirección de red del nodo en la variable red.

```
/* Busca dirección de red */
i=0;
while ((ch = fgetc(mapa)) != EOF && ch != '\n' && ch != ' ' )
     red[i++] = ch;
red[i] = '\0';
```

Se controla que la variable red tenga la longitud que le corresponde a una dirección de red. Si no es así se muestra un ERROR.

```
/* Verifica longitud de
** dirección de red
*/
if (i > LONGNET * 2) {
     printf("\nLa dirección de red es incorrecta %s", red);
     return ERROR;
}
```

Se transforma la dirección de red leida del archivo como cadena de caracteres (red) a hexadecimal (dir_red). Si existiera algún error se muestra el mensaje por pantalla.

```
/* Transforma la dirección de red
** a hexadecimal.
*/
if (StrToHexa(red, dir_red) == ERROR) {
```

```
                printf("\nLa dirección de red es incorrecta %s", red);
                return ERROR;
        }
```

Luego formará la dirección del nodo en la variable nodo leyendo el archivo mientras el caracter leído no sea fin del archivo, un fin de línea o un blanco ' '.

```
        /* Busca nodo */
        i=0;
        while ((ch = fgetc(mapa)) != EOF && ch != '\n' && ch != ' ' )
                nodo[i++] = ch;
        nodo[i] = '\0';
```

Se controla que la variable nodo tenga la longitud que le corresponde a una dirección de nodo. Si no es así se muestra un ERROR.

```
        /* Verifica longitud del nodo */
        if (i > LONGNODE * 2) {
                printf("\nLa dirección de nodo es incorrecta %s",
                        nodo);
                return ERROR;
        }
```

Se transforma la dirección de nodo leída del archivo como cadena de caracteres (nodo) a hexadecimal (dir_nodo), por medio de la función StrToHexa(). Si existiera algún error se muestra el mensaje por pantalla.

```
        /* Transforma el nodo a hexadecimal */
        if (StrToHexa(nodo, dir_nodo) == ERROR){
                printf("\nLa dirección del nodo es incorrecta %s",
                        nodo);
                return ERROR;
        }

        break;
    }
}
break;
```

Cuando el caracter es un fin de línea '\n' significa que el nombre de nodo contenido en mapanodo no es el nodo que se está buscando. Por lo tanto se ponen las variables en 0 y se limpia mapanodo.

```
        case '\n' :
              /* Encontró un fin de línea */
              i = 0;
              nbre = 0;
              memset(mapanodo, '\0', sizeof(mapanodo));
              break;
```

Cuando el caracter leído no es un blanco ' ' ni un fin de línea '\n' y aún no se ha formado un nombre completo, el caracter se incorpora a mapanodo, variable que contendrá el nombre de un nodo que está en el archivo.

```
        default   :
              /* Busca nombre de nodo */
              if (!nbre)
                    mapanodo[i++] = ch;
        }
  }
```

Se cierra el archivo de mapeo.

Si no se encontró el nodo se muestra un mensaje de error y la rutina retorna ERROR.

```
  fclose(mapa);
  if (!encontro) {
        printf("\nEl nodo %s no exite.\n", NodoName);
        return ERROR;
  }
```

Se encontró el nodo y las direcciones de red y nodo tienen la longitud correcta.

```
  return OKAY;
}
```

Función StrToHexa

Función que convierte una cadena de caracteres (origen) a una cadena de hexadecimales (destino).

Entradas : - **origen**, cadena de caracteres a convertir.

Salidas : - **destino**, cadena de hexadecimales.
ERROR y OKAY según corresponda.

```
extern int StrToHexa(char *origen, BYTE *destino)
{
      unsigned char hexa2, hexa1;
      short i, j, max;
```

Se inicializa la variable max con la longitud de la cadena a convertir y se comienza a recorrer la cadena origen.

```
      /* Recorre cadena de caracteres */
      max = strlen(origen);
      j = 0;
      for ( i= 0; i < max ; i++) {
```

Se toma el caracter i de la cadena y se verifica que contega un dígito hexadecimal. Si no es así devuelve ERROR.

```
            hexa2 = origen[i];
            /* Verifica que sea un digito hexadecimal */
            if (!isxdigit(hexa2))
                  return ERROR;
```

Si es un dígito alfabético se le suma 0x09.

```
            /* Convierte un caracter */
            if (isalpha(hexa2))
                  hexa2 += 0x09;
```

Si es el dígito impar se lo rota un byte. Si en cambio es un dígito par se lo concatena al dígito impar obtenido en el ciclo anterior.

```
            /* Convierte a hexadecimal */
            if (!(i % 2)) {
                  hexa1 = hexa2 << 4;
            }
            else {
                  destino[j++] = hexa1 ^ (hexa2 & 0xF);
            }
      }
```

No hubo errores, devuelve OKAY.

```
      return OKAY;
}
```

6.4.12 Desarrollo del Modulo DIRARCH (dirarch.c)

Módulo de funciones que facilita el manejo y verificación de la sintaxis de nodos, directorios y archivos.

Es utilizado por las aplicaciones PCOPIA y PDIR.

Includes

Includes provistos por el compilador.

```
#include <stdio.h>
#include <direct.h>
#include <string.h>
#include <ctype.h>
```

Includes correspondientes a módulos desarrollados para realizar las aplicaciones.

```
#include "dirarch.h"
#include "ipxspx.h"
```

Función ToUpper

Función que convierte una cadena de caracteres de minúscula (cadena) a mayúcula (upper).

Entradas : - **cadena**, cadena a transformar.

Salidas : - **upper**, cadena transformada.

```
void ToUpper(char *cadena, char *upper)
{
    short i;
```

Recorre la cadena para transformarla.

```
    for(i = 0; cadena[i] != '\0'; i++) {
```

Transforma los caracteres alfabéticos que están en minúscula.

```
        if (isalpha(cadena[i]) && !isupper(cadena[i])) {
            upper[i] = (int) cadena[i] - 32;
        }
```

```
        else {
            upper[i] = cadena[i];
        }
    }
```

Pone el fin de la cadena.

```
    upper[i] = '\0';
}
```

Función file_exists

Función que verifica la existencia de un archivo (filename).

Entradas : - **filename**, nombre del archivo.

Salidas : - **1**, el archivo existe.
- **0**, el archivo no existe.

```
int file_exists(char *filename)
{
```

La función access determina si el archivo exite, devolviendo en 0.

```
    return (access(filename, 0) == 0);
}
```

Función dir_exists

Función que verifica la existencia de un directorio (dirname).

Entradas : - **dirname**, nombre de un directorio.

Salidas : - **1**, el directorio existe.
- **0**, el directorio no existe.

```
int dir_exists(char *dirname)
{
    char    pwd[MAXPATH];
    short   okay;
```

Se guarda el directorio actual en la variable pwd.

```
    /* Guarda el directorio actual */
    getcwd(pwd, MAXPATH);
```

Si se trata del directorio raíz no se debe sacar el caracter del final de dirname '\'. Se consideran los siguientes casos: "c:\" o "\".

```
    /* Si es el directorio raíz no se debe sacar el \ final,
    ** se considera si tiene drive o no lo tiene  (c:\ o \)
    */
    if ( ( !( dirname[1] == ':' &&
              strlen(dirname) == 3) &&
              dirname[strlen(dirname)-1] == '\\' )  &&
           !( dirname[0] == '\\' && strlen(dirname) == 1)
       )
          dirname[strlen(dirname)-1] = '\0';
```

Se cambia al directorio indicado en dirname. Si no se puede la variable okay toma el valor ERROR, si se puede toma el valor OKAY.

```
    /* Cambia de directorio */
    if (chdir(dirname))
          okay = ERROR;
    else
          okay = OKAY;
```

Se posiciona nuevamente en el directorio anterior.

```
    /* Vuelve al directorio anterior */
    chdir(pwd);
```

Retorna el valor okay.

```
    return okay;
}
```

Función VerifNbreArch

Función que valida el nombre de un archivo (filename), considerando si se aceptan caracteres wildcards.

Entradas : - **filename**, nombre de archivo.

- **wildcards**, 1 indica que se aceptan, 0 indica que no se aceptan.

Salidas : - ERROR u OKAY según corresponda.

```
int  VerifNbreArch(char *filename, short wildcards)
{
```

Se llama a la función VerifFormatoArch() para analizar si la sintaxis del nombre del archivo es correcta.

```
    /* Verifica Formato de Archivo */
    if (VerifFormatoArch(filename, wildcards) == ERROR)
         return ERROR;
```

Un archivo no puede llamarse con el nombre de un dispositivo, por esta razón se controla.

```
    /* Verifica que no se llame como algún dispositivo */
    if (!strcmp(filename, "CON")    ||
        !strcmp(filename, "AUX")    ||
        !strcmp(filename, "PRN")    ||
        !strcmp(filename, "CLOCK$") ||
        !strcmp(filename, "LPT1")   ||
        !strcmp(filename, "LPT2")   ||
        !strcmp(filename, "LPT3")   ||
        !strcmp(filename, "NULL")   ||
        !strcmp(filename, "COM1")   ||
        !strcmp(filename, "COM2")   ||
        !strcmp(filename, "COM3")   ||
        !strcmp(filename, "COM4") )
          return ERROR;

    return OKAY;
}
```

Función VerifFormatoArch

Función que valida el nombre de un archivo (filename), considerando si se aceptan caracteres wildcards.

Entradas : - **filename**, nombre de archivo.

- **wildcards**, 1 indica que se aceptan, 0 indica que no se aceptan.

Salidas : - ERROR u OKAY según corresponda.

```
int  VerifFormatoArch(char *filename, short wildcards)
{
```

Si la longitud del nombre del archivo es mayor de 8 caracteres es un error.

```
    /* Verifica longitud del archivo */
    if (strlen(filename) >= MAXFILE)
          return ERROR;

    ToUpper(filename, filename);
```

Controla que no posea los caracteres inválidos ',', '/', '.'.

```
    /* Verifica que no tenga ',' '/' '.'  */
    if (strchr(filename, ',') || strchr(filename, '/') || strchr(filename, '.'))
          return ERROR;
```

Si la varible wildcards tiene el valor 0, se verifica que no se hayan ingresado los caracteres '?' y/o '*'.

```
    /* Si no debe tener wildcards lo considera error */
    if (!wildcards && ( strchr(filename, '?') || strchr(filename, '*')) )
          return ERROR;
```

El nombre del archivo no tiene errores.

```
    return OKAY;
}
```

Función VerifPath

Función que valida el nombre de un directorio (path), considerando si se aceptan wildcards.

Entradas : - **path**, nombre de directorio.

- **wildcards**, 1 indica que se aceptan, 0 indica que no se aceptan.

Salidas : - ERROR u OKAY según corresponda.

```
int  VerifPath(char *path)
{
    char  *subdir1;
    char  *subdir2;
    char  subdir[MAXDIR], *i, *j;
```

Considera que el caracter '/' es inválido para formar parte del nombre de un directorio.

```
/* Descarta caracter inválido */
if (strchr(path, '/') )
      return ERROR;
```

Si el directorio no comienza con un caracter '\' es inválido.

```
/* Verifica que comience con '\' */
if (*path != '\\')
      return ERROR;
```

Se divide el directorio en los subdirectorios que lo componen y se verifica que sean correctos los nombres que recibe.

```
/* Toma cada subdirectorio que compone el path y lo valida */
subdir1 = path;
do {
```

Busca el fin del nombre del subdirectorio a analizar.

```
      /* Busca el subdirectorio */
      subdir2 = strchr(subdir1, '\\');
```

Recorre el directorio (subdir1), por medio del índice i, para formar el nombre del subdirectorio en la variable subdir, usando el índice j.

```
      for(i = subdir1, j = subdir; subdir2 != NULL && i != subdir2; i++, j++)
            *j = *i;
      *j = '\0';
```

Si se formó un subdirectorio controla que el nombre sea correcto por medio de la función VerifFormatoArch(), aquí no se permiten wildcards.

```
      /* no considera wildcards */
      if (*subdir != '\0') {
            if (VerifFormatoArch(subdir, 0) == ERROR)
                  return ERROR;
      }
```

La variable de subdirectorios, subdir1, es apuntada al comienzo del próximo subdirectorio a analizar.

```
      subdir1 = ++subdir2;
} while ( *subdir1 != '\0');
```

No existe ningún error, la función retorna OKAY.

```
    return OKAY;
}
```

Función ExisteNode

Función que verifica la existencia del nombre de una estación en el archivo de asociación de estaciones con sus direcciones de red y nodo (mapa.txt).

Entradas : - **NodoName**, nombre del nodo.

Salidas : - ERROR u OKAY según corresponda.

```
int ExisteNode(char *NodoName)
{
    FILE  *mapa;        /* Handle del archivo NODONAME */
    short nbre,         /* Indica si se formó el nombre
                           de un nodo */
          encontro,     /* Indica si se encontró el nodo */
          i;            /* Indice de mapanodo */
    char  mapanodo[10]; /*Nombre de los nodos que están en NODONAME*/
    int   ch;           /* Caracter leído del archivo */
```

Abre el archivo mapa.txt usando el define NODOMAPA para lectura. Si el archivo no existe o se produce algún otro error al abrirlo la función retorna el valor -2.

```
    ToUpper(NodoName, NodoName);
    if ((mapa = fopen(NODOMAPA, "r")) == NULL) {
          return -2;
    }
    /* posisiona al comienzo del archivo */
    fseek(mapa, SEEK_SET, 0);
```

Inicializa las variables que indican que formó un nombre (nbre) y si el nombre formado es el nombre del nodo buscado (encontró).

```
    /* Busca nombre del nodo */
    i = 0;
    nbre = encontro = 0;
    memset(mapanodo, '\0', sizeof(mapanodo));
```

Mientras no encuentre el nodo y el archivo no llegue a su fin, lo recorre guardando el caracter leído en ch.

```
while (!encontro && (ch = fgetc(mapa)) != EOF ) {
      switch (ch) {
```

Si el caracter leído es un blanco ' ' entonces se formó un nombre de nodo y se setea la variable nbre en verdadero.

```
      case ' ' :
            /* Se encontró un nombre de nodo */
            if (!nbre) {
                  nbre = 1;
```

Compara el nombre leído del archivo en mapanodo con el nombre del nodo buscado NodoName. Si son iguales se indica en la variable encontro.

```
                  /* Se encontró el nodo bucado */
                  if (!strcmp(mapanodo, NodoName)) {
                        encontro = 1;
                        break;
                  }
            }
            break;
```

Cuando el caracter es un fin de línea '\n' significa que el nombre de nodo contenido en mapanodo no es el nodo que se está buscando. Por lo tanto se ponen las variables en 0 y se limpia mapanodo.

```
      case '\n' :
            /* Se terminó la línea */
            i = 0;
            nbre = 0;
            memset(mapanodo, '\0', sizeof(mapanodo));
            break;
```

Cuando el caracter leído no es un blanco ' ' ni un fin de línea '\n' y aún no se ha formado un nombre completo, el caracter se incorpora a mapanodo, variable que contendrá el nombre de un nodo que está en el archivo.

```
      default   :
            /* Forma el nombre del nodo */
            if (!nbre)
                  mapanodo[i++] = ch;
      }
}
```

Se cierra el archivo de mapeo.

Si no se encontró el nodo se retorna ERROR.

```
    fclose(mapa);
    if (!encontro)
          return ERROR;
```

Se encontró el nodo retorna OKAY.

```
    return OKAY;
}
```

Función fnsplit

Función que es capaz de dividir el nombre de un archivo con su camino absoluto (path) en disco-camino-archivo-extensión (drive-dir- name-ext). En la variable flags se indican los componentes.

Entradas : - **path**, formado por disco-directorio -archivo-extensión.

Salidas : - **drive**, disco.

- **dir**, directorio.

- **name**, nombre de archivo.

- **ext**, extensión del archivo.

- **flags**, indicador de componentes de path.

- ERROR u OKAY según corresponda.

```
extern int fnsplit( char    *path,
                    char    *drive,
                    char    *dir,
                    char    *name,
                    char    *ext,
                    unsigned int *flags)
{
    char *disco, *camino;
    char *subdir, *ultdir, *archivo, *archivof, *auxext;
    long length;
```

Inicializa todas las variables en NULL para comenzar a descomponer path. La variable flags toma el valor de 0 y se limpian todas las cadenas de caracteres en que se descompone path.

```
/* Setea para comenzar a descomponer */
disco = camino = subdir = ultdir = archivo = archivof = auxext = NULL;
*flags = 0x0;
memset(drive, '\0', MAXDRIVE);
memset(dir,   '\0', MAXDIR);
memset(ext,   '\0', MAXEXT);
memset(name,  '\0', MAXFILE);
```

Busca el disco a través del caracter ':'.

```
/* Busca el disco y marca la dirección de comienzo del path */
disco = strchr(path, ':');
if (disco) {
```

Toma la longitud del disco, si es menor que MAXDRIVE, la longitud es correcta. En este caso el disco es copiado en la variable drive, se indica en flags que el disco existe y se inicializa la variable camino para buscar el camino.

```
        /* Tiene disco */
        length= disco - path + 1;
        if (length < MAXDRIVE) {
              strncpy(drive, path,  length);
              *flags |= DRIVE;
              camino = disco + 1;
        }
        else {
              printf("\nLa longitud del disco es incorrecta.\n");
              return ERROR;
        }
}
```

El path no tiene disco, se inicializa la variable camino para buscar el camino.

```
else {
      /* No tiene disco */
      camino = path;
}
```

Verifica si contiene wilcards. Si es así lo indica en flags.

```
/* Busca wildcards */
if (strchr(camino, '*') != '\0' || strchr(camino, '?') != '\0')
      *flags |= WILD;
```

Busca el path completo, hasta encontrar el último caracter '\'.

```
/* Busca el path completo por medio de  '\' */
subdir = ultdir = camino;
while ((subdir = strchr(subdir, '\\')) != '\0') {
      ultdir = subdir;
      ++subdir;
}
```

Si no termina con '\' puede ser que se trate de '.' o '..', por lo tanto busca éstos directorios.

```
/* Si no termina con '\' puede que se trate de '.' o '..' */
if (*ultdir != '\\'){
      short i = 0;

      /* Busca el path por '.' */
      subdir = ultdir = camino;
      for (i=0;  i < 2 && subdir[i] == '.'; i++)
            ;
      ultdir = &subdir[i-1];
 }
```

Si termina con un caracter '\' o '.' se encontró el directorio. En este caso calcula su longitud y si es menor que MAXPATH es correcta, si no muestra un mensaje de error.

```
 if (*ultdir == '\\' || *ultdir == '.') {
      /* Tiene path */
      length= ultdir - camino + 1;
      if (length < MAXPATH) {
            strncpy(dir, camino,  length);
      }
      else {
            printf("\nLa longitud del camino es incorrecta.\n");
            return ERROR;
      }
```

En la variable flags se indica que se encontró el directorio y se inicializa la variable archivo para buscar el archivo.

```
        *flags |= DIRECTORY;
        archivo = archivof = ultdir + 1;
    }
```

No se encontró el directorio, se inicializa la variable archivo para buscar el archivo.

```
    else {
          /* No tiene path */
          archivo = archivof = camino;
    }
```

Si la variable archivo no es NULL o un fin de cadena comienza a descomponer el archivo en sus componentes: nombre y extensión.

```
    /* Descompone el archivo en el nombre y la extensión */
    if (archivo != NULL && archivo != '\0') {
```

Busca el primer caracter '.' desde el final de la cadena para encontrar el comienzo la extensión del archivo.

Si la encuentra, calcula su longitud, en caso de que sea mayor a MAXEXT muestra un mensaje de error.

```
        /* Busca la extensión del archivo */
        for (auxext = archivo + strlen(archivo) - 1;
                    auxext >= archivo && *auxext != '.'; auxext--)
            ;
        if (auxext >= archivo && *auxext == '.') {
            length = archivo + strlen(archivo) - auxext;
            /* Se pone uno más para reconocer el error de la
            ** extensión
            */
            if (length < MAXEXT) {
                strncpy(ext, auxext,  length);
            }
            else {
                printf("\nLa longitud de la extensión es incorrecta.\n");
                return ERROR;
            }
```

En la variable flags se indica que se encontró la extensión y se inicializa la variable archivof que indica el fin del archivo, si éste existiese.

```
                *flags |= EXTENSION;
                archivof = auxext ;
        }
        else {
                /* No tiene extensión */
                archivof = archivo + strlen(archivo);
        }
```

Si la variable archivo no es NULL y el comienzo del archivo (archivo) es distinto que su fin (archivof) se encontró el nombre del archivo.

Se calcula la longitud del archivo y si es incorrecta se muestra el mensaje de error por pantalla.

```
        /* Busca nombre de archivo */
        if (archivo != NULL && archivo != archivof) {
                length = archivof - archivo;
                if (length < MAXFILE) {
                        strncpy(name, archivo,  length);
                }
                else {
                        printf("\nLa longitud del archivo es incorrecta.\n");
                        return ERROR;
                }
```

En la variable flags se indica que se encontró el nombre del archivo.

```
                *flags |= FILENAME;
        }
    }
```

La rutina termina sin problemas.

```
    return OKAY;
}
```

6.5 Pruebas

Las técnicas que se han utilizado para diseñar los casos de prueba corresponden a los enfoques de caja negra (dirigidos por la entrada/salida) y caja blanca (dirigidos por la lógica).

Los métodos de caja negra ignoran la estructura interna del programa y utilizan sólo las especificaciones. Las consideraciones para seleccionar los casos se remiten exclusivamente al análisis sistemático del universo de entradas y salidas y las relaciones que deben ligar las primeras con las segundas.

Los métodos de caja blanca parten del análisis de la estructura interna del programa. Los datos se seleccionan en función de las ramas del programa que se desea testear.

Los métodos de caja negra pueden permitir el descubrimiento de errores de diseño pero no garantizan que se haya ejecutado todas las partes del programa, ni se sabrá si no existen errores por tratamiento especial de ciertos valores. Esto no puede asegurarse sin estudiar la lógica interna. Por otra parte, es mucho más económico el análisis de la lógica que el del universo de situaciones para determinar si se ha testeado "todo" el programa. La idea es usar métodos que correspondan a ambos enfoques.

