INTRODUCCION

Las Redes de Computadoras han revolucionado enormemente el mundo de la Informática. Desde su aparición en la década de los 80, ellas han transformado radicalmente la forma de trabajo en el ambiente de oficina moderna. Igual efecto han tenido en los ambientes industriales donde la necesidad de trabajar en equipo es vital para la productividad.

Sin embargo el grado de conectividad e integración que estas redes ofrecen ha evolucionado lentamente comparado con la demanda de los servicios esperados.

Entre los pasos que se han dado para resolver este problema tenemos los siguientes:
Crear un esquema de trabajo abierto y flexible, llamado Cliente- Servidor donde la unión y la igualdad son las normas, es decir, procesamiento en forma cooperativa. Clientes y Servidores trabajan juntos para llevar a cabio una tarea .
Varias son las soluciones que se han intentado, entre ellas tenemos las siguientes:

- Servidores de Archivos: Con un servidor de archivos, el cliente pasa las solicitudes de los registros del archivo bajo la red hacia el servidor de archivos.
- Servidores de Base de Datos: Con un servidor de base de datos , el cliente pasa las solicitudes SQL como mensajes al servidor de la base de datos. Los resultados de cada comando SQL son retornados bajo la red.
- Servidores de Transacciones: Con un servidor de transacciones, el cliente invoca a los procedimientos remotos que residen en el servidor con una sentencia SQL. Estos procedimientos remotos que se encuentran en el servidor ejecutan un grupo de

sentencias SQL llamadas transacciones. Estas aplicaciones tienen el nombre de procesamiento de la transacciòn en linea.

- Servidores de Aplicación: Con un servidor de aplicaciòn, se debe suplir del còdigo para el cliente y el servidor. Este tipo de servidores no son necesariamente una base de datos centralizada.

La diferencia del Esquema Cliente-Servidor de las otras formas de software distribuido, es que èste tiene las siguientes caracteristicas distintivas:

Servicio: Cliente-Servidor es principalmente una relaciòn entre procesos que estàn corriendo en màquinas separadas. El proceso servidor es un proveedor de servicios, mientras que el cliente es un consumidor de servicios. En escencia, cliente-servidor muestra una clara separaciòn de una funciòn basada en la idea de servicio.

Recursos compartidos: Un servidor puede servir a muchos clientes al mismo tiempo y regular su acceso a recursos compartidos.

Protocolos Asimètricos: Existen muchas relaciones de muchos a uno entre clientes y servidor. Los clientes siempre inician el diàlogo solicitando un servicio mientras que los servidores estàn esperando pasivamente por las solicitudes de los clientes.

Transparencia de la localidad: El servidor es un proceso que puede residir en la misma màquina como el cliente ò en una màquina diferente que estè conectada a la red.

Mezclar e Igualar: El software cliente-servidor es independiente del hardware y de las plataformas de software de sistemas operativos. Este sistema es capaz de mezclar e igualar las plataformas cliente-servidor.

Intercambio de informaciòn basado en mensajes. Este mensaje es el mecanismo de servicio de solicitudes y respuestas.

Encapsulaciòn de servicios. Un mensaje le dice al servidor que servicio es solicitado, entonces es subido al servidor para determinar el trabajo que debe realizarse.

Escalabilidad. Los sistemas cliente-servidor pueden ser escalados horizontal ò verticalmente. Escalar horizontalmente significa añadir ò borrar estaciones clientes con solo un ligero impacto en el rendimiento. Escalar verticalmente significa emigrar hacia hacia grandes y ràpidas màquinas ò multiservidores.

Integridad. El còdigo y datos del servidor son mantenidos centralmente, los cuales dan como resultado un mantenimiento barato y el cuidado de la integridad de los datos compartidos, al mismo tiempo los clientes se mantienen personales e independientes.

La comunicación Cliente - Servidor requiere mecanismos de Transporte del tipo requerimientos/respuesta como son los siguientes:

NetBIOS, Sockets de TCP/IP, Named-Pipes y RPC.