Para considerar que un programa ha sido completamente probado en su lógica consideraremos el criterio de cubrimiento por condiciones múltiples individualizadas. En él se toman todas las condiciones del programa, cada término de las condiciones se prueba individualmente, de manera que sea el único responsable del valor del predicado.

Cuando la conexión entre términos es "o", se hacen dos grupos de prueba: un caso con todos los términos falsos (predicado falso); y por cada término, un caso con ese término verdadero y los restantes falsos (predicado verdadero).

Cuando el predicado está conectado por una "y", sólo se desprecian los casos que tienen más de un término falso.

6.5.1 Modulo pdir

1 int main(int argc, char **argv)

Entrada: argc (cantidad de argumentos)

argv (argumentos ingresados)

Salida : ERROR

OKAY

Condición	Salida Verdadero		Salida Falso	
argc > 4	1	La cantidad de argumentos ingresados es inválida.	2	La cantidad de argumentos ingresados es válida.
argc == 2	3	La cantidad de argumentos es 2.	4 5	argc == 1 (sólo comando). argc > 2 (comando con opciones).
argv[1] == '?'	6	El segundo argumento es '?'.	7	El segundo argumento no es '?'.
argc > 2	8	La cantidad de argumentos es mayor que 2.	9	La cantidad de argumentos es menor o igual que 2.
VerificarOp() == ERROR	10	Las opciones no son correctas.	11	Las opciones son correctas.
VerificarArg() == ERROR	12	El pedido es incorrecto.	13	El pedido es correcto.
Pdir() == ERROR	14	No puede desplegar el pedido.	15	Despliega el pedido.

Casos de Prueba

1- Entrada : pdir /p /c ? nodo:c:*.*

Salida : "Error de sintaxis: pdir [/p /c <nodo>:\camino\archivo"
retorna ERROR

Condiciones : 1

2- Entrada : pdir

Salida : Despliega Help
retorna OKAY

Condiciones : 2, 4

3- Entrada : pdir ?

Salida : Despliega Help
retorna OKAY

Condiciones : 2, 3, 6

4- Entrada : pdir ? nodo1:c:*.*

Salida : retorna ERROR (error en opciones)

Condiciones : 2, 5, 6, 10

5- Entrada : pdir /c

Salida : retorna ERROR (el nodo fuente es obligatorio)

Condiciones : 2, 3, 7

6- Entrada : pdir /m nodo1:c:*.*

Salida : retorna ERROR (error en opciones)

Condiciones : 2, 8, 10

7- Entrada : pdir nodo1:*.*

Salida : retorna ERROR (el disco fuente es obligatorio)

Condiciones : 9, 12

8- Entrada : pdir nodo1:c:*.*

Salida : retorna ERROR (por ej. falla de conexión)

Condiciones : 2, 3, 7, 9, 11, 13, 14

9- Entrada : pdir /p nodo1:c:*.*

Salida : Despliega el pedido
retorna OKAY

Condiciones : 2, 5, 7, 8, 11, 13, 15

2 void Help(void)

Despliega el help correctamente.

**3 int VerificarOp(int argc, char *argv[],
unsigned int *opcion)**

Entrada: argc (cantidad de argumentos de la línea de comando)

argv (argumentos ingresados desde la línea de comando)

Salida : opcion (opción ingresada)

ERROR

OKAY

Condición	Salida Verdadero		Salida Falso	
argv[i] == '/p'	1	La opción ingresada es /p (paginar).	2	La opción ingresada no es /p (paginar).
argv[i] == '/P'	3	La opción ingresada es /P (paginar).	4	La opción ingresada no es /P (paginar).
opcion == SETPAG	5	Ya se ingresó la opcion /p o /P (paginar).	6	No se ingresó la opción /p o /P (paginar).
argv[i] == '/c'	7	La opción ingresada es /c (encolumnar).	8	La opción ingresada no es /c (encolumnar).
argv[i] == '/C'	9	La opción ingresada es /C (encolumnar).	10	La opción ingresada no es /C (encolumnar).
opcion == SETCOL	11	Ya se ingresó la opcion /c o /C (encolumnar).	12	No se ingresó la opción /c o /C (encolumnar).

Casos de Prueba

1- Entrada : pdir /p nodo:c:*.* (paginar)

Salida : opcion = SETPAG
retorna OKAY

Condiciones : 1, 4, 6, 8, 10

2- Entrada : pdir /P nodo:c:*.* (paginar)

Salida : opcion = SETPAG
retorna OKAY

Condiciones : 2, 3, 6, 8, 10

3- Entrada : pdir /p /P nodo:c:*.*

Salida : "Error de sintaxis: pdir [/p /c
<nodo>:\camino\archivo"
retorna ERROR

Condiciones : 1, 3, 5

4- Entrada : pdir /c nodo:c:*.* (encolumnar)

Salida : opcion = SETCOL
retorna OKAY

Condiciones : 2, 4, 6, 7, 10, 12

5- Entrada : pdir /C nodo:c:*.* (encolumnar)

Salida : opcion = SETCOL
retorna OKAY

Condiciones : 2, 4, 6, 8, 9, 12

6- Entrada : pdir /c /C nodo:c:*.*

Salida : "Error de sintaxis: pdir [/p /c
<nodo>:\camino\archivo"
retorna ERROR

Condiciones : 2, 4, 6, 7, 9, 11

7- Entrada : pdir /p /c nodo:c:*.* (paginar y encolumnar)

Salida : opcion = SETPAG | SETCOL
retorna OKAY

Condiciones : 1, 4, 6, 7, 10, 12

8- Entrada : pdir /m nodo:c:*.*

Salida : "Error de sintaxis: pdir [/p /c <nodo>:\camino\archivo"
retorna ERROR

Condiciones : 2, 4, 6, 8, 10, 12

4 int VerificarArg(char *directorio, char *nodo, char *fuente)

Entrada: directorio (compuesto por nodo-disco-camino-archivo requerido)

archivo mapa.txt

Salida : nodo (nodo del cual se requiere)

fuente (disco-camino-archivo que se desean ver)

ERROR

OKAY

Condición	Salida Verdadero		Salida Falso	
ptr	1	Tiene nodo fuente.	2	No tiene nodo fuente.
ExisteNodo() == ERROR	3	El nodo no existe en el mapa de conexión.	4	El nodo existe en el mapa de conexión.
ExisteNodo() == -2	5	El mapa de conexión no existe.	6	El mapa de conexión existe.
MapNodeNameToAddress() == ERROR	7	El nodo destino no tiene dirección de red.	8	El nodo destino tiene dirección de red.
networkAddress == dnetwork	9	Los nodos pertenecen a la misma red.	10	Los nodos no pertenecen a la misma red.
networkAddress + LONGNET == dnodo	11	Las direcciones de los nodos son iguales.	12	Las direcciones de los nodos no son iguales.
fnsplit() == ERROR	13	Existe un error en la fuente.	14	No hay error en la fuente.
!(flags & DRIVE)	15	No tiene disco fuente.	16	Tiene disco fuente existe.
!isalpha(drive[0])	17	El disco no comienza con una letra.	18	El disco comienza con una letra.
!(flags & DIRECTORY)	19	No tiene directorio fuente	20	Tiene directorio fuente.
VerifPath() == ERROR	21	El directorio tiene errores de sintaxis.	22	El directorio no tiene errores de sintaxis.
!(flags & FILENAME)	23	No tiene archivo fuente.	24	Tiene archivo fuente.
VerifNbreArch() == ERROR	25	El archivo fuente tiene errores de sintaxis.	26	El archivo fuente no tiene errores de sintaxis.

Casos de Prueba

1- Entrada : nodo1

Salida : "Nodo fuente es obligatorio." (no tiene ':')
retorna ERROR

Condiciones : 2

2-	Entrada	: c:*.* mapa.txt : no tiene 'c' como nodo
	Salida	: "El nodo no existe en el mapa de conexión." (toma como nodo a 'c'). retorna ERROR
	Condiciones	: 1, 3
3-	Entrada	: nodo:c:*.* mapa.txt : no tiene a 'nodo' como nombre de una estación.
	Salida	: "El nodo no existe en el mapa de conexión." (toma como nodo a 'nodo', pero éste no está en mapa.txt). retorna ERROR
	Condiciones	: 1, 3
4-	Entrada	: nodo1:c:*.* mapa.txt : no existe.
	Salida	: "El archivo de mapeo no existe." (no existe mapa.txt). retorna ERROR
	Condiciones	: 1, 5
5-	Entrada	: nodo1:c:*.*
	Salida	: El nodo en el cual se hace el pedido no tiene dirección en la red. retorna ERROR
	Condiciones	: 1, 4, 6, 7
6-	Entrada	: nodo1:c:*.* La dirección de red y nodo de 'nodo1' coincide con la estación destino.
	Salida	: "El nodo debe corresponder a otra estación" retorna ERROR
	Condiciones	: 1, 4, 6, 8, 9, 11

7- Entrada : nodo1:c1:\
Salida : retorna ERROR (existe error en la fuente)
Condiciones : 1, 4, 6, 8, 9, 12, 13

8- Entrada : nodo1:\
Salida : "El disco fuente es obligatorio." retorna ERROR
Condiciones : 1, 4, 6, 8, 9, 12, 14, 15

9- Entrada : nodo1:1:\
Salida : "El disco fuente es incorrecto." retorna ERROR
Condiciones : 1, 4, 6, 8, 9, 12, 14, 16, 17

10- Entrada : nodo1:c:prueba.doc
Salida : "El disco fuente es obligatorio." retorna ERROR
Condiciones : 1, 4, 6, 8, 9 12 14, 16, 18, 19

11- Entrada : nodo1:c:\inco/rrecto\prueba.doc
Salida : "El disco fuente es incorrecto." retorna ERROR
Condiciones : 1, 4, 6, 8, 9, 12, 14, 16, 18, 20, 21

12- Entrada : nodo1:c:\camino\pruebalar.doc
Salida : "El archivo fuente es incorrecto." retorna ERROR
Condiciones : 1, 4, 6, 8, 9, 12, 14, 16, 18, 20, 22, 24, 25

13- Entrada : nodo3:c:\camino\
Pertenecen a distinta red.
Salida : nodo : nodo3
fuente : c:\camino*.*
retorna OKAY
Condiciones : 1, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 23

14- Entrada : nodo2:c:\ms\doc\prueba.doc
Pertenecen a la misma red.

Salida : nodo : nodo2
fuente : c:\ms\doc\prueba.doc
retorna OKAY

Condiciones : 1, 4, 6, 8, 9, 12, 14, 16, 20, 22, 24, 26

5 int PDir(char *nodo, char *fuente, unsigned int opcion)

Entrada: nodo (nodo fuente)

fuente (compuesto por nodo-disco-camino-archivo requerido)

opcion (opción ingresada)

RcvECB (Variable global que apunta a los paquetes que se recibieron)

cant_rcv (Variable global que contiene la cantidad de paquetes recibidos y no fueron analizados)

Salida : Despliega el directorio o archivos del nodo y disco ingresados, con el formato dado por la opción.

ERROR

OKAY

Condición	Salida Verdadero		Salida Falso	
IPXInicialize() == 0x0	1	Las rutinas de IPX no están cargadas.	2	Las rutinas de IPX están cargadas.
SPXInicialize() == 0x0	3	Las rutinas de SPX no están cargadas.	4	Las rutinas de SPX están cargadas.
IPXOpenSocket() != OKAY	5	El socket no puede abrirse.	6	El socket pudo abrirse.
Escuchar() == ERROR	7	No puede poner los paquetes a escuchar.	8	Pone los paquetes a escuchar.
EstablecerConexion()== ERROR	9	No puede establecer conexión.	10	Puede establecer conexión.
control	11	Se presionó Ctrl-Break.	12	No se presionó Ctrl_Break.
MandarMje() == ERROR	13	No se puede enviar el paquete de pedido.	14	Se pudo enviar el paquete de pedido.
!problem y !terminar y !control	15	No hay problemas, no se recibió un paquete para terminar la conexión y no se presionó Ctrl-Break.	16 17 18	Hubo un problema. Se recibió un paquete para terminar conexión. Se presionó Ctrl_Break.
cant_rcv > 0	19	Se recibió al menos un paquete.	20	No se recibieron paquetes.
RcvECB[iRcvECB]-> CompletionCode == 0x00	21	El paquete llegó bien.	22	El paquete llegó mal.
RcvECB[iRcvECB]-> CompletionCode == 0xED	23	LLegó un paquete de "Falla de conexión".	24	No llegó un paquete de "Falla de conexión".
RcvECB[iRcvECB]-> CompletionCode != 0x00 y != 0xED	25	Error tratando de recibir un paquete.	26	Llegó un paquete con 0x00 o 0xED.
!problem y !control	27	No hay problemas y no se presionó Ctrl-Break.	28 29	Hubo algún problema. Se presionó Ctrl-Break.
PoolSPX-> DataStreamType == 0xFE	30	LLegó un paquete del tipo fin de conexión.	31	No llegó un paquete del tipo fin de conexión.
PoolSPX-> DataStreamType == 0x2	32	LLegó un paquete del tipo mensaje de error.	33	No llegó un paquete del tipo mensaje de error.
PoolSPX-> DataStreamType == 0x4	34	LLegó un paquete del tipo dato.	35	No llegó un paquete del tipo dato.

PoolSPX-> DataStreamType != 0xFE y != 0x2 y != 0x4	36	LLegó un paquete del tipo incorrecto.	37	LLegó un paquete de *tipo* conocido.
iRcvECB == VENTANA - 1	38	El paquete leído es el último de la ventana.	39	El paquete leído no es el último de la ventana.
problem	40	Hubo algún problema.	41	No hubo problemas.
control	42	Se presionó Ctrl-Break.	43	No se presionó Ctrl-Break.
!problem	44	No hubo problemas.	45	Hubo algún problema.

Casos de Prueba

1- Entrada : nodo : nodo1
fuente: c:*.*
opcion: 0x00
IPX no está instalado

Salida : "IPX no fue instalado."
retorna ERROR

Condiciones : 1

2- Entrada : nodo = "nodo1"
fuente = "c:*.*"
opcion = 0x00
SPX no está instalado.

Salida : "SPX no fue instalado."
retorna ERROR

Condiciones : 3

3- Entrada : nodo = "nodo1"
fuente = "c:*.*"
opcion = 0x00
El socket ya está abierto por otra aplicación.

Salida : "El socket no fue abierto."
retorna ERROR

Condiciones : 2, 4, 5

4- Entrada : nodo = "nodo1"
fuente = "c:*.*"
opcion = 0x00

Salida : retorna ERROR (No puede poner todos los paquetes a escuchar).

Condiciones : 2, 4, 6, 7

5- Entrada : nodo = "nodo1"
fuente = "c:*.*"
opcion = 0x00

Salida : retorna ERROR (No puede establecer conexión).

Condiciones : 2, 4, 6, 8, 9

6- Entrada : nodo = "nodo1"
fuente = "c:*.*"
opcion = 0x00

Salida : retorna ERROR (Se presionó ^C antes de enviar un paquete de pedido).

Condiciones : 2, 4, 6, 8, 10, 11

7- Entrada : nodo = "nodo1"
fuente = "c:*.*"
opcion = 0x00

Salida : retorna ERROR (No pudo enviar el paquete de pedido).

Condiciones : 2, 4, 6, 8, 10, 12, 13

8- Entrada : nodo = "nodo1"
fuente = "a:\prueba.doc"
opcion = 0x00
La diskettera 'a:' del nodo1 está vacía.

Salida : retorna ERROR (No se pudo abrir el archivo)

Condiciones : 2, 4, 6, 8, 10, 12, 14, 15, 19, 21, 24, 26, 27, 31, 32, 35, 37, 16, 40, 43, 45

9-	Entrada	: nodo = "nodo1" fuente = "c:\prueba.doc" opcion = 0x00 El archivo no existe en el nodo1.
	Salida	: retorna ERROR (No se encontró c:\prueba.doc)
	Condiciones	: 2, 4, 6, 8, 10, 12, 14, 15, 19, 21, 24, 26, 27, 31, 32, 35, 37, 16, 40, 43, 45
10-	Entrada	: nodo = "nodo1" fuente = "c:*.*" opcion = 0x00 Se presiona Ctrl-Break al estar recibiendo paquetes.
	Salida	: retorna ERROR (Se presiona Ctrl-Break)
	Condiciones	: 2, 4, 6, 8, 10, 12, 14, 15, 19, 21, 24, 26, 29, 18, 41, 42
11-	Entrada	: nodo = "nodo1" fuente = "c:*.*" opcion = 0x00 Se recibe un paquete de "Falla de conexión", se desconecta el nodo de la red.
	Salida	: "Falla de conexión" retorna ERROR
	Condiciones	: 2, 4, 6, 8, 10, 12, 14, 15, 19, 22, 23, 28, 16, 40, 43, 45
12-	Entrada	: nodo = "nodo1" fuente = "c:*.*" opcion = 0x00 Se recibe un paquete de error tratando de recibir un paquete.
	Salida	: "Error ocurrido tratando de recibir" retorna ERROR
	Condiciones	: 2, 4, 6, 8, 10, 12, 14, 15, 19, 22, 24, 25, 28, 16, 40, 43, 45

13-	Entrada	: nodo = "nodo1" fuente = "c:*.*" opcion = 0x00 Se recibe un paquete de tipo desconocido.
	Salida	: "El tipo de paquete es incorrecto" retorna ERROR
	Condiciones	: 2, 4, 6, 8, 10, 12, 14, 15, 19, 21, 24, 26, 27, 30, 31, 33, 35, 36, 16, 40, 43, 45
14	Entrada	: nodo = "nodo1" fuente = "c:*.*" opcion = 0x00 No se recibe un paquete por un tiempo mayor a ????.
	Salida	: La conexión es abortada. retorna ERROR
	Condiciones	: 2, 4, 6, 8, 10, 12, 14, 15, 19, 20, 40, 43, 45
15	Entrada	: nodo = "nodo1" fuente = "c:*.*" opcion = 0x00 La conexión es abortada por el otro lado ????.
	Salida	: La conexión es abortada. retorna ERROR
	Condiciones	: 2, 4, 6, 8, 10, 12, 14, 15, 19, 20, 40, 43, 45

16- Entrada : nodo = "nodo1"
fuente = "c:\dos*.*"
opcion = 0x00
Debe llegar más de un paquete, cada paquete tiene 12 archivos.

Salida : Despliega c:\dos*.* de nodo1
retorna OKAY

Condiciones : 2, 4, 6, 8, 10, 12, 14, 15, 19, 21, 24, 26, 27, 34, 39, 19, 21, 24, 26, 27, 34, 38, 19, 21, 24, 26, 27, 30, 33, 35, 37, 39, 17, 39, 41, 43, 44

Otros Casos de Prueba

17- Entrada : nodo = "nodo1"
fuente = "a:*.*"
opcion = 0x00

Salida : Despliega a:*.* del nodo
retorna OKAY

18- Entrada : nodo = "nodo1"
fuente = "c:*.*"
opcion = 0x00
No se hizo el login a Novell.

Salida : Despliega c:*.* del nodo
retorna OKAY

6 void _cdecl CtrlBreak1(void)

Entrada: ninguna

Salida : control = 1

7 void ReceiveESR(ECB *ECB)

Entrada: ECB (Even Control Block)

RcvECB (array de punteros a los paquetes recibidos)

lRcvECB (índice al puntero al siguiente paquete que se recibirá)

cant_rcv (cantidad de paquetes que se recibieron)

Salida: RcvECB

lRcvECB

cant_rcv

Condición	Salida Verdadero		Salida Falso	
lRcvECB == VENTANA - 1	1	Es último paquete de la ventana.	2	No es último paquete de la ventana.

Casos de Prueba

1- Entrada : ECB
RcvECB
lRcvECB = 4
cant_rcv = 5

Salida : RcvECB[4] = ECB
lRcvECB = 0
cant_rcv = 6

Condiciones : 1

2- Entrada : ECB
RcvECB
lRcvECB = 2
cant_rcv = 3

Salida : RcvECB[2] = ECB
lRcvECB = 3
cant_rcv = 4

Condiciones : 2

8 void Mostrar(char *nodo, char *fuente,
struct FIND ffblk[],
long cant, unsigned int opcion)

Entrada: nodo (nodo fuente)

fuente (compuesto por nodo-disco-camino-archivo requerido)

ffblk[] (archivos que se requieron)

cant (cantidad de archivos que hay en ffblk)

opcion (opción ingresada)

Salida : Despliega el directorio o archivos del nodo y disco ingresados, con el formato dado por la opción.

bfiles (cantidad de bytes de los archivos)

nfiles (cantidad de archivos)

linea (cantidad de líneas impresas)

Condición	Salida Verdadero		Salida Falso	
first	1	Es el primer paquete que llega con información.	2	No es el primer paquete que llega con información.
i < cant	3	Hay archivos para mostrar.	4	No hay archivos para mostrar.
opcion & SETPAG	5	Despliga los archivos paginando.	6	Despliga los archivos sin paginar.
opcion & SETCOL	7	Despliga los archivos encolumnando.	8	Despliga los archivos sin encolumnar.
ffblk[i].attribute == FA_DIREC	9	Es un directorio.	10	No es un directorio.
hora == 24	11	hora = 24	12	hora != 24
hora == 12	13	hora = 12	14	hora != 12
12 > hora < 24	15	12 > hora < 24	16	hora < 12
col == MAXCOL	17	Se imprimió sobre la última columna.	18	No se imprimió sobre la última columna.

Casos de Prueba

```
1- Entrada      : nodo    = "nodo1"
                  fuente  = "c:\dos\*.*"
                  ffblk[]
                  cant    = 12
                  opcion  = 0x00
                  Es el primer paquete que se recibe.

   Salida       : Despliega :
                  "Nodo nodo1 c:\dos\*.*"
                  Muestra los archivos que están en
                  ffblk, controlar formato.
                  Verificar bfiles, nfiles.

   Condiciones  : 1, 3, 4, 6, 8, 9, 10, 11, 12, 13, 14,
                  15, 16
```

2- Entrada : nodo = "nodo1"
fuente = "c:\dos*.*"
ffblk[]
cant = 12
opcion = SETPAG
No es el primer paquete que se recibe.

Salida : Despliega los archivos que están en ffblk, controlar formato.
Verificar paginado.
Verificar bfiles, nfiles.

Condiciones : 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 14, 15, 16

3- Entrada : nodo = "nodo1"
fuente = "c:\dos*.*"
ffblk[]
cant = 12
opcion = SETCOL
No es el primer paquete que se recibe.

Salida : Despliega los archivos que están en ffblk, controlar formato.
Verificar paginado.
Verificar bfiles, nfiles.

Condiciones : 2, 3, 4, 6, 7, 10, 11, 12, 13, 14, 15, 16, 17, 18

4- Entrada : nodo = "nodo1"
fuente = "c:\dos*.*"
ffblk[]
cant = 5
opcion = SETCOL
No es el primer paquete que se recibe, es el último.

Salida : Despliega los archivos que están en ffblk, controlar formato.
Verificar encolumnado.
Verificar bfiles, nfiles.

Condiciones : 2, 3, 4, 6, 7, 10, 11, 12, 13, 14, 15, 16, 17, 18

5- Entrada : nodo = "nodo1"
fuente = "c:\dos*.*"
ffblk[]
cant = 12
opcion = SETCOL | SETPAG
No es el primer paquete que se recibe.

Salida : Despliega los archivos que están en ffblk, controlar formato.
Verificar paginado y encolumnado.
Verificar bfiles, nfiles, linea.

Condiciones : 2, 3, 4, 5, 7, 10, 11, 12, 13, 14, 15, 16, 17, 18

9 void Continuar(void)

Entrada: linea

Salida : Despliega el directorio o archivos del nodo y disco ingresados, con el formato dado por la opción.

Condición	Salida Verdadero		Salida Falso	
linea == MAXLINE	1	Se llegó a la cantidad máxima de líneas (no entra en la pantalla).	2	No se llegó a la cantidad máxima de líneas.

Casos de Prueba

1- Entrada : linea = MAXLINEA

Salida : "Presione una tecla para continuar."
linea = 0

Condiciones : 1

2- Entrada : linea < MAXLINEA

Salida : linea++

Condiciones : 2

10 void MostrarFin(unsigned int opcion)

Entrada: opcion (opcion ingresada)

bfiles (cantidad de bytes de los archivos)

nfiles (cantidad de archivos)

Salida : Despliega el directorio o archivos del nodo y disco ingresados, con el formato dado por la opción.

Condición	Salida Verdadero		Salida Falso	
opcion & setpag == SETPAG	1	Paginar.	2	No paginar.

Casos de Prueba

1- Entrada : opcion = SETPAG
Salida : "<nfiles> Archivo(s) <nbytes> bytes"
Verificar paginado.
Condiciones : 1

2- Entrada : opcion = 0x00
Salida : "<nfiles> Archivo(s) <nbytes> bytes"
Condiciones : 2

6.5.2 Modulo pcopia.c

1 int main(int argc, char *argv[])

Entrada: argc (cantidad de argumentos)

argv (argumentos ingresados)

Salida : ERROR

OKAY

Condición	Salida Verdadero		Salida Falso	
argc == 1	1	La cantidad de argumentos es 1 (sólo comando).	2	argc > 1.
argc == 2 y argv[1] == '?'	3	Se ingresó el comando con la opción '?'.	4 5	argc != 2 El segundo argumento no es la opción '?'.
argc != 4	6	La cantidad de argumentos ingresados es inválida.	7	La cantidad de argumentos ingresados es válida.
VerificarArg() == ERROR	8	El pedido es incorrecto	9	El pedido es correcto.
opcion == DEOTRONODO	10	La copia es desde el otro nodo a éste.	11	La copia es desde éste nodo al otro.
CopiarArchivoRx() == ERROR	12	La copia desde el otro nodo a este tuvo problemas.	13	La copia desde el otro nodo a este no tuvo problemas.
CopiarArchivoTx() == ERROR	14	La copia desde éste nodo al otro tuvo problemas.	15	La copia desde éste nodo al otro no tuvo problemas.

Casos de Prueba

1- Entrada : pcopia

Salida : Despliega Help
retorna OKAY

Condiciones : 1

2- Entrada : pcopia ?

Salida : Despliega Help
retorna OKAY

Condiciones : 2, 3

3- Entrada : pcopia ? a nodo:c:\tesis\prueba.doc

Salida : "El nodo fuente es obligatorio."
retorna ERROR

Condiciones : 2, 4, 7, 8

4- Entrada : pcopia nodo:c:\tesis\prueba.doc

Salida : "Error de sintaxis: pcopia <nodo>:<disco>:\<camino>\archivo> A <nodo>:<disco>:\<camino>\archivo>." retorna ERROR

Condiciones : 2, 5, 6

5- Entrada : pcopia nodo1:c:\tesis\prueba.doc A c:.

Salida : retorna ERROR (el nodo destino es obligatorio)

Condiciones : 2, 4, 5, 7, 8

6- Entrada : pcopia nodo2:c:\tesis\prueba.doc A nodo1:c:.
nodo1 es la estación desde la cual se ejecuta el comando.

Salida : retorna ERROR (no puede establecerse la conexión)

Condiciones : 2, 4, 5, 7, 9, 10, 12

7- Entrada : pcopia nodo1:c:\tesis\prueba.doc A nodo2:c:.
nodo1 es la estación desde la cual se ejecuta el comando.

Salida : retorna ERROR (no puede establecerse la conexión)

Condiciones : 2, 4, 5, 7, 9, 11, 14

8- Entrada : pcopia nodo2:c:\tesis\prueba.doc A nodo1:c:.
nodo1 es la estación desde la cual se ejecuta el comando.

Salida : retorna OKAY

Condiciones : 2, 4, 5, 7, 9, 10, 13

9- Entrada : pcopia nodo1:c:\tesis\prueba.doc A nodo2:c:.
nodo1 es la estación desde la cual se ejecuta el comando.

Salida : retorna OKAY

Condiciones : 2, 4, 5, 7, 9, 11, 15

2 void Help(void)

Despliega el help correctamente.

3 int VerificarArg(char **argv, short *opcion, char *nodo1, char *nodo2, char *fuente, char *destino)

Entrada: argv (argumentos ingresados)

Salida : opcion (opción ingresada)

nodo1 (nodo fuente)

nodo2 (nodo destino)

fuente (disco-directorio-archivo fuente)

destino (disco-directorio-archivo destino)

ERROR

OKAY

Condición	Salida Verdadero		Salida Falso	
strcmp(comando,"A")	1	El segundo argumento no es una 'A'.	2	El segundo argumento es una 'A'.
ptr	3	Tiene nodo fuente.	4	No tiene nodo fuente.
MapNodeNameToAddress() == ERROR	5	El nodo fuente no está en mapa.txt.	6	El nodo fuente está en mapa.txt.
networkAddress == dnetwork	7	La dirección de red de la estación y el nodo fuente coinciden.	8	La dirección de red de la estación y el nodo fuente no coinciden.
networkAddress + LONGNET == dnodo	9	La dirección de nodo de la estación y el nodo fuente coinciden.	10	Las dirección de nodo de la estación y el nodo fuente no coinciden.
MapNodeNameToAddress() == ERROR	11	El nodo destino no está en mapa.txt.	12	El nodo destino está en mapa.txt.
networkAddress == dnetwork	13	La dirección de red de la estación y el nodo destino coinciden.	14	La dirección de red de la estación y el nodo destino no coinciden.
networkAddress + LONGNET == dnodo	15	La dirección de nodo de la estación y el nodo destino coinciden.	16	La dirección de nodo de la estación y el nodo destino no coinciden.
*opcion == DEESTENODO	17	El nodo fuente y destino coinciden.	18	El nodo fuente y destino no coinciden.
*opcion == NADA	19	Ni el nodo fuente ni el nodo destino coinciden con la estación.	20 21	El nodo fuente coincide con la estación. El nodo destino coincide con la estación.
fnsplit() == ERROR (fuente)	22	Existe un error en la fuente.	23	No hay error en la fuente.
flags & WILD (fuente)	24	El archivo fuente tiene wildcards.	25	El archivo fuente no tiene wildcards.

opcion == DEOTRONODO && !(flags & DRIVE) (fuente)	26	Copia el archivo fuente desde otro nodo y no tiene drive fuente.	27 28 29	Copia el archivo fuente desde otro nodo y *tiene* drive fuente. Copia el archivo fuente desde este nodo y tiene drive fuente. Copia el archivo fuente desde este nodo y no tiene drive fuente.
flags & DRIVE (fuente)	27 28	Copia el archivo fuente desde otro nodo y tiene drive fuente. Copia el archivo fuente desde este nodo y tiene drive fuente.	29	Copia el archivo fuente desde este nodo y no tiene drive fuente.
!isalpha(drive[0]) (fuente)	30	El disco fuente no comienza con una letra.	31	El disco fuente comienza con una letra.
opcion == DEOTRONODO (fuente)	32	Copia el archivo fuente desde otro nodo.	33	Copia el archivo fuente desde este nodo.
!(flags & DIRECTORY) (fuente)	34	No tiene directorio fuente.	35	Tiene directorio fuente.
VerifPath() == ERROR (fuente)	36	El directorio fuente tiene errores de sintaxis.	37	El directorio fuente no tiene errores de sintaxis.
dir_exists() == ERROR (fuente)	38	El directorio fuente no existe.	39	El directorio fuente existe.
!(flags & FILENAME)	40	No tiene archivo fuente	41	Tiene archivo fuente.
VerifNbreArch() == ERROR (fuente)	42	El archivo fuente tiene errores de sintaxis.	43	El archivo fuente no tiene errores de sintaxis.
opcion == DEESTENODO y !file_exists() (fuente)	44	Copia la fuente desde este nodo y el archivo fuente no existe.	45	Copia la fuente desde este nodo y el archivo fuente existe.
att & FA_HIDDEN (fuente)	46	El archivo fuente es oculto.	47	El archivo fuente no es oculto.
att & FA_SYSTEM (fuente)	48	El archivo fuente es del sistema.	49	El archivo fuente no es del sistema.
fnsplit() == ERROR (destino)	50	Existe un error en el destino.	51	No hay error en el destino.

flagsdest & WILD (destino)	52	El archivo destino tiene wildcards.	53	El archivo destino no tiene wildcards.
opcion == DEESTENODO && !(flagsdest & DRIVE) (destino)	54	Copia el archivo fuente desde este nodo y no tiene drive destino.	55 56 57	Copia el archivo fuente desde este nodo y tiene drive destino. Copia el archivo fuente desde otro nodo y tiene drive destino. Copia el archivo fuente desde otro nodo y no tiene drive destino.
flagsdest & DRIVE (destino)	55 56	Copia el archivo fuente desde este nodo y tiene drive destino. Copia la fuente desde otro nodo y tiene drive destino.	57	Copia el archivo fuente desde otro nodo y no tiene drive destino.
!isalpha(drive[0]) (destino)	58	El disco detino no comienza con una letra.	59	El disco destino comienza con una letra.
opcion == DEESTENODO (destino)	60	Copia la fuente desde este nodo.	61	Copia la fuente desde otro nodo.
!(flagsdest & DIRECTORY) (destino)	62	No tiene directorio destino.	63	Tiene directorio destino.
VerifPath() == ERROR (destino)	64	El directorio destino tiene errores de sintaxis.	65	El directorio destino no tiene errores de sintaxis.
dir_exists() == ERROR (destino)	66	El directorio destino no existe.	67	El directorio destino existe.
!(flagsdest & FILENAME)	68	El nombre del archivo destino no existe.	69	El nombre del archivo destino no existe.
!strcmp(dirdest,".")	70	El directorio destino es "." (directorio corriente).	71	El directorio destino no es "." (no es el directorio corriente).
!strcmp(dirdest,"..")	72	El directorio destino es ".." (directorio padre del corriente).	73	El directorio destino no es ".." (no es el directorio padre del corriente).
VerifNbreArch() == ERROR (destino)	74	El archivo destino tiene errores de sintaxis.	75	El archivo destino no tiene errores de sintaxis.

opcion == DEOTRONODO && file_exists() (destino)	76	Copia el archivo fuente desde otro nodo y el archivo destino ya existe.	77	Copia el archivo fuente desde otro nodo y el archivo destino no existe.
att & FA_HIDDEN (destino)	78	El archivo destino es oculto.	79	El archivo destino no es oculto.
att & FA_SYSTEM (destino)	80	El archivo destino es del sistema.	81	El archivo destino no es del sistema.
att & FA_DIREC (destino)	82	El archivo destino es un directorio.	83	El archivo destino no es un directorio.
att & FA_RDONLY (destino)	84	El archivo destino es un directorio.	85	El archivo destino no es un directorio.
strcmp(rta,"S")	86	La respuesta es una "S".	87	La respuesta no es una "S".
strcmp(rta,"A")	88	La respuesta es una "A".	89	La respuesta no es una "A".

Casos de Prueba

1- Entrada : nodo1:c:\prueba.doc nodo2:c:\tesis\

Salida : "Error de sintaxis: pcopia <nodo>:<disco>:\<camino>\archivo> A <nodo>:<disco>:\<camino>\archivo>." retorna ERROR

Condiciones : 1

2- Entrada : prueba.doc a nodo2:c:\tesis\

Salida : "El nodo fuente es obligatorio." retorna ERROR

Condiciones : 2, 4

3- Entrada : nodo6:c:\prueba.doc a nodo2:c:\tesis\
El nodo6 no está en el mapa.txt.

Salida : retorna ERROR (nodo6 no está en mapa.txt)

Condiciones : 2, 3, 5

4- Entrada : nodo1:c:\prueba.doc a nodo6:c:\tesis\
El nodo6 no está en el mapa.txt.

Salida : retorna ERROR (nodo6 no está en mapa.txt)

Condiciones : 2, 3, 6, 7, 9, 11

5- Entrada : nodo2:c:\prueba.doc a nodo2:c:\tesis\
nodo2 tiene dirección distinta a la estación.

Salida : "Los nodos son incorrectos."
retorna ERROR

Condiciones : 2, 3, 6, 8, 10, 12, 14, 16, 19

6- Entrada : nodo1:c:\prueba.doc a nodo4:c:\tesis\
nodo1 y nodo4 tienen la dirección de la misma estación.

Salida : "El nodo fuente y destino son iguales."
retorna ERROR

Condiciones : 2, 3, 6, 7, 9, 12, 13, 15, 17

7- Entrada : nodo1:c:\pruebitas.doc a nodo2:c:\tesis\
La longitud del archivo fuente es incorrecta.

Salida : retorna ERROR (la longitud del archivo fuente es incorrecta)

Condiciones : 2, 3, 6, 7, 9, 12, 13, 16, 18, 20, 22

8- Entrada : nodo1:c:\prue*.doc a nodo2:c:\tesis\

Salida : "No se puede usar wildcards."
retorna ERROR

Condiciones : 2, 3, 6, 7, 9, 12, 13, 16, 18, 20, 23, 24

9- Entrada : nodo2:\tesis\prueba.doc a nodo1:c:\

Salida : "El disco fuente es obligatorio."
retorna ERROR

Condiciones : 2, 3, 6, 7, 9, 12, 13, 16, 18, 20, 23, 25, 26

10- Entrada : nodo2:1:\tesis\prueba.doc a nodo1:c:\

Salida : "El disco fuente es incorrecto."
retorna ERROR

Condiciones : 2, 3, 6, 7, 9, 12, 13, 16, 18, 20, 23, 25, 27, 30

11- Entrada : nodo1:1:\prueba.doc a nodo2:c:\tesis\

Salida : "El nombre del archivo fuente es incorrecto."
retorna ERROR

Condiciones : 2, 3, 6, 7, 9, 12, 13, 16, 18, 20, 23, 25, 28, 30

12- Entrada : nodo2:c:prueba.doc a nodo1:c:\

Salida : "El directorio fuente es obligatorio."
retorna ERROR

Condiciones : 2, 3, 6, 7, 9, 12, 13, 16, 18, 20, 23, 25, 27, 31, 32, 34

13- Entrada : nodo2:c:\tesis/prueba.doc a nodo1:c:\

Salida : "El directorio fuente es incorrecto."
retorna ERROR

Condiciones : 2, 3, 6, 7, 9, 12, 13, 16, 18, 20, 23, 25, 27, 31, 32, 35, 36

14- Entrada : nodo1:c:/prueba.doc a nodo2:c:\tesis\

Salida : "El directorio fuente es incorrecto."
retorna ERROR

Condiciones : 2, 3, 6, 7, 9, 12, 13, 16, 18, 20, 23, 25, 28, 31, 33, 35, 36

15- Entrada : nodo1:c:\noexiste\prueba.doc a nodo2:c:\tesis\

Salida : "El directorio fuente no existe."
retorna ERROR

Condiciones : 2, 3, 6, 7, 9, 12, 13, 16, 18, 20, 23, 25, 28, 31, 32, 35, 37, 38

16- Entrada : nodo1:c:\ a nodo2:c:\tesis\

Salida : "El archivo fuente es obligatorio."
retorna ERROR

Condiciones : 2, 3, 6, 7, 9, 12, 13, 16, 18, 20, 23, 25, 28, 31, 32, 35, 39, 40

17- Entrada : nodo1:c:\COM a nodo2:c:\tesis\

Salida : "El nombre del archivo fuente es incorrecto."
retorna ERROR

Condiciones : 2, 3, 6, 7, 9, 12, 13, 16, 18, 20, 23, 25, 28, 31, 32, 35, 39, 41, 42

18- Entrada : nodo1:c:\noexiste.doc a nodo2:c:\tesis\

Salida : "El archivo fuente no existe c:\noexiste.doc"
retorna ERROR

Condiciones : 2, 3, 6, 7, 9, 12, 13, 16, 18, 20, 23, 25, 28, 31, 32, 35, 39, 41, 43, 44

19-	Entrada	: nodo1:c:\oculto.doc a nodo2:c:\tesis\ oculto.doc tiene como atributo oculto.
	Salida	: "No puede acceder a c:\oculto.doc" retorna ERROR
	Condiciones	: 2, 3, 6, 7, 9, 12, 13, 16, 18, 20, 23, 25, 28, 31, 32, 35, 39, 41, 43, 45, 46
20-	Entrada	: nodo1:c:\sistema.sys a nodo2:c:\tesis\ sistema.sys tiene como atributo sistema.
	Salida	: "No puede acceder a c:\sistema.sys" retorna ERROR
	Condiciones	: 2, 3, 6, 7, 9, 12, 13, 16, 18, 20, 23, 25, 28, 31, 32, 35, 39, 41, 43, 45, 47, 49
21-	Entrada	: nodo1:c:\io.sys a nodo2:c:\tesis\ io.sys tiene como atributo oculto, sistema, lectura solamente.
	Salida	: "No puede acceder a c:\sistema.sys" retorna ERROR
	Condiciones	: 2, 3, 6, 7, 9, 12, 13, 16, 18, 20, 23, 25, 28, 31, 32, 35, 39, 41, 43, 45, 46, 48
22-	Entrada	: nodo1:c:\prueba.doc a nodo2:c:\tesis\pruebitas.doc La longitud del archivo destino es incorrecta.
	Salida	: retorna ERROR (la longitud del archivo destino es incorrecta)
	Condiciones	: 2, 3, 6, 7, 9, 12, 13, 16, 18, 20, 23, 25, 28, 31, 32, 35, 39, 41, 43, 45, 47, 49, 50

23- Entrada : nodo1:c:\prueba.doc a
nodo2:c:\tesis\prue*ba.doc

Salida : "No se puede usar wildcards."
retorna ERROR

Condiciones : 2, 3, 6, 7, 9, 12, 13, 16, 18, 20, 23, 25, 28, 31, 32, 35, 39, 41, 43, 45, 47, 49, 51, 52

24- Entrada : nodo1:prueba.doc a
nodo2:\tesis\prueba.doc

Salida : "El disco destino es obligatorio."
retorna ERROR

Condiciones : 2, 3, 6, 7, 9, 12, 13, 16, 18, 20, 23, 25, 29, 33, 34, 41, 43, 45, 47, 49, 51, 53, 54

25- Entrada : nodo1:prueba.doc a
nodo2:1:\tesis\prueba.doc

Salida : "El disco destino es incorrecto."
retorna ERROR

Condiciones : 2, 3, 6, 7, 9, 12, 13, 16, 18, 20, 23, 25, 29, 33, 34, 41, 43, 45, 47, 49, 51, 53, 55, 58

26- Entrada : nodo2:c:\tesis\prueba.doc a
nodo1:1:\prueba.c

Salida : "El disco destino es incorrecto."
retorna ERROR

Condiciones : 2, 3, 6, 7, 9, 12, 13, 15, 18, 21, 23, 25, 27, 31, 32, 35, 37, 41, 43, 51, 53, 56, 58

27- Entrada : nodo1:prueba.doc a nodo2:c:prueba.doc

Salida : "El directorio destino es obligatorio." retorna ERROR

Condiciones : 2, 3, 6, 7, 9, 12, 13, 16, 18, 20, 23, 25, 29, 33, 34, 41, 43, 45, 47, 49, 51, 53, 55, 59, 60, 62

28- Entrada : nodo1:prueba.doc a nodo2:c:\tesis/doc\prueba.doc

Salida : "El directorio destino es incorrecto." retorna ERROR

Condiciones : 2, 3, 6, 7, 9, 12, 13, 16, 18, 20, 23, 25, 29, 33, 34, 41, 43, 45, 47, 49, 51, 53, 55, 59, 60, 63, 64

29- Entrada : nodo1:prueba.doc a nodo2:c:.\prueba.doc

Salida : "El directorio destino es incorrecto." retorna ERROR

Condiciones : 2, 3, 6, 7, 9, 12, 13, 16, 18, 20, 23, 25, 29, 33, 34, 41, 43, 45, 47, 49, 51, 53, 55, 59, 60, 63, 64

30- Entrada : nodo2:c:\tesis\prueba.doc a nodo1:c:\noexite\prueba.c

Salida : "El disco destino es incorrecto." retorna ERROR

Condiciones : 2, 3, 6, 7, 9, 12, 13, 15, 18, 21, 23, 25, 27, 31, 32, 35, 37, 41, 43, 51, 53, 56, 59, 61, 63, 65, 66

31- Entrada : nodo1:prueba.doc a
nodo2:c:\tesis\pru.eba.doc

Salida : "El archivo destino es incorrecto."
retorna ERROR

Condiciones : 2, 3, 6, 7, 9, 12, 13, 16, 18, 20, 23, 25, 29, 33, 34, 41, 43, 45, 47, 49, 51, 53, 55, 59, 60, 63, 65, 69, 71, 73, 74

32- Entrada : nodo2:c:\tesis\prueba.doc a
nodo1:c:\pru.eba.doc

Salida : "El archivo destino es incorrecto."
retorna ERROR

Condiciones : 2, 3, 6, 7, 9, 12, 13, 15, 18, 21, 23, 25, 27, 31, 32, 35, 37, 41, 43, 51, 53, 56, 59, 61, 63, 65, 67, 69, 74

33- Entrada : nodo2:c:\tesis\prueba.doc a
nodo1:oculto.doc
oculto.doc tiene como atributo oculto.

Salida : "No puede acceder a c:\oculto.doc"
retorna ERROR

Condiciones : 2, 3, 6, 7, 9, 12, 13, 15, 18, 21, 23, 25, 27, 31, 32, 35, 37, 41, 43, 51, 53, 57, 61, 62, 69, 71, 73, 75, 76, 78

34- Entrada : nodo2:c:\tesis\prueba.doc a
nodo1:.\sistema.sys
sistema.sys tiene como atributo
sistema.

Salida : "No puede acceder a c:\sistema.sys"
retorna ERROR

Condiciones : 2, 3, 6, 7, 9, 12, 13, 15, 18, 21, 23, 25, 27, 31, 32, 35, 37, 41, 43, 51, 53, 57, 61, 63, 65, 67, 69, 71, 73, 75, 76, 79, 80

35- Entrada : nodo2:c:\tesis\prueba.doc a nodo1:doc
doc tiene como atributo directorio.

Salida : "No puede copiar el directorio doc"
retorna ERROR

Condiciones : 2, 3, 6, 7, 9, 12, 13, 15, 18, 21, 23, 25, 27, 31, 32, 35, 37, 41, 43, 51, 53, 57, 61, 63, 65, 67, 69, 71, 73, 75, 76, 79, 81, 82

36- Entrada : nodo2:c:\tesis\prueba.doc a nodo1:read
read tiene como atributo lectura solamente.

Salida : "No puede sobreescribir el archivo read"
retorna ERROR

Condiciones : 2, 3, 6, 7, 9, 12, 13, 15, 18, 21, 23, 25, 27, 31, 32, 35, 37, 41, 43, 51, 53, 57, 61, 63, 65, 67, 69, 71, 73, 75, 76, 79, 81, 83, 84

37- Entrada : nodo2:c:\tesis\prueba.doc a nodo1:existe.doc
No se quiere sobreescribir se ingresa "a".

Salida : "El archivo existe.doc, sobreescribe o aborta? [S/A]"
retorna ERROR

Condiciones : 2, 3, 6, 7, 9, 12, 13, 15, 18, 21, 23, 25, 27, 31, 32, 35, 37, 41, 43, 51, 53, 57, 61, 63, 65, 67, 69, 71, 73, 75, 76, 79, 81, 83, 85, 87, 88

39- Entrada : nodo2:c:\tesis\prueba.doc a nodo1:existe.doc
No se quiere sobreescribir se ingresa "p", la primera vez. La segunda vez se ingresa "A".

Salida : "El archivo existe.doc, sobreescribe o aborta? [S/A]"
Como la respuesta es incorrecta se pregunta nuevamente: "El archivo existe.doc, sobreescribe o aborta? [S/A]"
retorna ERROR

Condiciones : 2, 3, 6, 7, 9, 12, 13, 15, 18, 21, 23, 25, 27, 31, 32, 35, 37, 41, 43, 51, 53, 57, 61, 63, 65, 67, 69, 71, 73, 75, 76, 79, 81, 83, 85, 87, 88

40- Entrada : nodo2:c:\tesis\prueba.doc a nodo1:existe.doc
Se quiere sobreescribir se ingresa "s".

Salida : "El archivo existe.doc, sobreescribe o aborta? [S/A]"
opcion = DEOTRONODO
nodo1 = "nodo2"
nodo2 = "nodo1"
fuente = "c:\tesis\prueba.doc"
destino = "existe.doc"
retorna OKAY

Condiciones : 2, 3, 6, 7, 9, 12, 13, 15, 18, 21, 23, 25, 27, 31, 32, 35, 37, 41, 43, 51, 53, 57, 61, 63, 65, 67, 69, 71, 73, 75, 76, 79, 81, 83, 85, 86, 89

```
41- Entrada     : nodo2:c:\tesis\prueba.doc a
                  nodo1:prueba.doc

    Salida      : opcion  = DEOTRONODO
                  nodo1   = "nodo2"
                  nodo2   = "nodo1"
                  fuente  = "c:\tesis\prueba.doc"
                  destino = "prueba.doc"
                  retorna OKAY

    Condiciones : 2, 3, 6, 7, 9, 12, 13, 15, 18, 21, 23,
                  25, 27, 31, 32, 35, 37, 41, 43, 51, 53,
                  57, 61, 63, 65, 67, 69, 71, 73, 75, 77
```

```
42- Entrada     : nodo2:c:\tesis\prueba.doc a nodo1:.
                  El directorio corriente es \

    Salida      : opcion  = DEOTRONODO
                  nodo1   = "nodo2"
                  nodo2   = "nodo1"
                  fuente  = "c:\tesis\prueba.doc"
                  destino = ".\prueba.doc"
                  retorna OKAY

    Condiciones : 2, 3, 6, 7, 9, 12, 13, 15, 18, 21, 23,
                  25, 27, 31, 32, 35, 37, 41, 43, 51, 53,
                  57, 61, 63, 65, 67, 68, 70, 77
```

```
43- Entrada     : nodo2:c:\tesis\prueba.doc a nodo1:..
                  El directorio corriente es c:\doc\

    Salida      : opcion  = DEOTRONODO
                  nodo1   = "nodo2"
                  nodo2   = "nodo1"
                  fuente  = "c:\tesis\prueba.doc"
                  destino = "..\prueba.doc"
                  retorna OKAY

    Condiciones : 2, 3, 6, 7, 9, 12, 13, 15, 18, 21, 23,
                  25, 27, 31, 32, 35, 37, 41, 43, 51, 53,
                  57, 61, 63, 65, 67, 68, 71, 72, 77
```

```
44- Entrada     : nodo2:c:\tesis\prueba.doc a nodo1:

    Salida      : opcion  = DEOTRONODO
                  nodo1   = "nodo2"
                  nodo2   = "nodo1"
                  fuente  = "c:\tesis\prueba.doc"
                  destino = "prueba.doc"
                  retorna OKAY

    Condiciones : 2, 3, 6, 7, 9, 12, 13, 15, 18, 21, 23,
                  25, 27, 31, 32, 35, 37, 41, 43, 51, 53,
                  57, 61, 63, 65, 66, 68, 71, 73, 77
```

```
45- Entrada     : nodo1:prueba.doc a
                  nodo2:c:\tesis\prueba.doc

    Salida      : opcion  = DEESTENODO
                  nodo1   = "nodo1"
                  nodo2   = "nodo2"
                  fuente  = "prueba.doc"
                  destino = "c:\tesis\prueba.doc"
                  retorna OKAY

    Condiciones : 2, 3, 6, 7, 9, 12, 13, 16, 18, 20, 23,
                  25, 29, 33, 34, 41, 43, 45, 47, 49, 51,
                  53, 55, 59, 60, 63, 65, 69, 71, 73, 75,
                  77
```

```
46- Entrada     : nodo1:prueba.doc a nodo2:c:\tesis\

    Salida      : opcion  = DEESTENODO
                  nodo1   = "nodo1"
                  nodo2   = "nodo2"
                  fuente  = "prueba.doc"
                  destino = "c:\tesis\prueba.doc"
                  retorna OKAY

    Condiciones : 2, 3, 6, 7, 9, 12, 13, 16, 18, 20, 23,
                  25, 29, 33, 34, 41, 43, 45, 47, 49, 51,
                  53, 55, 59, 60, 63, 65, 68, 77
```

```
47- Entrada     : nodo1:.\prueba.doc a nodo2:c:\tesis\

    Salida      : opcion  = DEESTENODO
                  nodo1   = "nodo1"
                  nodo2   = "nodo2"
                  fuente  = ".\prueba.doc"
                  destino = "c:\tesis\prueba.doc"
                  retorna OKAY

    Condiciones : 2, 3, 6, 7, 9, 12, 13, 16, 18, 20, 23,
                  25, 29, 33, 35, 37, 39, 41, 43, 45, 47,
                  49, 51, 53, 55, 59, 60, 63, 65, 68, 77
```

```
48- Entrada     : nodo1:c:\prueba.doc a nodo2:c:\tesis\

    Salida      : opcion  = DEESTENODO
                  nodo1   = "nodo1"
                  nodo2   = "nodo2"
                  fuente  = "c:\prueba.doc"
                  destino = "c:\tesis\prueba.doc"
                  retorna OKAY

    Condiciones : 2, 3, 6, 7, 9, 12, 13, 16, 18, 20, 23,
                  25, 28, 31, 33, 35, 37, 39, 41, 43, 45,
                  47, 49, 51, 53, 55, 59, 60, 63, 65, 68,
                  77
```

```
49- Entrada     : nodo1:a:\prueba.doc a nodo2:a:\

    Salida      : opcion  = DEESTENODO
                  nodo1   = "nodo1"
                  nodo2   = "nodo2"
                  fuente  = "a:\prueba.doc"
                  destino = "a:\prueba.doc"
                  retorna OKAY

    Condiciones : 2, 3, 6, 7, 9, 12, 13, 16, 18, 20, 23,
                  25, 28, 31, 33, 35, 37, 39, 41, 43, 45,
                  47, 49, 51, 53, 55, 59, 60, 63, 65, 68,
                  77
```

50- Entrada : nodo1:c:\read a nodo2:c:\tesis\
read tiene como atributo lectura
solamente.

Salida : opcion = DEESTENODO
nodo1 = "nodo1"
nodo2 = "nodo2"
fuente = "c:\read"
destino = "c:\tesis\read"
retorna OKAY

Condiciones : 2, 3, 6, 7, 9, 12, 13, 16, 18, 20, 23,
25, 28, 31, 33, 35, 37, 39, 41, 43, 45,
47, 49, 51, 53, 55, 59, 60, 63, 65, 68,
77

51- Entrada : nodo1:c:\prueba.doc a nodo20:c:\tesis\
nodo20 pertenece a otro red y tiene la
misma dirección de nodo de la estación.

Salida : opcion = DEESTENODO
nodo1 = "nodo1"
nodo2 = "nodo20"
fuente = "c:\prueba.doc"
destino = "c:\tesis\prueba.doc"
retorna OKAY

Condiciones : 2, 3, 6, 7, 9, 12, 14, 15, 18, 20, 23,
25, 28, 31, 33, 35, 37, 39, 41, 43, 45,
47, 49, 51, 53, 55, 59, 60, 63, 65, 68,
77

52- Entrada : nodo1:c:\prueba.doc a nodo21:c:\tesis\
nodo21 pertenece a otro red y tiene
distinta dirección
de nodo de la estación.

Salida : opcion = DEESTENODO
nodo1 = "nodo1"
nodo2 = "nodo21"
fuente = "c:\prueba.doc"
destino = "c:\tesis\prueba.doc"
retorna OKAY

Condiciones : 2, 3, 6, 7, 9, 12, 14, 16, 18, 20, 23,
25, 28, 31, 33, 35, 37, 39, 41, 43, 45,
47, 49, 51, 53, 55, 59, 60, 63, 65, 68,
77

53- Entrada : nodo20:c:\tesis\prueba.doc a
nodo1:prueba.doc
nodo20 pertenece a otro red y la misma
dirección
de nodo de la estación.

Salida : opcion = DEOTRONODO
nodo1 = "nodo20"
nodo2 = "nodo1"
fuente = "c:\tesis\prueba.doc"
destino = "prueba.doc"
retorna OKAY

Condiciones : 2, 3, 6, 8, 9, 12, 13, 15, 18, 21, 23,
25, 27, 31, 32, 35, 37, 41, 43, 51, 53,
57, 61, 63, 65, 67, 69, 71, 73, 75, 77

54- Entrada : nodo21:c:\tesis\prueba.doc a
nodo1:prueba.doc
nodo21 pertenece a otro red y distinta dirección
de nodo de la estación.

Salida : opcion = DEOTRONODO
nodo1 = "nodo21"
nodo2 = "nodo1"
fuente = "c:\tesis\prueba.doc"
destino = "prueba.doc"
retorna OKAY

Condiciones : 2, 3, 6, 8, 10, 12, 13, 15, 18, 21, 23, 25, 27, 31, 32, 35, 37, 41, 43, 51, 53, 57, 61, 63, 65, 67, 69, 71, 73, 75, 77

4 int CopiarArchivoTx(char *nodo, char *fuente, char *destino)

Entrada: nodo (nodo fuente)

fuente (compuesto por nodo-disco-camino-archivo requerido)

destino (compuesto por nodo-disco-camino-archivo requerido)

RcvECB (Variable global que apunta a los paquetes que se recibieron)

cant_rcv (Variable global que contiene la cantidad de paquetes recibidos y no fueron analizados)

Salida : Copia el archivo fuente en el archivo destino.

ERROR

OKAY

Condición	Salida Verdadero		Salida Falso	
(arch = open()) == ERROR	1	No puede abrir el archivo.	2	Puede abrir el archivo.
larch == 0	3	El archivo está vacío.	4	El archivo no está vacío.
IPXInicialize() == 0x0	5	Las rutinas de IPX no están cargadas.	6	Las rutinas de IPX están cargadas.
SPXInicialize() == 0x0	7	Las rutinas de SPX no están cargadas.	8	Las rutinas de SPX están cargadas.
IPXOpenSocket() != OKAY	9	El socket no puede abrirse.	10	El socket fue pudo abrirse.
Escuchar() == ERROR	11	No puede poner los paquetes a escuchar.	12	Pone los paquetes a escuchar.
EstablecerConexion()== ERROR	13	No puede establecer conexión.	14	Puede establecer conexión.
control	15	Se presionó Ctrl-Break.	16	No se presionó Ctrl-Break.
MandarMje() == ERROR	17	No se puede enviar el paquete de pedido.	18	Se pudo enviar el paquete de pedido.
!problem y !terminar y !control	19	No hay problemas, no se recibió un paquete para terminar la conexión y no se presionó Ctrl-Break.	20 21 22	Hubo un problema. Se recibió un paquete para terminar conexión. Se presionó Ctrl-Break.
cant_rcv > 0	23	Se recibió al menos un paquete.	24	No se recibieron paquetes.
RcvECB[iRcvECB] ->CompletionCode == 0x00	25	El paquete llegó bien.	26	El paquete llegó mal.
RcvECB[iRcvECB] ->CompletionCode == 0xED	27	LLegó un paquete de "Falla de conexión".	28	No llegó un paquete de "Falla de conexión".
RcvECB[iRcvECB] ->CompletionCode != 0x00 y != 0xED	29	Error tratando de recibir un paquete.	30	Llegó un paquete con 0x00 o 0xED.
!problem y !control	31	No hay problemas y no se presionó Ctrl-Break.	32 33	Hubo algún problema. Se presionó Ctrl-Break.
PoolSPX-> DataStreamType == 0xFE	34	LLegó un paquete del tipo fin de conexión.	35	No llegó un paquete del tipo fin de conexión.
PoolSPX-> DataStreamType == 0x2	36	LLegó un paquete del tipo mensaje de error.	37	No llegó un paquete del tipo mensaje de error.

PoolSPX->DataStreamType == 0x4	38	LLegó un paquete del tipo dato.	39	No llegó un paquete del tipo dato.
PoolSPX->DataStreamType != 0xFE y != 0x2 y != 0x4	40	LLegó un paquete del tipo incorrecto.	41	LLegó un paquete de tipo conocido.
!problem && !control	42	No hubo problemas ni se presionó Ctrl-Break.	43 44	No hubo problemas. Se presionó Ctrl-Break.
!first && larch > 0	45	No es el primer paquete que se envía y aún no se terminó de enviar el archivo.	46 44	Es el primer paquete que se envía. Se terminó de enviar el archivo.
problem	45	Hubo algún problema.	46	No hubo problemas.
control	47	Se presionó Ctrl-Break.	48	No se presionó Ctrl-Break.

Casos de Prueba

```
1- Entrada      : nodo     = "nodo2"
                  fuente   = "c:\prueba.txt"
                  destino  = "c:\tesis\prueba.txt"

   Salida       : "No puede abrir c:\prueba.txt."
                  retorna ERROR

   Condiciones  : 1
```

```
2- Entrada      : nodo     = "nodo2"
                  fuente   = "c:\vacio.doc"
                  destino  = "c:\tesis\vacio.doc"

   Salida       : "Archivo vacío, c:\vacio.doc"
                  retorna ERROR

   Condiciones  : 2, 3
```

```
3- Entrada      : nodo     = "nodo2"
                  fuente   = "c:\prueba.doc"
                  destino  = "c:\tesis\prueba.doc"
                  IPX no está instalado.

   Salida       : "IPX no fue instalado."
                  retorna ERROR

   Condiciones  : 2, 4, 5
```

4-	Entrada	: nodo = "nodo2" fuente = "c:\prueba.doc" destino = "c:\tesis\prueba.doc" SPX no está instalado.
	Salida	: "SPX no fue instalado." retorna ERROR
	Condiciones	: 2, 4, 6, 7
5-	Entrada	: nodo = "nodo2" fuente = "c:\prueba.doc" destino = "c:\tesis\prueba.doc" El socket ya está abierto por otra conexión.
	Salida	: "El socket no fue abierto." retorna ERROR
	Condiciones	: 2, 4, 6, 8, 9
6-	Entrada	: nodo = "nodo2" fuente = "c:\prueba.doc" destino = "c:\tesis\prueba.doc"
	Salida	: retorna ERROR (No puede poner todos los paquetes a escuchar).
	Condiciones	: 2, 4, 6, 8, 10, 11
7-	Entrada	: nodo = "nodo2" fuente = "c:\prueba.doc" destino = "c:\tesis\prueba.doc"
	Salida	: retorna ERROR (No puede establecer conexión).
	Condiciones	: 2, 4, 6, 8, 10, 12, 13
8-	Entrada	: nodo = "nodo2" fuente = "c:\prueba.doc" destino = "c:\tesis\prueba.doc"
	Salida	: retorna ERROR (Se presionó Ctrl-Break antes de enviar un paquete de pedido).
	Condiciones	: 2, 4, 6, 8, 10, 12, 14, 15

9-	Entrada	: nodo = "nodo2" fuente = "c:\prueba.doc" destino = "c:\tesis\prueba.doc"
	Salida	: retorna ERROR (No pudo enviar el paquete de pedido).
	Condiciones	: 2, 4, 6, 8, 10, 12, 14, 16, 17
10-	Entrada	: nodo = "nodo2" fuente = "c:\prueba.doc" destino = "c:\tesis\prueba.doc" Se recibe un paquete de "Falla de conexión", se desconecta el nodo en la mitad de la transmisión.
	Salida	: "Falla de conexión" retorna ERROR
	Condiciones	: 2, 4, 6, 8, 10, 12, 14, 16, 18, 19, 23, 25, 30, 31, 35, 37, 38, 39, 41, 42, 19, 23, 26, 27, 32, 20, 48, 51
11-	Entrada	: nodo = "nodo2" fuente = "c:\prueba.doc" destino = "c:\tesis\prueba.doc" Se presiona Ctrl-Break antes de enviar el primer paquete del archivo.
	Salida	: retorna ERROR (Se presiona Ctrl-Break)
	Condiciones	: 2, 4, 6, 8, 10, 12, 14, 16, 18, 19, 23, 25, 30, 33, 22, 49, 50
12-	Entrada	: nodo = "nodo2" fuente = "c:\prueba.doc" destino = "c:\tesis\existe.doc" El archivo no existe en el nodo2, se recibe paquete de error.
	Salida	: Despliega mensaje de error recibido retorna ERROR
	Condiciones	: 2, 4, 6, 8, 10, 12, 14, 16, 18, 19, 23, 25, 28, 30, 31, 36, 20, 41, 48, 51

13- Entrada : nodo = "nodo2"
fuente = "c:\prueba.doc"
destino = "c:\tesis\existe.doc"
Error tratando de recibir un paquete.

Salida : "Error ocurrido tratando de recibir"
retorna ERROR

Condiciones : 2, 4, 6, 8, 10, 12, 14, 16, 18, 19, 23, 26, 28, 29, 20, 48, 51

14- Entrada : nodo = "nodo2"
fuente = "c:\prueba.doc"
destino = "c:\tesis\existe.doc"
Se recibe un paquete de tipo desconocido.

Salida : "El tipo de paquete es incorrecto"
retorna ERROR

Condiciones : 2, 4, 6, 8, 10, 12, 14, 16, 18, 19, 23, 25, 28, 30, 31, 35, 37, 39, 40, 20, 48, 51

15- Entrada : nodo = "nodo2"
fuente = "c:\prueba.doc"
destino = "c:\tesis\prueba.doc"
Se presiona ^C después de enviar el primer paquete del archivo.

Salida : retorna ERROR (Se presiona ^C)

Condiciones : 2, 4, 6, 8, 10, 12, 14, 16, 18, 19, 23, 25, 28, 30, 31, 38, 41, 44, 22, 49, 50

16- Entrada : nodo = "nodo2"
fuente = "c:\prueba.doc"
destino = "c:\tesis\prueba.doc"
No se puede enviar el primer paquete del archivo.

Salida : retorna ERROR (Error leyendo el archivo)

Condiciones : 2, 4, 6, 8, 10, 12, 14, 16, 18, 19, 23, 25, 28, 30, 31, 38, 41, 43, 20, 48, 51

17-	Entrada	: nodo = "nodo2" fuente = "c:\prueba.doc" destino = "c:\tesis\prueba.doc" El archivo tiene más de un paquete, se presiona ^C.
	Salida	: retorna ERROR (Se presiona ^C)
	Condiciones	: 2, 4, 6, 8, 10, 12, 14, 16, 18, 19, 23, 25, 28, 30, 31, 35, 37, 38, 42, 19, 24, 45, 22, 49, 50
18-	Entrada	: nodo = "nodo2" fuente = "c:\prueba.doc" destino = "c:\tesis\prueba.doc" El archivo tiene más de un paquete, se produce un error al tratar de enviar un paquete.
	Salida	: retorna ERROR (Error leyendo el archivo)
	Condiciones	: 2, 4, 6, 8, 10, 12, 14, 16, 18, 19, 23, 25, 28, 30, 31, 38, 41, 42, 19, 24, 20, 48, 51
19-	Entrada	: nodo = "nodo2" fuente = "c:\prueba.doc" destino = "c:\tesis\prueba.doc" El archivo tiene más de un paquete.
	Salida	: retorna OKAY
	Condiciones	: 2, 4, 6, 8, 10, 12, 14, 16, 18, 19, 23, 25, 28, 30, 31, 38, 41, 42, 19, 24, 45, 21, 49, 51
20	Entrada	: nodo = "nodo2" fuente = "c:\prueba.doc" destino = "c:\tesis\prueba.doc" No se recibe un paquete por un tiempo mayor a ????.
	Salida	: La conexión es abortada. retorna ERROR
	Condiciones	: 2, 4, 6, 8, 10, 12, 14, 16, 18, 19, 23, 25, 28, 30, 31, 38, 40, 42, 19, 24, 45, 19, 47, 21, 49, 51

21	Entrada	: nodo : nodo1 fuente: c:*.* opcion: 0x00 La conexión es abortada por el otro lado.
	Salida	: La conexión es abortada. retorna ERROR
	Condiciones	: 2, 4, 6, 8, 10, 12, 14, 15, 19, 24, 46, 19, 24, 46

Otros Casos de Prueba

22-	Entrada	: nodo = "nodo2" fuente = "c:\prueba.doc" destino = "c:\tesis\prueba.doc" El archivo tiene más de un paquete. No se hizo login a Novell.
	Salida	: retorna OKAY
23-	Entrada	: nodo = "nodo2" fuente = "c:\prueba.doc" destino = "a:\tesis\prueba.doc" El archivo tiene más de un paquete. No se hizo login a Novell.
	Salida	: retorna OKAY
24-	Entrada	: nodo = "nodo2" fuente = "a:\prueba.doc" destino = "c:\tesis\prueba.doc" El archivo tiene más de un paquete. No se hizo login a Novell.
	Salida	: retorna OKAY
	Condiciones	: Caja Negra.
25-	Entrada	: nodo = "nodo2" fuente = "c:\dos\mem.exe" destino = "c:\tesis\mem.exe" El archivo tiene más de un paquete y es binario. No se hizo login a Novell.
	Salida	: Controlar cantidad de bytes y ejecución correcta el archivo copiado. retorna OKAY
26-	Entrada	: nodo = "nodo2" fuente = "c:\largo.txt" destino = "c:\tesis\largo.txt" El archivo tiene más de 1MG de longitud.
	Salida	: Controlar cantidad de bytes. retorna OKAY

27- Entrada : nodo = "nodo2"
fuente = "c:\prueba.txt"
destino = "a:\tesis\largo.txt"
La diskettera no tiene puesto el diskette.

Salida : retorna ERROR (no puede abrirse el archivo)

```
5 int EjecutarCopy(char path[], long *larch,
                   BYTE Socket_dest[LONGSOCKET],
                   BYTE Network_dest[LONGNET],
                   BYTE Nodo_dest[LONGNODE],
                   WORD Socket, WORD ConnectionID)
```

Entrada: path (disco-camino-archivo fuente)

larch (bytes del archivo que faltan enviar)

Socket_dest (socket destino)

Network_dest (Dirección de red destino)

Nodo_dest (Dirección de nodo destino)

Socket (socket de la estación local)

ConnectionID (identificador de conexión de la estación local)

Salida : ERROR

OKAY

Condición	Salida Verdadero		Salida Falso	
(numread = read()) == ERROR	1	Error leyendo el archivo.	2	Se pudo leer bien el archivo.

MandarMje() == ERROR	3	Error enviando un paquete del archivo.	4	Se envío bien el paquete del archivo.

Casos de Prueba

1- Entrada : path = "c:\prueba.doc"
Socket_dest = INVITE_SOCKET
Socket = CHAT_SOCKET
El resto de los parametros es dado por la conexión. Se produce error al leer el archivo.

Salida : Envía un paquete con el mensaje de error: "Error leyendo el archivo"
retorna ERROR

Condiciones : 1

2- Entrada : path = "c:\prueba.doc"
Socket_dest = INVITE_SOCKET
Socket = CHAT_SOCKET
El resto de los parametros es dado por la conexión. Se produce error al enviar el paquete.

Salida : retorna ERROR (No puede enviar paquete)

Condiciones : 3

3- Entrada : path = "c:\prueba.doc"
Socket_dest = INVITE_SOCKET
Socket = CHAT_SOCKET
El resto de los parametros es dado por la conexión. Puede enviar el paquete.

Salida : Si es el primer paquete del archivo que envía despliega:
"c:\prueba.doc"
Por cada paquete despliega: "."
retorna OKAY

Condiciones : 2, 4

```
4- Entrada      : nodo    = "nodo2"
                  fuente  = "c:\vacio.doc"
                  destino = "c:\tesis\vacio.doc"
   Salida       : "Archivo vacío, c:\vacio.doc"
                  retorna ERROR
   Condiciones  : 3
```

6 void EscucharPaquete(ECB *RcvECB)

Entrada: RcvECB (Even Control Block)

iRcvECB (índice al puntero al siguiente paquete que se leerá)

cant_rcv (cantidad de paquetes que se recibieron)

Salida: iRcvECB

cant_rcv

Condición	Salida Verdadero		Salida Falso	
iRcvECB == VENTANA - 1	1	Es último paquete de la ventana.	2	No es último paquete de la ventana.

Casos de Prueba

```
1- Entrada      : RcvECB
                  iRcvECB  = 4
                  cant_rcv = 5
   Salida       : RcvECB[4] = ECB
                  iRcvECB   = 0
                  cant_rcv  = 4
   Condiciones  : 1
```

2- Entrada : RcvECB
iRcvECB = 2
cant_rcv = 3

Salida : RcvECB[2] = ECB
iRcvECB = 3
cant_rcv = 2

Condiciones : 2

7 void ReceiveESR(ECB *ECB)

Entrada: ECB (Even Control Block)

RcvECB (array de punteros a los paquetes recibidos)

lRcvECB (índice al puntero al siguiente paquete que se recibirá)

cant_rcv (cantidad de paquetes que se recibieron)

Salida: RcvECB

lRcvECB

cant_rcv

Condición	Salida Verdadero		Salida Falso	
lRcvECB == VENTANA - 1	1	Es último paquete de la ventana.	2	No es último paquete de la ventana.

Casos de Prueba

1- Entrada : ECB
RcvECB
lRcvECB = 4
cant_rcv = 5

Salida : RcvECB[4] = ECB
lRcvECB = 0
cant_rcv = 6

Condiciones : 1

```
2- Entrada      : ECB
                  RcvECB
                  lRcvECB  = 2
                  cant_rcv = 3
   Salida       : RcvECB[2] = ECB
                  lRcvECB   = 3
                  cant_rcv  = 4
   Condiciones : 2
```

6.5.3 Modulo tsrcom.c

1 int main(int argc, char **argv)

Entrada: argc (cantidad de argumentos)

argv (argumentos ingresados)

Salida : ERROR

OKAY

Condición	Salida Verdadero		Salida Falso	
strncmp() == 0	1	argv[1] == "/R" (Intenta remover el programa residente de memoria).	2	argv[1] != "/R".
strncmp() == 0	3	argv[1] == "/r" (Intenta remover el programa residente de memoria).	4	argv[1] != "/r" (Intenta cargar el programa residente en memoria).
i == 0	5	El programa residente fue removido de memoria con éxito.	6	i != 0 (Ocurrió un error al intentar remover el programa residente).
i == 2	7	No puede remover porque el programa no fue cargado en memoria.	8	i != 2 (Ocurrió otro tipo de error).
i == 3	9	No puede remover porque otro programa está cargado sobre el programa residente.	10	i no puede tomar otro valor que no sea algunos de los antes mencionados.
IPXInitialize() == 0x0	11	Las rutinas de IPX no están cargadas.	12	Las rutinas de IPX no están cargadas.
SPXInitialize() == 0x0	13	Las rutinas de SPX no están cargadas.	14	Las rutinas de SPX no están cargadas.
IPXOpenSocket() != OKAY	15	Se produjo un error al abrir el socket, o el socket ya fue abierto y el programa residente cargado en memoria.	16	Abre el socket.
i != 0	17	Ocurrió un error al cargar el programa residente en memoria.	18	i = 0 (El programa fue cargado en memoria quedando residente).

Casos de Prueba

1- Entrada : TSRCOM /R
(Anteriormente el programa residente se cargó en memoria).

Salida : "Program removido"
retorna OKAY.

Condiciones : 1, 4, 5.

2- Entrada : TSRCOM /r
(Anteriormente el programa residente se cargó en memoria).

Salida : "Program removido"
retorna OKAY.

Condiciones : 1, 3, 5.

3- Entrada : TSRCOM /R
(Anteriormente el programa residente no se cargó en memoria).

Salida : "No se puede remover, el programa no ha sido cargado."
retorna ERROR.

Condiciones : 1, 4, 6, 7.

4- Entrada : TSRCOM /r
(Anteriormente el programa residente no se cargó en memoria).

Salida : "No se puede remover, el programa no ha sido cargado."
retorna ERROR.

Condiciones : 2, 3, 6, 7.

5- Entrada : TSRCOM /R
(Anteriormente el programa residente se cargó en memoria, y luego se cargó otro programa residente sobre él).

Salida : "No se puede remover, otro programa cargado sobre él."
retorna ERROR.

Condiciones : 1, 4, 6, 8, 9.

6- Entrada : TSRCOM /r
(Anteriormente el programa residente se cargó en memoria, y luego se cargó otro programa residente sobre él).

Salida : "No se puede remover, otro programa cargado sobre él."
retorna ERROR.

Condiciones : 2, 3, 6, 8, 9.

7- Entrada : TSRCOM
(Las rutinas de IPX no están cargadas).

Salida : "IPX no fue instalado."
"TSRCOM no quedará residente en memoria."
retorna ERROR.

Condiciones : 2, 4, 11.

8- Entrada : TSRCOM
(Las rutinas de SPX no están cargadas).

Salida : "SPX no fue instalado."
"TSRCOM no quedará residente en memoria."
retorna ERROR.

Condiciones : 2, 4, 12, 13.

9- Entrada : TSRCOM
(Anteriormente el programa residente se cargó en memoria).

Salida : "Ocurrió un error al abrir el socket o el programa ya fue cargado en memoria y el socket abierto."
retorna ERROR.

Condiciones : 2, 4, 12, 14, 15.

10- Entrada : TSRCOM
(Ocurre un fallo al instalar el residente).

Salida : "Falla al instalar, error i."
retorna ERROR.

Condiciones : 2, 4, 12, 14, 16, 17.

11- Entrada : TSRCOM

Salida : Instala el programa residente.

Condiciones : 2, 4, 12, 14, 16, 18.

2 void popmain (void)

Entrada: copia (indica si se está ejecutando un comando TXPCOPIA)

cant_rcv (cantidad de paquetes recibidos)

RcvECB (array de punteros a los paquetes que llegan)

iRcvECB (indica el paquete que se tiene que analizar ahora)

Salida : copia (indica si se está ejecutando un comando TXPCOPIA)

cant_rcv (cantidad de paquetes recibidos)

iRcvcECB (indica el paquete que se analizará la próxima vez)

Condición	Salida Verdadero		Salida Falso	
!copia	1	copia = 0 (No está copiando un archivo. Comando TXPCOPIA).	2	copia = 1 (Está copiando un archivo. Comando TXPCOPIA).
ConnectionECB.InUseFlag	3	ConnectionECB.InUseFlag != 0 (Sigue escuchando por una conexión).	4	ConnectionECB.InUseFlag = 0 (ECB disponible).
ConnectionECB. CompletionCode == 0x00	5	Se establece conexión con éxito.	6	Ocurrió un problema al querer establecer una conexión (Socket no abierto o tabla de conexión local llena).
cant_rcv > 0	7	Hay, por lo menos, un paquete para analizar.	8	cant_rcv = 0 (No llegó ningún paquete, todavia).
RcvECB[iRcvECB]-> CompletionCode == 0x00	9	El paquete se recibió con éxito.	10	Ocurrió un problema en la recepción del paquete.
PoolSPX->DataStreamType == 0xFE	11	Es un paquete para terminar la conexión.	12	No es un paquete para terminar la conexión.
copia	13	copia = 1 (Está procesando un comando TXPCOPIA).	14	copia = 0 (No está procesando un comando TXPCOPIA).
PoolSPX-> DataStreamType == 0x1	15	Es un comando.	16	No es un comando.
copia	17	copia = 1 (Está procesando un comando TXPCOPIA).	18	copia = 0 (No está procesando un comando TXPCOPIA).
!strcmp()	19	comando->comando = "PDIR" (Es un comando "PDIR").	20	No es un comando "PDIR".
!strcmp()	21	comando->comando = "RXPCOPIA" (Es un comando "RXPCOPIA").	22	No es un comando "RXPCOPIA".

!strcmp()	23	comando->comando = "TXPCOPIA" (Es un comando "TXPCOPIA").	24	No es un comando "TXPCOPIA".
file_exists()	25	El archivo ya existe.	26	El archivo no existe.
ctrl_disp	27	ctrl_disp = 1 (Ocurrió un error en el dispositivo).	28	ctrl_disp = 0 (No ocurrió error de dispositivo).
(dest = open()) == ERROR	29	Ocurrió un error en la creación y apertura del archivo.	30	Crea y abre el archivo sin problemas.
ctrl_disp	31	ctrl_disp = 1 (Ocurrió un error en el dispositivo).	32	ctrl_disp = 0 (No ocurrió error de dispositivo).
!problema	33	Se envió un paquete con éxito.	34	Ocurrió un problema en el envió del paquete.
PoolSPX->DataStreamType == 0x4	35	Es un paquete de datos.	36	No es un paquete de datos.
(numwr = write ()) != longitud	37	Ocurrió un error en la escritura del archivo.	38	Escribe en el archivo el paquete de datos.
ctrl_disp	39	ctrl_disp = 1 (Ocurrió un error en el dispositivo).	40	ctrl_disp = 0 (No ocurrió error de dispositivo).
PoolSPX->DataStreamType == 0x2	41	Es un paquete de error.	42	No es un paquete de error.
iRcv_ECB == VENT-1	43	Es el fin de la ventana. Hay que comenzar nuevamente desde 0 (ventana circular).	44	No es el fin de la ventana.
problema	45	Ocurrió un problema.	46	No hubo problemas.
copia	47	copia = 1 (Está procesando un comando TXPCOPIA).	48	copia = 0 (No está procesando un comando TXPCOPIA).
cant_rcv > 0	49	Quedaron paquetes sin analizar.	50	cant_rcv = 0 (No quedaron paquetes por analizar).
iRcv_ECB == VENT-1	51	Es el fin de la ventana. Hay que comenzar nuevamente desde 0 (ventana circular).	52	No es el fin de la ventana.
!copia	53	copia = 0 (No está copiando un archivo, Comando TXPCOPIA).	54	copia = 1 (Está copiando un archivo. Comando TXPCOPIA).

Casos de Prueba

1- Entrada : copia = 0
cant_rcv = 0
iRcvECB = 0
RcvECB[iRcvEBC] no apunta a un paquete, todavía no se recibió ninguno.
No se ejecutó ningún comando desde la otra estación.

Salida : copia = 0
cant_rcv = 0
iRcvECB = 0
retorna, escuchando por una conexión.

Condiciones : 1, 3.

2- Entrada : copia = 0
cant_rcv = 0
iRcvECB = 0
RcvECB[iRcvEBC] no apunta a un paquete, todavía no se recibió ninguno.
Desde otra estación se ejecutó el comando:
PDIR nodo:disco:\[directorio\] (es indistinto cual se ejecute, se eligió uno al azar).
Ocurrió un error al querer establecer la conexión (se desconecta la terminal).

Salida : copia = 0
cant_rcv = 0
iRcvECB = 0
retorna escuchando por una conexión

Condiciones : 1, 4, 6.

3- Entrada : copia = 0
cant_rcv = 0
iRcvECB = 0
RcvECB[iRcvEBC] no apunta a un paquete, todavía no se recibió ninguno.
Desde otra estación se ejecutó el comando:
PDIR nodo:disco:\[directorio\] (es indistinto cual se ejecute, se eligió uno al azar).

Salida : Se establece la conexión sin problemas, pero todavía no se recibió el comando.
copia = 0
cant_rcv = 0
iRcvECB = 0

Condiciones : 1, 4, 5, 8.

4- Entrada : copia = 0
cant_rcv = 1
iRcvECB = 0
RcvECB[iRcvEBC] apunta a un paquete.
Desde otra estación se ejecutó el comando:
PDIR nodo:disco:\[directorio\] (es indistinto el comando que se ejecute, se eligió uno al azar).
El paquete se recibe con error, posiblemente por falla en la conexión.

Salida : copia = 0
cant_rcv = 0
iRcvECB = 0
retorna escuchando por una conexión.

Condiciones : 1, 4, 5, 7, 10, 45, 48, 50.

5- Entrada : copia = 0
cant_rcv = 1
iRcvECB = 0
RcvECB[iRcvEBC] apunta a un paquete, que contiene el comando "PDIR".
Desde otra estación se ejecutó el comando:
PDIR nodo:disco:\[directorio\]

Salida : Se produce un error al ejecutar el comando.
copia = 0
cant_rcv = 0
iRcvECB = 0
La otra estación recibe un mensaje de error o detecta falla de conexión.
retorna escuchando por una conexión.

Condiciones : 1, 4, 5, 7, 9, 12, 15, 18, 19, 44, 45, 48, 50.

6- Entrada : copia = 0
cant_rcv = 1
iRcvECB = 0
RcvECB[iRcvEBC] apunta a un paquete, que contiene el comando "PDIR".
Desde otra estación se ejecutó el comando:
PDIR nodo:disco:\[directorio\]

Salida : copia = 0
cant_rcv = 0
iRcvECB = 0
La otra estación recibe los paquetes con los archivos que componen el directorio y los muestra por pantalla.
retorna escuchando por una conexión.

Condiciones : 1, 4, 5, 7, 9, 12, 15, 18, 19, 44, 46, 53.

7- Entrada : copia = 0
cant_rcv = 1
iRcvECB = 0
RcvECB[iRcvEBC] apunta a un paquete, que contiene el comando "RXPCOPIA". Desde otra estación se ejecutó el comando:
PCOPIA otronodo:disco:\[directorio\] otroarchivo A estenodo:disco:\ [direstorio\]estearchivo

Salida : Se produce un problema al ejecutar el comando.
copia = 0
cant_rcv = 0
iRcvECB = 0
La otra estación recibe un mensaje de error o detecta falla de conexión.
retorna escuchando por una conexión.

Condiciones : 1, 4, 5, 7, 9, 12, 15, 18, 20, 21, 44, 45, 48, 53.

8- Entrada : copia = 0
cant_rcv = 1
iRcvECB = 0
RcvECB[iRcvEBC] apunta a un paquete, que contiene el comando "RXPCOPIA". Desde otra estación se ejecutó el comando:
PCOPIA otronodo:disco:\[directorio\] otroarchivo A estenodo:disco:\ [direstorio\]estearchivo

Salida : copia = 0
cant_rcv = 0
iRcvECB = 0
La otra estación recibe paquetes que contienen información del archivo leído y los escribe en un archivo local.
retorna escuchando por una conexión.

Condiciones : 1, 4, 5, 7, 9, 12, 15, 18, 20, 21, 44, 46, 53.

9- Entrada : copia = 0
cant_rcv = 1
iRcvECB = 0
RcvECB[iRcvEBC] apunta a un paquete, que contiene el comando "TXPCOPIA". Desde otra estación se ejecutó el comando:
PCOPIA estenodo:disco:\[directorio\] estearchivo A otronodo:disco:\ [direstorio\]otroarchivo

Salida : El archivo ya existe.
copia = 0
cant_rcv = 0
iRcvECB = 0
La otra estación recibe un mensaje de error ("Error de dispositivo o Archivo ya existe").
retorna escuchando por una conexión.

Condiciones : 1, 4, 5, 7, 9, 12, 15, 18, 20, 22, 23, 25, 28, 44, 45, 48, 50 (No quedaron paquetes por analizar).
1, 4, 5, 7, 9, 12, 15, 18, 20, 22, 23, 25, 28, 44, 45, 49, 51, 52 (Quedaron paquetes sin analizar).

10- Entrada : copia = 0
cant_rcv = 1
iRcvECB = 0
RcvECB[iRcvEBC] apunta a un paquete, que contiene el comando "TXPCOPIA". Desde otra estación se ejecutó el comando:
PCOPIA estenodo:disco:\[directorio\] estearchivo A otronodo:disco:\ [direstorio\]otroarchivo

Salida : Se produce un error de dispositivo, (Por ejemplo: Disquetera vacia).
copia = 0
cant_rcv = 0
iRcvECB = 0
La otra estación recibe un mensaje de error ("Error de dispositivo o Archivo ya existe").
retorna escuchando por una conexión.

Condiciones : 1, 4, 5, 7, 9, 12, 15, 18, 20, 22, 23, 27, 44, 45, 48, 50 (No quedaron paquetes por analizar).
1, 4, 5, 7, 9, 12, 15, 18, 20, 22, 23, 27, 44, 45, 49, 51, 52 (Quedaron paquetes sin analizar).

11- Entrada : copia = 0
cant_rcv = 1
iRcvECB = 0
RcvECB[iRcvEBC] apunta a un paquete, que contiene el comando "TXPCOPIA". Desde otra estación se ejecutó el comando:
PCOPIA estenodo:disco:\[directorio\] estearchivo A otronodo:disco:\ [direstorio\]otroarchivo

Salida : Error al crear el archivo.
copia = 0
cant_rcv = 0
iRcvECB = 0
La otra estación recibe un mensaje de error ("No puede crear el archivo destino").
retorna escuchando por una conexión.

Condiciones : 1, 4, 5, 7, 9, 12, 15, 18, 20, 22, 23, 26, 28, 29, 32, 44, 45, 48, 50 (No quedaron paquetes por analizar).
1, 4, 5, 7, 9, 12, 15, 18, 20, 22, 23, 26, 28, 29, 32, 44, 45, 49, 51, 52 (Quedaron paquetes sin analizar).

12- Entrada : copia = 0
cant_rcv = 1
iRcvECB = 0
RcvECB[iRcvEBC] apunta a un paquete, que contiene el comando "TXPCOPIA". Desde otra estación se ejecutó el comando:
PCOPIA estenodo:disco:\[directorio\] estearchivo A otronodo:disco:\ [direstorio\]otroarchivo

Salida : Se produce un error de dispositivo, (Por ejemplo: Disquetera vacia).
copia = 0
cant_rcv = 0
iRcvECB = 0
La otra estación recibe un mensaje de error ("No puede crear el archivo destino").
retorna escuchando por una conexión.

Condiciones : 1, 4, 5, 7, 9, 12, 15, 18, 20, 22, 23, 26, 28, 31, 44, 45, 48, 50 (No quedaron paquetes por analizar).
1, 4, 5, 7, 9, 12, 15, 18, 20, 22, 23, 26, 28, 31, 44, 45, 49, 51, 52
(Quedaron paquetes sin analizar).

13- Entrada : copia = 0
cant_rcv = 1
iRcvECB = 0
RcvECB[iRcvEBC] apunta a un paquete, que contiene el comando "TXPCOPIA". Desde otra estación se ejecutó el comando:
PCOPIA estenodo:disco:\[directorio\] estearchivo A otronodo:disco:\ [direstorio\]otroarchivo

Salida : Se produce un error al mandar el mensaje que habilita a la otra estación para enviar paquetes (Por ejemplo: se desconecta la estación de la red).
copia = 0
cant_rcv = 0
iRcvECB = 0
retorna escuchando por una conexión.

Condiciones : 1, 4, 5, 7, 9, 12, 15, 18, 20, 22, 23, 26, 28, 30, 32, 34, 44, 45, 48, 50 (No quedaron paquetes por analizar).
1, 4, 5, 7, 9, 12, 15, 18, 20, 22, 23, 26, 28, 30, 32, 34, 44, 45, 49, 51, 52 (Quedaron paquetes sin analizar).

14- Entrada : copia = 0
cant_rcv = 1
iRcvECB = 0
RcvECB[iRcvEBC] apunta a un paquete, que contiene el comando "TXPCOPIA". Desde otra estación se ejecutó el comando:
PCOPIA estenodo:disco:\[directorio\] estearchivo A otronodo:disco:\ [direstorio\]otroarchivo

Salida : copia = 1
cant_rcv > 0
iRcvECB >= 0
retorna escuchando por un paquete de datos.

Condiciones : 1, 4, 5, 7, 9, 12, 15, 18, 20, 22, 23, 26, 28, 30, 32, 33, 44, 46, 54
(Quedaron paquetes por analizar, no termina la conexión).

15- Entrada : copia = 1
cant_rcv > 0
iRcvECB >= 0
RcvECB[iRcvEBC] apunta a un paquete de datos.
En un paso anterior desde la otra estación se ejecutó el comando:
PCOPIA estenodo:disco:\[directorio\] estearchivo A otronodo:disco:\ [direstorio\]otroarchivo
El paquete se recibe con error, posiblemente por falla en la conexión.

Salida : copia = 0
cant_rcv = 0
iRcvECB = 0
retorna escuchando por una conexión.

Condiciones : 2, 7, 10, 45, 47, 49, 51, 52.

16- Entrada : copia = 1
cant_rcv > 0
iRcvECB >= 0
RcvECB[iRcvEBC] apunta a un paquete de datos.
En un paso anterior desde la otra estación se ejecutó el comando:
PCOPIA estenodo:disco:\[directorio\] estearchivo A otronodo:disco:\ [direstorio\]otroarchivo

Salida : Ocurrió un error en la escritura del archivo.
copia = 0
cant_rcv = 0
iRcvECB = 0
La otra estación recibe un mensaje de error ("Error al intentar escribir en el disco").
retorna escuchando por una conexión.

Condiciones : 2, 7, 9, 12, 16, 35, 37, 40, 43 o 44, 45, 47, 50 (No quedaron paquetes por analizar).
2, 7, 9, 12, 16, 35, 37, 40, 43 o 44, 45, 47, 49, 51, 52 (Quedaron paquetes sin analizar).

17- Entrada : copia = 1
cant_rcv > 0
iRcvECB >= 0
RcvECB[iRcvEBC] apunta a un paquete de datos.
En un paso anterior desde la otra estación se ejecutó el comando:
PCOPIA estenodo:disco:\[directorio\] estearchivo A otronodo:disco:\ [direstorio\]otroarchivo

Salida : Se produce un error de dispositivo, (Por ejemplo: Disquetera abierta).
copia = 0
cant_rcv = 0
iRcvECB = 0
La otra estación recibe un mensaje de error ("Error al intentar escribir en el disco").
retorna escuchando por una conexión.

Condiciones : 2, 7, 9, 12, 16, 35, 39, 43 o 44, 45, 47, 50 (No quedaron paquetes por analizar).
2, 7, 9, 12, 16, 35, 39, 43 o 44, 45, 47, 49, 51, 52 (Quedaron paquetes sin analizar).

18- Entrada : copia = 1
cant_rcv > 0
iRcvECB >= 0
RcvECB[iRcvEBC] apunta a un paquete de datos.
En un paso anterior desde la otra estación se ejecutó el comando:
PCOPIA estenodo:disco:\[directorio\] estearchivo A otronodo:disco:\ [direstorio\]otroarchivo

Salida : copia = 1
cant_rcv >= 0
iRcvECB >= 0
retorna escuchando por un paquete de datos.

Condiciones : 2, 7, 9, 12, 16, 35, 38, 40, 43 o 44, 46, 54

19- Entrada : copia = 1
cant_rcv > 0
iRcvECB >= 0
RcvECB[iRcvEBC] apunta a un paquete de error.
En un paso anterior desde la otra estación se ejecutó el comando:
PCOPIA estenodo:disco:\[directorio\] estearchivo A otronodo:disco:\ [direstorio\]otroarchivo

Salida : copia = 0
cant_rcv = 0
iRcvECB = 0
retorna escuchando por una conexión.

Condiciones : 2, 7, 9, 12, 16, 36, 41, 43 o 44, 45, 47, 50 (No quedaron paquetes por analizar).
2, 7, 9, 12, 16, 36, 41, 43 o 44, 45, 49, 51, 52 (Quedaron paquetes por analizar).

20- : copia = 1
Entrada cant_rcv > 0
iRcvECB >= 0
RcvECB[iRcvEBC] apunta a un paquete para terminar la conexión. Este paquete debería llegar sólo si se está ejecutando un comando:
PCOPIA estenodo:disco:\[directorio\] estearchivo A otronodo:disco:\ [direstorio\]otroarchivo

Salida : copia = 0
cant_rcv = 0
iRcvECB = 0
retorna escuchando por una conexión.

Condiciones : 2, 7, 9, 11, 13, 43 o 44, 46, 54.

21- Entrada : copia = 0
cant_rcv = 1
iRcvECB = 0
RcvECB[iRcvEBC] apunta a un paquete de error. Este paquete debería llegar sólo si se está ejecutando un comando:
PCOPIA estenodo:disco:\[directorio\] estearchivo A otronodo:disco:\ [direstorio\]otroarchivo

Salida : copia = 0
cant_rcv = 0
iRcvECB = 0
retorna escuchando por una conexión.

Condiciones : 1, 4, 5, 7, 9, 12, 16, 41, 44, 45, 48, 50.

22- Entrada : copia = 0
cant_rcv = 1
iRcvECB = 0
RcvECB[iRcvEBC] apunta a un paquete para terminar la conexión. Este paquete debería llegar sólo si se está ejecutando un comando:
PCOPIA estenodo:disco:\[directorio\] estearchivo A otronodo:disco:\ [direstorio\]otroarchivo

Salida : copia = 0
cant_rcv = 0
iRcvECB = 0
retorna escuchando por una conexión.

Condiciones : 1, 4, 5, 7, 9, 11, 14, 43 o 44, 46, 53.

23- Entrada : copia = 0
cant_rcv = 1
iRcvECB = 0
RcvECB[iRcvEBC] apunta a un tipo de paquete desconocido. Este no debería llegar nunca.

Salida : copia = 0
cant_rcv = 0
iRcvECB = 0
retorna escuchando por una conexión.

Condiciones : 1, 4, 5, 7, 9, 12, 16, 36, 42, 44, 46, 53.

3 int TerminarConexión (WORD Socket, WORD *ConnectionID)*

Entrada: Socket (número de socket de la estación que desea terminar la conexión)

ConnectionID (número de identificación de conexión)

Salida : OKAY.

Condición	Salida Verdadero		Salida Falso	
TerminateECB.InUseFlag	1	TerminateECB.InUseFlag != 0 (Se está procesando el paquete que termina la conexión)	2	TerminateECB.InUseFlag = 0. (Se terminó la conexión SPX)

Casos de Prueba

1- Entrada : Socket = 0x0250
ConnectionID = N

Salida : retorna OKAY

Condiciones : 1, 2.

4 int EjecutarCopia (char path[MAXPATH],
BYTE Socket_dest[LONGSOCK],
BYTE Network_dest[LONGNET],
BYTE Nodo_dest[LONGNODE])

Entrada: path (path del archivo que se transmitirá por la red)

Socket_dest (número de socket destino)

Network_dest (dirección de red destino)

Nodo_dest (dirección de nodo destino)

Salida : ERROR

OKAY

Condición	Salida Verdadero		Salida Falso	
ctrl_disp	1	Ocurrió un error de dispositivo.	2	No ocurrió error de dispositivo.
att && FA_HIDDEN	3	Es un archivo oculto.	4	No es un archivo oculto.
att && FA_SYSTEM	5	Es un archivo del sistema.	6	No es un archivo del sistema.
ERROR == (arch=open())	7	Error en la apertura del archivo.	8	Abre el archivo.
ctrl_disp	9	Ocurrió un error de dispositivo.	10	No ocurrió error de dispositivo.
larch == 0	11	Archivo vacio.	12	larch > 0. (Archivo no vacio).
!problem && larch > 0	13	No se produjeron errores y no se llegó al fin de archivo.	14	problema = 1 (Ocurrió un error). larch = 0 (se llegó al fin de archivo).
(numread = read()) == ERROR	15	Error de lectura	16	lee del archivo.
ctrl_disp	17	Ocurrió un error de dispositivo.	18	No ocurrió error de dispositivo.
MandarMje() == ERROR	19	Ocurrió un error al mandar un paquete.	20	Envió un paquete a la estación destino.
problem	21	Ocurrió un error.	22	No ocurrieron errores.

Casos de Prueba

1- Entrada : path = a:\[dir\]nom_arch
(disquetera vacia)
Socket_dest = número de socket destino
Network_dest = dirección de red destino
Nodo_dest = dirección de nodo destino
Desde otra estación se ejecutó el comando:
PCOPIA otronodo:a:\[dir\]nom_arch
A estenodo:disco:\[direstorio\]
estearchivo

Salida : retorna ERROR.
La otra estación recibe un mensaje de error ("No puede acceder a a:\nom_arch").

Condiciones : 1, 21.

2- Entrada : path = disco:\[dir\]nom_arch
(archivo oculto)
Socket_dest = número de socket destino
Network_dest = dirección de red destino
Nodo_dest = dirección de nodo destino
Desde otra estación se ejecutó el comando:
PCOPIA otronodo:disco:\[dir\]nom_arch
A estenodo:disco:\[direstorio\]
estearchivo

Salida : retorna ERROR.
La otra estación recibe un mensaje de error ("No puede acceder a disco:\[dir\]nom_arch").

Condiciones : 2, 3, 21.

3- Entrada : path = disco:\[dir\]nom_arch
(archivo del sistema)
Socket_dest = número de socket destino
Network_dest = dirección de red destino
Nodo_dest = dirección de nodo destino
Desde otra estación se ejecutó el comando:
PCOPIA otronodo:disco:\[dir\]nom_arch
A estenodo:disco:\[direstorio\]
estearchivo

Salida : retorna ERROR.
La otra estación recibe un mensaje de error ("No puede acceder a disco:\[dir\]nom_arch").

Condiciones : 2, 4, 5, 21.

4- Entrada : path = disco:\[dir\]nom_arch
(archivo no existe)
Socket_dest = número de socket destino
Network_dest = dirección de red destino
Nodo_dest = dirección de nodo destino
Desde otra estación se ejecutó el comando:
PCOPIA otronodo:disco:\[dir\]nom_arch
A estenodo:disco:\[direstorio\]
estearchivo

Salida : retorna ERROR.
La otra estación recibe un mensaje de error ("No se encontró o no se puede abrir disco:\[dir\]nom_arch").

Condiciones : 2, 4, 6, 7, 21.

5- Entrada : path = disco:\[dir\]nom_arch
Socket_dest = número de socket destino
Network_dest = dirección de red destino
Nodo_dest = dirección de nodo destino
Desde otra estación se ejecutó el comando:
PCOPIA otronodo:disco:\[dir\]nom_arch
A estenodo:disco:\[direstorio\]
estearchivo

Salida : Se produce un error de dispositivo.
retorna ERROR.
La otra estación recibe un mensaje de error ("No se encontró o no se puede abrir disco:\[dir\]nom_arch").

Condiciones : 2, 4, 6, 9, 21.

6- Entrada : path = disco:\[dir\]nom_arch
(archivo vacio)
Socket_dest = número de socket destino
Network_dest = dirección de red destino
Nodo_dest = dirección de nodo destino
Desde otra estación se ejecutó el comando:
PCOPIA otronodo:disco:\[dir\]nom_arch
A estenodo:disco:\[direstorio\]
estearchivo

Salida : retorna ERROR.
La otra estación recibe un mensaje de error ("Archivo vacio
disco:\[dir\]nom_arch").

Condiciones : 2, 4, 6, 8, 10, 11, 21.

7- Entrada : path = disco:\[dir\]nom_arch
Socket_dest = número de socket destino
Network_dest = dirección de red destino
Nodo_dest = dirección de nodo destino
Desde otra estación se ejecutó el comando:
PCOPIA otronodo:disco:\[dir\]nom_arch
A estenodo:disco:\[direstorio\]
estearchivo

Salida : Se produce un error de lectura de disco.
retorna ERROR.
La otra estación recibe un mensaje de error ("Error leyendo el archivo disco:\[dir\]nom_arch").

Condiciones : 2, 4, 6, 8, 10, 12, 13, 15, 14, 21.

8- Entrada : path = disco:\[dir\]nom_arch
Socket_dest = número de socket destino
Network_dest = dirección de red destino
Nodo_dest = dirección de nodo destino
Desde otra estación se ejecutó el comando:
PCOPIA otronodo:disco:\[dir\]nom_arch
A estenodo:disco:\[direstorio\]
estearchivo

Salida : Se produce un error de dispositivo en el momento de leer del disco.
retorna ERROR.
La otra estación recibe un mensaje de error ("Error leyendo el archivo disco:\[dir\]nom_arch").

Condiciones : 2, 4, 6, 8, 10, 12, 13, 17, 14, 21.

9- Entrada : path = disco:\[dir\]nom_arch
Socket_dest = número de socket destino
Network_dest = dirección de red destino
Nodo_dest = dirección de nodo destino
Desde otra estación se ejecutó el comando:
PCOPIA otronodo:disco:\[dir\]nom_arch
A estenodo:disco:\[direstorio\]
estearchivo

Salida : Se produce un error al enviar el paquete, (Por ejemplo: se desconecta la estación destino de la red).
retorna ERROR.
La otra estación detecta falla de conexión.

Condiciones : 2, 4, 6, 8, 10, 12, 13, 16, 18, 19, 14, 21.

10- Entrada : path = disco:\[dir\]nom_arch
Socket_dest = número de socket destino
Network_dest = dirección de red destino
Nodo_dest = dirección de nodo destino
Desde otra estación se ejecutó el comando:
PCOPIA otronodo:disco:\[dir\]nom_arch
A estenodo:disco:\[direstorio\]
estearchivo

Salida : Se envía todo el archivo a la estación que originó el comando.
retorna OKAY.

Condiciones : 2, 4, 6, 8, 10, 12, 13, 16, 18, 20, 14, 22.

5 int MandarMje (BYTE *Socket_dest, BYTE *Network_dest,
BYTE *Nodo_dest, void *dato,
int longitud, BYTE tipo)

Entrada: Socket_dest (número de socket destino)

Network_dest (dirección de red destino)

Nodo_dest (dirección de nodo destino)

dato (dato que se enviará en el paquete)

longitud (longitud del dato)

tipo (tipo de paquete)

Socket origen) (número de socket de la estación

ConnectionID (número de identificación de conexión)

Salida : ERROR

OKAY

Condición	Salida Verdadero		Salida Falso	
SendECB.InUseFlag	1	SendECB.InUseFlag != 0. (El paquete está siendo procesado.)	2	SendECB.InUseFlag = 0. (Se terminó de procesar el paquete).
SendECB.CompletionCode == 0x00	3	El paquete fue enviado con éxito.	4	El paquete no se envió con éxito.

Casos de Prueba

1- Entrada : Socket_dest = número de socket destino
Network_dest = dirección de red destino
Nodo_dest = dirección de nodo destino
dato = dato a enviar
longitud = longitud del dato
tipo = 0x4 o 0x2
Socket = 0x0250
ConnectionID = número de la conexión establecida previamente

Salida : Se produce una falla de conexión (Se desconecta la estación destino). retorna ERROR.

Condiciones : 1, 2, 4.

2- Entrada : Socket_dest = número de socket destino
Network_dest = dirección de red destino
Nodo_dest = dirección de nodo destino
dato = dato a enviar
longitud = longitud del dato
tipo = 0x4 o 0x2
Socket = 0x0250
ConnectionID = número de la conexión establecida previamente

Salida : retorna OKAY.

Condiciones : 1, 2, 3.

```
6 int EjecutarDir (char path[MAXPATH],
                   BYTE Socket_dest[LONGSOCK]
                   BYTE Network_dest[LONGNET],
                   BYTE Nodo_dest[LONGNODE])
```

Entrada: path (path del directorio cuya información se transmitirá por la red, para ser mostrado en la otra estación)

Socket_dest (número de socket destino)

Network_dest (dirección de red destino)

Nodo_dest (dirección de nodo destino)

Salida : ERROR

OKAY

Condición	Salida Verdadero		Salida Falso	
ctrl_disp	1	Ocurrió un error de dispositivo.	2	No ocurrió error de dispositivo.
ffblk == 0	3	No se encontró directorio.	4	Se encontro directorio.
ffblk && !pma	5	Hay archivos en el directorio y no se produjeron errores.	6	ffblk = 0 (No hay más archivos en el directorio). pma != 0 (Ocurrió algún problema).
elem == MAXINFO	7	Se completo un paquete de datos.	8	Todavía hay lugar en el paquete de datos para sumar información.
!pma	9	No se produjeron errores. Sigue colectando información del directorio.	10	Se produjo un error al enviar el paquete. No continua colectando información del directorio.
ctrl_disp	11	Se produjo un error de dispositivo.	12	No se produjo error de dispositivo.
!pma	13	Terminó de colectar información del directorio sin problemas y enviará el último paquete de datos.	14	Se produjo un error.

Casos de Prueba

1- Entrada : path = a:\[dir\] (disquetera vacia)
Socket_dest = número de socket destino
Network_dest = dirección de red destino
Nodo_dest = dirección de nodo destino
Desde otra estación se ejecutó el comando:
PDIR nodo:a:\[dir\]

Salida : retorna ERROR.
La otra estación recibe el siguiente mensaje de error: "Error al intentar leer del disco a:\[dir\]".

Condiciones : 1, 6, 14.

2- Entrada : path = disco:\[dir\] (no existe el directorio)
Socket_dest = número de socket destino
Network_dest = dirección de red destino
Nodo_dest = dirección de nodo destino
Desde otra estación se ejecutó el comando:
PDIR nodo:disco:\[dir\]

Salida : retorna ERROR.
La otra estación recibe un mensaje de error ("No se encontró disco:\[dir\]").

Condiciones : 2, 3, 6, 14.

3- Entrada : path = disco:\[dir\]
Socket_dest = número de socket destino
Network_dest = dirección de red destino
Nodo_dest = dirección de nodo destino
Desde otra estación se ejecutó el comando:
PDIR nodo:disco:\[dir\]

Salida : Se produce una falla de conexión al mandar el primer paquete con la información del directorio.
retorna ERROR.

Condiciones : 2, 4, 5, 8, 7, 10, 6, 14.

4- Entrada : path = disco:\[dir\]
Socket_dest = número de socket destino
Network_dest = dirección de red destino
Nodo_dest = dirección de nodo destino
Desde otra estación se ejecutó el comando:
PDIR nodo:disco:\[dir\]

Salida : Se produce un error de dispositivo cuando está colectando información del directorio.
retorna ERROR.
La otra estación recibe un mensaje de error ("Error al intentar leer del disco a:\[dir\]").

Condiciones : 2, 4, 5, 9, 11, 6, 14.

5- Entrada : path = disco:\[dir\]
Socket_dest = número de socket destino
Network_dest = dirección de red destino
Nodo_dest = dirección de nodo destino
Desde otra estación se ejecutó el comando:
PDIR nodo:disco:\[dir\]

Salida : Envía toda la información del directorio a la otra estación.
retorna OKAY.

Condiciones : 2, 4, 5, 6, 7, 8, 9, 12, 13.

6- Entrada : path = disco:\[dir\]
Socket_dest = número de socket destino
Network_dest = dirección de red destino
Nodo_dest = dirección de nodo destino
Desde otra estación se ejecutó el comando:
PDIR nodo:disco:\[dir\]

Salida : El directorio solicitado está vacio.
retorna OKAY.

Condiciones : 2, 4, 6, 13.

7 void ReceiveESR (ECB *ESR_ECB)

Entrada: ESR_ECB (puntero a un ECB)

RcvECB (array de punteros a los paquetes que llegan)

lRcvECB (indica el lugar que ocupará, en el array antes mencionado, el paquete que acaba de llegar)

Salida : RcvECB (array de punteros a los paquetes que llegan)

lRcvECB (indica el lugar que ocupará, en el array antes mencionado, próximo paquete que llegue)

Condición	Salida Verdadero		Salida Falso	
lRcvECB == VENT - 1	1	Llegó al final de la ventana circular.	2	No llegó al final de la ventana.

Casos de Prueba

1- Entrada : ESR_ECB = puntero al ECB que acaba de llegar.
RcvECB = contiene los punteros a los ECB que todavía no se analizaron.
lRcvECB < VENT -1.

Salida : RcvECB = contiene el puntero al ECB que recien llegó, más los que ya tenía.
lRcvECB = lRcvECB++.

Condiciones : 2.

```
2- Entrada      : ESR_ECB = puntero al ECB que acaba de
                  llegar.
                  RcvECB  = contiene los punteros a los
                  ECB que todavía no
                            se analizaron.
                  lRcvECB = VENT -1.

   Salida       : RcvECB  = contiene el puntero al ECB
                  que recien llegó.
                  lRcvECB = 0.

   Condiciones : 1.
```

6.5.4 Modulo conexion.c

1 extern int TerminarConexión (WORD Socket,
WORD ConnectionID)

Entrada: Socket (número de socket de la estación que desea terminar la conexión)

ConnectionID (número de identificación de conexión)

Salida : ERROR

OKAY

Condición	Salida Verdadero		Salida Falso	
(TerminateECB = (ECB *) calloc()) == NULL	1	Memoria saturada, tratando de ubicar un ECB.	2	Ubica memoria para un ECB de terminación.
(TerminateSPX = (SPXHeader *) calloc()) == NULL	3	Memoria saturada, tratando de ubicar un header SPX.	4	Ubica memoria para un header SPX de terminación.
TerminateECB.InUseFlag	5	TerminateECB.InUseFlag != 0 (Se está procesando el paquete que termina la conexión)	6	TerminateECB.InUseFlag = 0. (Se terminó la conexión SPX)

Casos de Prueba

1- Entrada : Socket = NS
ConnectionID = N

Salida : No hay memoria para un ECB.
Muestra por pantalla lo siguiente:
"Memoria saturada, tratando de alocar para ECB de terminación."
retorna ERROR.

Condiciones : 1.

2- Entrada : Socket = NS
ConnectionID = N

Salida : No hay memoria para un header SPX.
Muestra por pantalla lo siguiente:
"Memoria saturada, tratando de alocar para SPX de terminación."
retorna ERROR.

Condiciones : 2, 3.

3- Entrada : Socket = NS
ConnectionID = N

Salida : retorna OKAY.

Condiciones : 2, 4, 5, 6.

2 extern int Escuchar (short ventana, WORD Socket, void (*ESR)())

Entrada: ventana (número de elementos de la ventana circular)

Socket (número de socket)

ESR (dirección de la rutina de servicio de eventos)

Salida : ERROR

OKAY

Condición	Salida Verdadero		Salida Falso	
(PoolECB = (ECB *) calloc()) == NULL	1	Memoria saturada, tratando de ubicar un ECB.	2	Ubica memoria para un ECB.
(PoolSPX = (SPXHeader *) calloc()) == NULL	3	Memoria saturada, tratando de ubicar un header SPX.	4	Ubica memoria para un header SPX.
(data = (char *) calloc()) == NULL	5	Memoria saturada, tratando de ubicar datos.	6	Ubica memoria para datos.

Casos de Prueba

1- Entrada : ventana = 5
Socket = NS
ESR = dirección de rutina de servicio de eventos.

Salida : No hay memoria para un ECB.
Muestra por pantalla lo siguiente:
"Memoria saturada, tratando de alocar para ECB de escucha."
retorna ERROR.

Condiciones : 1.

2- Entrada : ventana = 5
Socket = NS
ESR = dirección de rutina de servicio de eventos.

Salida : No hay memoria para un header SPX.
Muestra por pantalla lo siguiente:
"Memoria saturada, tratando de alocar para SPX de escucha."
retorna ERROR.

Condiciones : 2, 3.

3- Entrada : ventana = 5
Socket = NS
ESR = dirección de rutina de servicio de eventos.

Salida : No hay memoria para los datos.
Muestra por pantalla lo siguiente:
"Memoria saturada, tratando de alocar para Datos."
retorna ERROR.

Condiciones : 2, 4, 5.

4- Entrada : ventana = 5
Socket = NS
ESR = dirección de rutina de servicio de eventos.

Salida : Coloca los ECBs a escuchar
retorna OKAY.

Condiciones : 2, 4, 6.

```
3 extern int EstablecerConexión (char *NodoName,
                                 WORD Socket,
                                 BYTE *dnetwork,
                                 BYTE *dnodo,
                                 WORD dsocket,
                                 WORD *ConnectionID)
```

Entrada: NodoName (nombre de la estación con la que se establecerá conexión)

Socket (número de socket de la estación que establece la conexión)

dsocket (número de socket de la estación con la que se establecerá la conexión)

Salida : dnodo (dirección de nodo de la estación con la que se establecerá la conexión)

dnetwork (dirección de red de la estación con la que se establecerá la conexión)

ConnectionID (número de identificación de conexión)

ERROR

OKAY

Condición	Salida Verdadero		Salida Falso	
(ConnectionECB = (ECB *) calloc()) == NULL	1	Memoria saturada, tratando de ubicar un ECB.	2	Ubica memoria para un ECB de conexión.
(ConnectionECB = (SPXHeader *) calloc()) == NULL	3	Memoria saturada, tratando de ubicar un header SPX.	4	Ubica memoria para un header SPX.
MapNodeNameToAddress() == ERROR	5	Se produjo un error al tratar de obtener la dirección de red y nodo, de la estación con la que se desea establecer conexión.	6	Se obtiene la direción de red y nodo de la estación con la que se desea establecer la conexión.
ConectionECB-> CompletionCode == 0x00	7	Se estableció la conexión.	8	No se estableció conexión.
ConectionECB-> CompletionCode == 0xED	9	No se estableció la conexión. No se recibió respuesta desde la otra estación.	10	Error durante el establecimiento de la conexión.

Casos de Prueba

1- Entrada : NodoName = NODOX
Socket = 0x0150
dnetwork = rrrrrrrr
nodo = nnnnnnnnnnnn
dsocket = 0x0250
El comando a procesar es, por ejemplo:
PDIR nodo:disco:\[dir\]

Salida : No hay memoria para un ECB.
Muestra por pantalla lo siguiente:
"Memoria saturada, tratando de alocar para ECB de Conexión."
ConnectionID = NADA (no se estableció conexión).
retorna ERROR.

Condiciones : 1.

2- Entrada : NodoName = NODOX
Socket = 0x0150
dnetwork = rrrrrrrr
nodo = nnnnnnnnnnnn
dsocket = 0x0250
El comando a procesar es, por ejemplo:
PDIR nodo:disco:\[dir\]

Salida : No hay memoria para un header SPX.
Muestra por pantalla lo siguiente:
"Memoria saturada, tratando de alocar para SPX de Conexión."
ConnectionID = NADA (no se estableció conexión).
retorna ERROR.

Condiciones : 2, 3.

3- Entrada : NodoName = NODOX (no existe)
Socket = 0x0150
dnetwork = rrrrrrrr
nodo = nnnnnnnnnnnn
dsocket = 0x0250
El comando a procesar es, por ejemplo:
PDIR nodo:disco:\[dir\]

Salida : Se produce un error al tratar de obtener la dirección de red y nodo destino.
Por ejemplo, el nodo ingresado no existe en el archivo de mapeo.
ConnectionID = NADA (no se estableció conexión).
retorna ERROR.

Condiciones : 2, 4, 5.

4- Entrada : NodoName = NODOX
Socket = 0x0150
dnetwork = rrrrrrrr
nodo = nnnnnnnnnnnn
dsocket = 0x0250
El comando a procesar es, por ejemplo:
PDIR nodo:disco:\[dir\]

Salida : Se produce un error al tratar de establecer la conexión. Por ejemplo, de desconecta la otra estación de la red.
Muestra por pantalla lo siguiente:
"No se estableció la conexión, no se recibió respuesta."
ConnectionID = NADA (no se estableció conexión).
retorna ERROR.

Condiciones : 2, 4, 6, 8, 9.

5- Entrada : NodoName = NODOX
Socket = 0x0150
dnetwork = rrrrrrrr
nodo = nnnnnnnnnnnn
dsocket = 0x0250
El comando a procesar es, por ejemplo:
PDIR nodo:disco:\[dir\]

Salida : Se produce un error al tratar de establecer la conexión. Por ejemplo, la tabla de conexión SPX está llena.
Muestra por pantalla lo siguiente:
"Error durante el establecimiento de la conexión."
ConnectionID = NADA (no se estableció conexión).
retorna ERROR.

Condiciones : 2, 4, 6, 8, 10.

6- Entrada : NodoName = NODOX
Socket = 0x0150
dnetwork = rrrrrrrr
nodo = nnnnnnnnnnnn
dsocket = 0x0250
El comando a procesar es, por ejemplo:
PDIR nodo:disco:\[dir\]

Salida : Se establece la conexión
ConnectionID = N
retorna OKAY.

Condiciones : 2, 4, 6, 7.

4 int MapNodeToAddress (char *NodoName, BYTE *dir_red, BYTE *dir_nodo)

Entrada: NodoName (nombre de la estación de la cual se obtendrá la dirección de red y la dirección de nodo)

Salida : dir_red (dirección de red de la estación)

dir_nodo (dirección de nodo de la estación)

ERROR

OKAY

Condición	Salida Verdadero		Salida Falso	
(mapa = fopen()) == NULL	1	El archivo de mapeo no existe.	2	Abre el archivo de mapeo.
!encontro && (ch = fgetc()) != EOF	3	No se encontró y el caracter leído no es el fin de archivo.	4	a. Se encontró. b. Se terminó el archivo.
ch = ' '	5	Es un blanco (separador). Puede haber obtenido un nombre de nodo.	6	No es un blanco.
!nbre	7	Analiza el nombre de nodo que acaba de obtener.	8	Lo que viene ahora no es un nombre de nodo.
!strcmp()	9	Encontró el nombre de la estación.	10	No es el nombre de la estación.
(ch = fgetc()) != EOF && ch != '\n' && ch != ' '	11	El caracter leído no es el fin de archivo, es distinto de fin de línea y es distinto de blanco.	12	El caracter leído es fin de archivo. El caracter leído es un fin de línea. El caracter leído es un blanco.
i > LONGNET*2	13	La dirección de red es incorrecta.	14	La longitud de la dirección de red es correcta.
StrToHexa() == ERROR	15	La dirección de red es incorrecta.	16	La dirección de red es correcta.
(ch = fgetc()) != EOF && ch != '\n' && ch != ' '	17	El caracter leído no es el fin de archivo, es distinto de fin de línea y es distinto de blanco.	18	El caracter leído es fin de archivo. El caracter leIdo es un fin de línea. El caracter leído es un blanco.
i > LONGNODE*2	19	La dirección de nodo es incorrecta.	20	La longitud de la dirección de nodo es correcta.
StrToHexa() == ERROR	21	La dirección de nodo es incorrecta.	22	La dirección de nodo es correcta.
ch == '\n'	23	Es fin de línea.	24	No es fin de línea.
!nbre	25	No encontró, todavía, un nombre.	26	Ya tiene un nombre.

!encontró	27	No encontró el nombre de la estación en el archivo de mapeo.	28	Encontró el nombre en el archivo de mapeo.

Casos de Prueba

1- Entrada : NodoName = NODOX

Salida : El archivo de mapeo no esta cargado en la estación que origina el comando.
Muestra por pantalla lo siguiente:
"El archivo mapa.txt no existe."
dir_red = NADA.
dir_nodo = NADA.
retorna ERROR.

Condiciones : 1.

2- Entrada : NodoName = NODOX (no existe)

Salida : El nodo no existe en el archivo de mapeo.
Muestra por pantalla lo siguiente:
"El nodo NODOX no existe."
dir_red = NADA.
dir_nodo = NADA.
retorna ERROR.

Condiciones : 2, 3, 6, 24, 25, 5, 7, 10, 26, 23, 4b, 27.

3- Entrada : NodoName = NODOX

Salida : Archivo de mapeo vacio.
Muestra por pantalla lo siguiente:
"El nodo NODOX no existe."
dir_red = NADA.
dir_nodo = NADA.
retorna ERROR.

Condiciones : 2, 4b, 27.

4- Entrada : NodoName = NODOX

Salida : Archivo de mapeo vacio.
Muestra por pantalla lo siquiente:
"El nodo NODOX no existe."
dir_red = NADA.
dir_nodo = NADA.
retorna ERROR.

Condiciones :

5- Entrada : NodoName = NODOX

Salida : La longitud de dirección de red encontrada, es mayor de cuatro bytes.
Muestra por pantalla lo siquiente:
"La dirección de red es incorrecta rrrrrrrrr."
dir_red = NADA.
dir_nodo = NADA.
retorna ERROR.

Condiciones : 2, 3, 6, 24, 25, 5, 7, 9, 11, 12, 13.

6- Entrada : NodoName = NODOX

Salida : No puede convertir la dirección de red a un número hexadecimal (posee un caracter que no es un dígito hexadecimal).
Muestra por pantalla lo siquiente:
"La dirección de red es incorrecta rrrrrrrr."
dir_red = NADA.
dir_nodo = NADA.
retorna ERROR.

Condiciones : 2, 3, 6, 24, 25, 5, 7, 9, 11, 12, 14, 16.

7- Entrada : NodoName = NODOX

Salida : La longitud de dirección de nodo encontrada, es mayor de seis bytes.
Muestra por pantalla lo siquiente:
"La dirección de red es incorrecta nnnnnnnnnnnn."
dir_red = NADA.
dir_nodo = NADA.
retorna ERROR.

Condiciones : 2, 3, 6, 24, 25, 5, 7, 9, 11, 12, 14, 16, 17, 18, 19.

8- Entrada : NodoName = NODOX

Salida : No puede convertir la dirección de nodo a un número hexadecimal (posee un caracter que no es un dígito hexadecimal).
Muestra por pantalla lo siquiente:
"La dirección de red es incorrecta nnnnnnnnnnnn."
dir_red = NADA.
dir_nodo = NADA.
retorna ERROR.

Condiciones : 2, 3, 6, 24, 25, 5, 7, 9, 11, 12, 14, 16, 17, 18, 20, 21.

9- Entrada : NodoName = NODOX

Salida : Encuentra la dirección de red y nodo de la estación.
dir_red = rrrrrrrr.
dir_nodo = nnnnnnnnnnnn.
retorna OKAY.

Condiciones : 2, 3, 6, 24, 25, 5, 7, 10, 26, 8, 23, 9, 11, 12, 14, 16, 17, 18, 20, 22, 4a, 28.

5 extern int StrToHexa (char *origen, BYTE *destino)

Entrada: origen (cadena de caracteres)

Salida : destino (número hexadecimal)

ERROR

OKAY

Condición	Salida Verdadero		Salida Falso	
!isxdigit()	1	No es un dígito hexadecimal.	2	Es un dígito hexadecimal.
!isalpha()	3	Es una letra.	4	No es una letra.
!(i % 2)	5	Convierte a hexadecimal el caracter que ocupa un lugar impar.	6	Convierte a hexadecimal el caracter que ocupa un lugar par.

Casos de Prueba

1- Entrada : origen = 'cc...cc'

Salida : La cadena posee un caracter que no es un dígito hexadecimal.
destino = NADA.
retorna ERROR.

Condiciones : 1.

2- Entrada : origen = 'cc...cc'

Salida : La cadena no posee letras.
destino = hh...hh
retorna OKAY.

Condiciones : 2, 4, 5, 6.

3- Entrada : origen = 'cc...cc'

Salida : La cadena posse letras y dígitos.
destino = hh...hh
retorna OKAY.

Condiciones : 2, 3, 4, 5, 6.

6 int MandarMje (BYTE *dnetwork, BYTE *dnodo,
BYTE *dsocket, WORD Socket,
WORD ConnectionID, void *dato,
int longitud, BYTE tipo)

Entrada: detwork (dirección de red destino)

dnodo (dirección de nodo destino)

dsocket (número de socket destino)

Socket (número de socket de la estación origen)

ConnectionID (número de identificación de conexión)

dato (dato que se enviará en el paquete)

longitud (longitud del dato)

tipo (tipo de paquete)

Salida : ERROR

OKAY

Condición	Salida Verdadero		Salida Falso	
(SendECB = (ECB *) calloc()) == NULL	1	Memoria saturada, tratando de ubicar un ECB.	2	Ubica memoria para un ECB.
(SendSPX = (SPXHeader *) calloc()) == NULL	3	Memoria saturada, tratando de ubicar un header SPX.	4	Ubica memoria para un header SPX.
SendECB.InUseFlag	5	SendECB.InUseFlag != 0. (El paquete está siendo procesado.)	6	SendECB.InUseFlag = 0. (Se terminó de procesar el paquete).
SendECB.CompletionCode == 0x00	7	El paquete fue enviado con éxito.	8	El paquete no se envió con éxito.
SendECB.CompletionCode == 0xED	9	Se produjo una falla de conexión.	10	No se produjo falla de conexión.
SendECB.CompletionCode == 0xEC	11	La conexión fue terminada por la otra estación.	12	Se produjo un error tratando de enviar un paquete.

Casos de Prueba

1- Entrada : dnetwork = dirección de red destino
dnodo = dirección de nodo destino
dsocket = número de socket destino
Socket = 0x0150
ConnectionID = número de conexión establecida previamente
dato = dato a enviar
longitud = longitud del dato
tipo = 0x4 o 0x2

Salida : No hay memoria para un ECB.
Muestra por pantalla lo siguiente:
"Memoria saturada, tratando de alocar para ECB de Paquete."
retorna ERROR.

Condiciones : 1.

2- Entrada : dnetwork = dirección de red destino
dnodo = dirección de nodo destino
dsocket = número de socket destino
Socket = 0x0150
ConnectionID = número de conexión establecida previamente
dato = dato a enviar
longitud = longitud del dato
tipo = 0x4 o 0x2

Salida : No hay memoria para un header SPX. Muestra por pantalla lo siguiente: "Memoria saturada, tratando de alocar para SPX de Paquete." retorna ERROR.

Condiciones : 2, 3.

3- Entrada : dnetwork = dirección de red destino
dnodo = dirección de nodo destino
dsocket = número de socket destino
Socket = 0x0150
ConnectionID = número de conexión establecida previamente
dato = dato a enviar
longitud = longitud del dato
tipo = 0x4 o 0x2

Salida : retorna ERROR.

Condiciones : Se produce una falla de conexión (Se desconecta la estación origen de la red).
2, 4, 5, 6, 8, 9.

4- Entrada : dnetwork = dirección de red destino
dnodo = dirección de nodo destino
dsocket = número de socket destino
Socket = 0x0150
ConnectionID = número de conexión establecida previamente
dato = dato a enviar
longitud = longitud del dato
tipo = 0x4 o 0x2

Salida : La conexión es terminada por el otro nodo (Se desconecta la estación destino de la red).
retorna ERROR.

Condiciones : 2, 4, 5, 6, 8, 10, 11.

5- Entrada : dnetwork = dirección de red destino
dnodo = dirección de nodo destino
dsocket = número de socket destino
Socket = 0x0150
ConnectionID = no es el número de conexión establecida previamente.
dato = dato a enviar
longitud = longitud del dato
tipo = 0x4 o 0x2

Salida : Se produce otro tipo de error (Se pasa mal el número de conexión).
retorna ERROR.

Condiciones : 2, 4, 5, 6, 8, 10, 12.

6- Entrada	: dnetwork	= dirección de red destino
	dnodo destino	= dirección de nodo
	dsocket	= número de socket destino
	Socket	= 0x0150
	ConnectionID establecida previamente	= número de conexión
	dato	= dato a enviar
	longitud	= longitud del dato
	tipo	= 0x4 o 0x2
Salida	: El paquete es enviado con éxito. retorna OKAY.	
Condiciones	: 2, 4, 5, 6, 7.	

6.5.5 Modulo dirarch.c

1 int file_exists(char *filename)

Entrada: filename (nombre de archivo)

Salida : ERROR

OKAY

Casos de Prueba

1- Entrada	: prueba.doc	
Salida	: OKAY	
Condiciones	: El archivo existe.	
2- Entrada	: noexiste.doc	
Salida	: retorna ERROR	
Condiciones	: El archivo no existe.	

2 int dir_exists(char *dirname)

Entrada: dirname (directorio)

Salida : ERROR

OKAY

Condición	Salida Verdadero		Salida Falso	
dirname[1] == ':' y strlen(dirname) == 3 y dirname[strlen(dirname-1)] == '\'	1	El directorio es del tipo "<disco>:\".	2	El directorio no es del tipo "<disco>:\".
dirname[0] == '\' y strlen(dirname) ==1	3	El directorio es del tipo "\".	4	El directorio no es del tipo "\".
chdir(dirname)	5	Puede cambiar de directorio.	6	No puede cambiar de directorio.

Casos de Prueba

1- Entrada : c:\
   Salida : retorna OKAY
   Condiciones : 1

2- Entrada : \
   Salida : retorna OKAY
   Condiciones : 3

3- Entrada : c:\ms\
   Salida : retorna OKAY
   Condiciones : 2, 4, 5

4- Entrada : c:\noexiste\
   Salida : retorna ERROR
   Condiciones : 2, 4, 6

3 int VerifNbreArch(char *filename, short wildcards)

Entrada: filename (nombre de arhivo)

wildcards (se permite, o no, wildcards)

Salida : ERROR

OKAY

Condición	Salida Verdadero		Salida Falso	
VerifFormatoArch() == ERROR	1	El formato del archivo es incorrecto.	2	El formato del archivo es correcto.
filename == "CON"	3	filename == "CON"	4	filename != "CON"
filename == "AUX"	5	filename == "AUX"	6	filename != "AUX"
filename == "PRN"	7	filename == "PRN"	8	filename != "PRN"
filename == "CLOCK$"	9	filename == "CLOCK$"	10	filename != "CLOCK$"
filename == "LPT1"	11	filename == "LPT1"	12	filename != "LPT1"
filename == "LPT2"	13	filename == "LPT2"	14	filename != "LPT2"
filename == "LPT3"	15	filename == "LPT3"	16	filename != "LPT3"
filename == "NULL"	17	filename == "NULL"	18	filename != "NULL"
filename == "COM1"	19	filename == "COM1"	20	filename != "COM1"
filename == "COM2"	21	filename == "COM2"	22	filename != "COM2"
filename == "COM3"	23	filename == "COM3"	24	filename != "COM3"
filename == "COM4"	25	filename == "COM4"	26	filename != "COM4"

Casos de Prueba

1- Entrada : archivo = "PRUE*.DOC"
wildcards = 0

Salida : retorna ERROR (no se permiten wildcards)

Condiciones : 1

```
2- Entrada     : archivo    = "CON"
                 wildcards = 0
   Salida      : retorna ERROR (nombre de dispositivo)
   Condiciones : 3

3- Entrada     : archivo    = "AUX"
                 wildcards = 0
   Salida      : retorna ERROR (nombre de dispositivo)
   Condiciones : 5

4- Entrada     : archivo    = "PRN"
                 wildcards = 0
   Salida      : retorna ERROR (nombre de dispositivo)
   Condiciones : 7

5- Entrada     : archivo    = "CLOCK$"
                 wildcards = 0
   Salida      : retorna ERROR (nombre de dispositivo)
   Condiciones : 9

6- Entrada     : archivo    = "LPT1"
                 wildcards = 0
   Salida      : retorna ERROR (nombre de dispositivo)
   Condiciones : 11

7- Entrada     : archivo    = "LPT2"
                 wildcards = 0
   Salida      : retorna ERROR (nombre de dispositivo)
   Condiciones : 13
```

```
 8- Entrada      : archivo    = "LPT3"
                   wildcards = 0
    Salida       : retorna ERROR (nombre de dispositivo)
    Condiciones  : 15

 9- Entrada      : archivo    = "NULL"
                   wildcards = 0
    Salida       : retorna ERROR (nombre de dispositivo)
    Condiciones  : 17

10- Entrada      : archivo    = "COM1"
                   wildcards = 0
    Salida       : retorna ERROR (nombre de dispositivo)
    Condiciones  : 19

11- Entrada      : archivo    = "COM2"
                   wildcards = 0
    Salida       : retorna ERROR (nombre de dispositivo)
    Condiciones  : 21

12- Entrada      : archivo    = "COM3"
                   wildcards = 0
    Salida       : retorna ERROR (nombre de dispositivo)
    Condiciones  : 23

13- Entrada      : archivo    = "COM4"
                   wildcards = 0
    Salida       : retorna ERROR (nombre de dispositivo)
    Condiciones  : 25
```

```
14- Entrada      : archivo    = "PRUEBA.DOC"
                   wildcards = 0
    Salida       : OKAY
    Condiciones  : 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22,
                   24, 26
```

4 int VerifFormtoArch(char *filename, short wildcards)

Entrada: filename (nombre de archivo)

wildcards (se permite, o no, wildcards)

Salida : ERROR

OKAY

Condición	Salida Verdadero		Salida Falso	
strlen(filename) >= MAXFILE	1	La longitud del nombre del archivo es mayor que 8.	2	La longitud del nombre del archivo es menor o igual que 8.
strchr(filename,',')	3	El nombre del archivo tiene una ','.	4	El nombre del archivo no tiene una ','.
strchr(filename,'.')	5	El nombre del archivo tiene un '.'.	6	El nombre del archivo no tiene un '.'.
strchr(filename,'/')	7	El nombre del archivo tiene un '/'.	8	El nombre del archivo no tiene un '/'.
!wildcards	9	No se aceptan wildcards	10	Se aceptan wildcards.
strchr(filename,'?')	11	El nombre del archivo tiene un '?'.	12	El nombre del archivo no tiene un '?'.
strchr(filename,'*')	13	El nombre del archivo tiene un '*'.	14	El nombre del archivo no tiene un '*'.

Casos de Prueba

1-	Entrada	: archivo = "MASLARGOS"
		wildcards = 0
	Salida	: retorna ERROR
	Condiciones	: 1
2-	Entrada	: archivo = "PRU,EBA"
		wildcards = 0
	Salida	: retorna ERROR
	Condiciones	: 2, 3
3-	Entrada	: archivo = "PRU.EBA"
		wildcards = 0
	Salida	: retorna ERROR
	Condiciones	: 2, 4, 5
4-	Entrada	: archivo = "PRU/EBA"
		wildcards = 0
	Salida	: retorna ERROR
	Condiciones	: 2, 4, 6, 7
5-	Entrada	: archivo = "PRU?EBA"
		wildcards = 0
	Salida	: retorna ERROR
	Condiciones	: 2, 4, 6, 8, 9, 11, 14
6-	Entrada	: archivo = "PRU*EBA"
		wildcards = 0
	Salida	: retorna ERROR
	Condiciones	: 2, 4, 6, 8, 9, 12, 13

```
7- Entrada      : archivo    = "PRU?EBA"
                  wildcards = 1
   Salida       : OKAY
   Condiciones  : 2, 4, 6, 8, 10, 11, 14
-------------------------------------------------------
8- Entrada      : archivo    = "PRU*EBA"
                  wildcards = 1
   Salida       : OKAY
   Condiciones  : 2, 4, 6, 8, 10, 12, 13
-------------------------------------------------------
9- Entrada      : archivo    = "PRUEBA"
                  wildcards = 0
   Salida       : OKAY
   Condiciones  : 2, 4, 6, 8, 9, 12, 14

-------------------------------------------------------
10- Entrada     : archivo    = "PRUEBA"
                  wildcards = 1
   Salida       : OKAY
   Condiciones  : 2, 4, 6, 8, 10, 12, 14
```

5 int VerifPath(char *path)

Entrada: path (directorio)

Salida : ERROR

OKAY

Condición	Salida Verdadero		Salida Falso	
strchr(path, '/')	1	El directorio tiene '/'.	2	El directorio no tiene '/'.
*path != '\'	3	El directorio no comienza con '\'.	4	El directorio comienza con '\'.
*subdir1 != '\0'	5	No se llegó al fin del directorio.	6	Se llegó al fin del directorio.
subdir2 != NULL	7	Tiene subdirectorios.	8	No tiene subdirectorios
i != subdir2	9	No se terminó de armar el nombre del subdirectorio.	10	Se terminó de armar el nombre del subdirectorio.
*subdir != '\0'	11	Hay un subdirectorio.	12	No hay un subdirectorio
VerifFormatoArch() == ERROR	13	El formato del nombre del subdirectorio tiene errores.	14	El formato del nombre del subdirectorio no tiene errores.

Casos de Prueba

1- Entrada : tes/is
   Salida : retorna ERROR
   Condiciones : 1

2- Entrada : tesis
   Salida : retorna ERROR
   Condiciones : 3

3- Entrada : \
   Salida : retorna OKAY
   Condiciones : 2, 4, 7, 9, 10, 11, 14, 6

4- Entrada : \tesis\doc\
   Salida : retorna OKAY
   Condiciones : 2, 4, 7, 9, 10, 11, 14, 5, 8, 12, 6

5- Entrada : \tesis\doc\
   Salida : retorna OKAY
   Condiciones : 2, 4, 5, 7, 9, 10, 11, 14, 8, 12, 6

6- Entrada : \maslargos\doc\
El subdirectorio tiene una longitud mayor que la permitida.

Salida : retorna ERROR

Condiciones : 2, 4, 5, 7, 9, 10, 11, 13

6 int ExisteNodo(char *NodoName)

Entrada: NodoName (nombre del nodo)

Archivo mapa.txt

Salida : ERROR

OKAY

Condición	Salida Verdadero		Salida Falso	
(mapa = fopen(NODOMAPA,"r"))== ERROR	1	No se puede abrir el archivo para lectura "mapa.txt".	2	No se puede abrir el archivo para lectura "mapa.txt".
!encontro y (ch=fgetc(mapa)) != EOF	3	No se encontró el nombre del nodo en el archivo y no se llegó al fin del archivo.	4 5	Se encontró el nombre del nodo en el archivo. Se llegó al fin del archivo.
ch == ' '	6	El caracter es un blanco.	7	El caracter no es un blanco.
ch == '\n'	8	El caracter es un fin de línea.	9	El caracter no es un fin de línea.
ch != '\n' y ch != ' '	10	El caracter no es un fin de línea ni es un blanco.	11 12	El caracter es un blanco. El caracter es fin de línea.
nbre	13	Se tiene un nombre de nodo.	14	No se tiene un nombre de nodo.
!strcmp(mapanodo, NodoName)	15	Se econtró el nombre del nodo deseado en el mapa.	16	No se econtró el nombre del nodo deseado en el mapa.
encontro	17	Se econtró el nombre del nodo deseado en el mapa.	18	No se econtró el nombre del nodo deseado en el mapa.

Casos de Prueba

1-	Entrada	: nodo1 No puede abrir mapa.txt.
	Salida	: retorna -2
	Condiciones	: 1
2-	Entrada	: nodo1 nodo1 es el primer nombre de nodo que está en mapa.txt.
	Salida	: retorna OKAY
	Condiciones	: 2, 3, 7, 9, 10, 14, 13, 6, 15, 17, 4
3-	Entrada	: nodo2 nodo2 no es el primer nombre de nodo que está en mapa.txt.
	Salida	: retorna OKAY
	Condiciones	: 2, 3, 7, 9, 10, 14, 6, 11, 12, 13, 16, 8, 15, 17, 4
4-	Entrada	: nodo6 No existe nodo6 en mapa.txt.
	Salida	: retorna ERROR
	Condiciones	: 2, 3, 6, 7, 8, 9, 10, 11, 12, 13, 14, 16, 18, 5

7 int fnsplit(char *path, char *drive,
char *dir, char *name,
char *ext, unsigned int *flags)

Entrada: path (drive-directorio-archivo)

Salida : drive (disco)

dir (directorio)

name (nombre de archivo)

ext (extensión del archivo)

flags (indica en qué partes se dividió path)

ERROR

OKAY

Condición	Salida Verdadero		Salida Falso	
strchr(path, ':')	1	Tiene disco.	2	No tiene disco.
length < MAXDRIVE	3	La longitud del disco es menor que 3.	4	La longitud del disco no es mayor o igual a 3.
strchr(camino, '*') != '\0'	5	Tiene un '*'.	6	No tiene un '*'.
strchr(camino, '?'	7	Tiene un '?'.	8	No tiene un '?'.
*ultdir != '\'	9	El directorio no termina con '\'.	10	El directorio termina con '\'.
subdir[i] == '.'	11	El directorio tiene un '.'.	12	El directorio no tiene un '.'.
*ultdir == '\'	13	El directorio termina con '\'.	14	El directorio no termina con '\'.
*ultdir == '.'	15	El directorio termina con '.'.	16	El directorio no termina con '.'.
length < MAXPATH	17	La longitud del directorio es menor que 80.	18	La longitud del directorio es mayor o igual que 80.
archivo != NULL y archivo != '\0'	19	No tiene archivo.	20	Tiene archivo.
auxext >= archivo y *auxext == '.'	21	Tiene extensión.	22	No tiene extensión.
length < MAXEXT	23	La longitud de la extensión es menor que 5.	24	La longitud de la extensión es mayor o igual a 5.
archivo != NULL y archivo != archivof	25	El archivo tiene nombre.	26	El archivo no tiene nombre.
length < MAXFILE	27	La longitud del archivo es menor que 9.	28	La longitud del archivo es mayor o igual a 9.

Casos de Prueba

1- Entrada : c1:\tesis\prueba.doc

Salida : "La longitud del disco es incorrecta"
retorna ERROR

Condiciones : 1, 4

2- Entrada : c:\...\prueba.doc
El directorio tiene 80 caracteres.

Salida : "La longitud del camino es incorrecta"
retorna ERROR

Condiciones : 1, 3, 6, 8, 10, 13, 16, 18

3- Entrada : c:\tesis\prueba.docu

Salida : "La longitud de la extensión es incorrecta"
retorna ERROR

Condiciones : 1, 3, 6, 8, 10, 14, 16, 17, 20, 21, 24

4- Entrada : c:\tesis\pruebaste.doc

Salida : "La longitud del archivo es incorrecta"
retorna ERROR

Condiciones : 1, 3, 6, 8, 10, 14, 16, 17, 20, 21, 25, 28

5- Entrada : \tesis\prueba.doc

Salida : drive = "\0"
dir = "\tesis\"
name = "prueba"
ext = "doc"
flags = DIRECTORY | EXTENSION | FILENAME
retorna OKAY

Condiciones : 2, 6, 8, 10, 13, 16, 17, 20, 21, 23, 25, 27

```
6- Entrada      : c:\prueba
  Salida        : drive = "c:"
                  dir   = "\"
                  name  = "prueba"
                  ext   = "\0"
                  flags = DRIVE | DIRECTORY | FILENAME
                  retorna OKAY
  Condiciones   : 1, 3, 6, 8, 10, 13, 16, 17, 20, 22, 25,
                  27
```

```
7- Entrada      : c:..
  Salida        : drive = "c:"
                  dir   = ".."
                  name  = "\0"
                  ext   = "\0"
                  flags = DRIVE | DIRECTORY
                  retorna OKAY
  Condiciones   : 1, 3, 6, 8, 9, 11, 14, 15, 17, 19,  26
```

```
8- Entrada      : c:...
  Salida        : drive = "c:"
                  dir   = ".."
                  name  = "\0"
                  ext   = "."
                  flags = DRIVE | DIRECTORY | EXTENSION
                  retorna OKAY
  Condiciones   : 1, 3, 6, 8, 9, 11, 14, 15, 17, 20, 21,
                  23, 26
```

```
9- Entrada      : c:\tesis\
  Salida        : drive = "c:"
                  dir   = "\tesis\"
                  name  = "\0"
                  ext   = "\0"
                  flags = DRIVE | DIRECTORY
                  retorna OKAY
  Condiciones   : 1, 3, 6, 8, 10, 13, 16, 17, 19, 22, 26
```

```
10- Entrada      : c:tesis\.doc
    Salida       : drive = "c:"
                   dir   = "tesis\"
                   name  = "\0"
                   ext   = ".doc"
                   flags = DRIVE | DIRECTORY |EXTENSION
                   retorna OKAY
    Condiciones  : 1, 3, 6, 8, 10, 13, 16, 17, 20, 21, 23,
                   26
```

```
11- Entrada      : c:\tesis\prue*.doc
    Salida       : drive = "c:"
                   dir   = "\tesis\"
                   name  = "prue*"
                   ext   = ".doc"
                   flags = DRIVE | WILD | DIRECTORY
                   |EXTENSION
                   retorna OKAY
    Condiciones  : 1, 3, 5, 8, 10, 13, 16, 17, 20, 21, 23,
                   25, 27
```

```
12- Entrada      : c:\te?is\prueba.doc
    Salida       : drive = "c:"
                   dir   = "\te?is\"
                   name  = "prueba"
                   ext   = ".doc"
                   flags = DRIVE | WILD | DIRECTORY
                   |EXTENSION
                   retorna OKAY
    Condiciones  : 1, 3, 5, 6, 7, 8, 10, 13, 16, 17, 20,
                   21, 23, 25, 27
```

CONCLUSIONES

El objetivo de la presente tesis fue acceder, a través de Novell NetWare, desde una estación de trabajo a la información almacenada en el disco local de otra. Este objetivo se logró en su totalidad.

Los comandos que se suman a los ya existentes en NetWare son:

- **PDIR** : Visualiza la información acerca de archivos y directorios del disco de otra estación de trabajo.

- **PCOPIA** : Copia un archivo desde el disco local de la estación de trabajo origen al disco local de la estación de trabajo destino (RXPCOPIA) o viceversa (TXPCOPIA).

Estos comandos usando el protocolo SPX establecen, en forma transparente para el usuario, una conexión con una aplicación residente en la estación de trabajo remota (TSRCOM).

- **TSRCOM** Busca la información requerida por los comandos PDIR o RXPCOPIA y la transmite a través de la red. En caso de tratarse de TXPCOPIA, recibe la información y la copia en el archivo destino.

INSTALACION

El diskette nro. 1 contiene los archivos:

- pdir.exe
- pcopia.exe
- tsrcom.exe
- mapa.txt

Todos los archivo contenidos en este diskette deberán copiarse en un subdirectorio, llamado por ejemplo: c:\tsr.

Este subdirectorio deberá incluirse en el camino de búsqueda de la variable de ambiente **PATH** en el archivo **autoexec.bat**. También en este archivo se deberá llamar a la aplicación residente TSRCOM el acceso a los nuevos comandos desde que la computadora es encendida.

El archivo **mapa.txt** que se entrega es un ejemplo, deberá crearse uno similar con las direcciones de red y nodo de las estaciones en que se desee trabajar. Este archivo no puede faltar.

También deberán inicializarse en el archivo **net.cfg** las variables de ambiente correspondiente al protocolo SPX:

Protocol IPXODI

```
SPX ABORT TIMEOUT 200
SPX LISTEN TIMEOUT 540
SPX VERIFY TIMEOUT 540
```

Es importante recordar que **las PCs no deben tener instalados programas residentes** para su correcto funcionamiento.

El diskette nro. 2 contiene los fuentes desarrollados:

- ipxspx.h
- ipxspx.c
- conexion.h
- conexion.c
- dirarch.h
- dirarch.c
- pdir.c

- pcopia.c
- tsrcom.c

BIBLIOGRAFIA

[TAN/88] Andrew S. Tanenbaum
Computer Networks, Prentice Hall, 1988, pp. 3-27, 196-199

[NOV/91] Novell
Novell NetWare Versión 3.11 Utilities Reference Novell, NOVELL, 1991.

[RAL/90] Davis Ralph
NetWare Programmer's Guide, Adison-Wesley, 1990, cap. 14 y pp. 589-601.

[DAY/92] Michael Day
Enterprise series: Downsizing to NetWare, New Riders Publishing, 1992.

[MIL/90] Mark A. Miller
LAN Protocol Handbook, M&T Books, 1990, cap. 3 y cap. 8.

[SHE/90] Tom Sheldon
NetWare 386: The Complete Reference, McGraw Hill, 1990, pp. 32-37, 109-113, 865-866 y 870-872.

[SHE/91] Tom Sheldon
Novell NetWare: Manual de Referencia, McGraw Hill, 1991, cap. 2.

[WEB/91] Douglas Weber
Novell NetWare a su Alcance, McGraw Hill, 1991, cap. 1 y 2.

[BOD/90] Bill Bodine
A Comparison of NetWare IPX, SPX and NetBIOS, Novell Research, Agosto 1990.

[TUR/90] Paul Turner
NetWare Communications Processes, Novell Research, Septiembre 1990.

[NOV] Novell (R)
Client Transport Protocol API for C, cap. 2.

[ROS/90] Charles G. Rose
Programmers's Guide To NetWare, McGraw Hill, 1990, cap. 14.

[DON/74] Madnik Donovan
Operating Systems, McGraw Hill, 1974, cap. 6.

[TAN/92] Andrew S. Tanenbaum
Modern Operatin Systems, Prentice Hall, 1992, pp. 145-173 y 329-332.

[Bev/90] Roberto Bevilacqua
Apuntes de Arquitectura y Sistemas Operativos, CECEN, 1990.

[GAB/88] Alejandro Gabay
Apuntes de Laboratorio III, CECEN, 1988.

[Wol/94] Van Wolverton
Guia Completa MS-DOS 6.2, McGraw Hill, 1994, pp. 71, 96-97, 131.

[DOS] DOS
DOS Disk Operating System, 511-513.

[KER/91] Brian W. Kernighan - Dennis M. Ritchie
El Lenguaje de Programación C, Prentice Hall, 1991.

[SYM/91] Zymantec Corporation
Zortech C++ Compiler v3.0 Function Reference, Symantec Corp., 1991.

[BOR/90] Borland International
Turbo C++ Programmer's Guide, Gren Hill Road, 1990.

[SCA] Leo J. Scanlon
80286 Programación Ensamblador en Entorno DOS, Anaya, pp. 257-266.

[NOR/87] Peter Norton
The Norton Guides, Peter Nortor Computing, 1987.

[MYE/84] Glendford J. Myers
El Arte de Probar el Software, El Ateneo, 1984.