En este trabajo se va a hacer un estudio de todas las alternativas de solución al problema planteado, desarrollando ciertos prototipos y analizando las ventajas y desventajas del esquema cliente-servidor.

CAPITULO I:

1.1 ARQUITECTURA DE SOFTWARE DE COMUNICACIONES

1.1. MODELO OSI: (Open Systems Interconnection Model)

FIG. 1.1. MODELO OSI de 7-CAPAS

La complejidad del Software de Comunicaciones, obliga a que éste sea desarrollado modularmente, es decir, con varias capas de Software. El alcance de cada capa es hasta cierto punto arbitrario, aunque se busca que cada capa agrupe un conjunto de funciones afines. Así tenemos que se han desarrollado varios modelos de Software de Comunicaciones. La proliferación de los modelos ha conducido a la falta de compatibilidad entre ellos.

A fin de remediar esta falta de compatibilidad entre los modelos , entre los años 1977 y 1984 se desarrolló el Modelo OSI, que se compone de 7 capas y es mostrado en la Figura 1.1. Los principios aplicados para el establecimiento de ellas fueron los siguientes:

1.- Una capa se creará en situaciones en donde se necesita un nivel diferente de abstracción.

2.- Cada capa deberá efectuar una función bien definida.

3.- La función que desempeñará cada capa deberá seleccionarse con la intención de definir protocolos normalizados internacionalmente.

4.- Los límites de las capas deberán seleccionarse tomando en cuenta la minimización del flujo de información a través de las interfases.

5.- El número de capas deberá ser lo suficientemente grande para que diferentes funciones no tengan que ponerse en la misma capa y, por otra parte, también deberá ser lo suficientemente pequeño para que su arquitectura no llegue a ser difícil de manejar.

Debemos recalcar que cada capa por sí sola constituye un protocolo. Así un protocolo es un conjunto de reglas y convenciones entre los participantes de la red.

1. Capa Física

La capa física se ocupa de todos los detalles del hardware de Comunicaciones. Esta capa sólo entiende de bits, es decir, no le importa

el contenido y significado de la información. Su función principal es la de asegurar que cuando un extremo envía un bit con valor de 1, el otro extremo reciba exactamente el bit con ese valor en el otro extremo, y no como un bit 0.

2. Capa de Enlace

La tarea principal de la capa de enlace consiste en transformar un medio de transmisión común y corriente en una línea de transmisión de datos sin errores para la capa de Red. Para realizar esta tarea, el emisor divide los datos que se van a en transmitir en varios segmentos que son transmitidos en forma secuencial, aunque con la posibilidad de alterar la secuencia en cualquier momento.

3. Capa de Red

La capa de Red se ocupa de la operación de la subred. Su función principal es la de encaminar los paquetes del origen al destino. Para esto la capa de red emplea dos métodos básicos: Datagramas y Circuitos Virtuales (CV).

El método de Datagramas puede ser comparado con el Sistema de Correo [TANE 91]. El usuario deposita "el mensaje" en el sistema, y espera que éste realice su mejor esfuerzo para llevarlo a su destino. Si la "carta" se pierde en el correo, el usuario NO PUEDE esperar que el sistema de

correo la "envíe" nuevamente. El usuario es el único responsable de la "retransmisión" de todos sus mensajes.

El Circuito Virtual es comparable a un sistema telefónico. El usuario debe "marcar un número", "mantener" la comunicación hasta que lo desee, y finalmente "colgar" el teléfono. Igualmente ***la capa de red con circuito virtual es la responsable de establecer, mantener y terminar la conexión***.

4. Capa de Transporte

La función principal de la capa de transporte consiste en aceptar los datos de la capa de sesión, dividirlos en unidades más pequeñas si es necesario, pasarlos a la capa de red y asegurarse que todos ellos lleguen correctamente al otro extremo. Esta capa introduce ***mayor confiabilidad*** en la transmisión de datos, provee de la ***recuperación de errores*** y de ***control de flujo*** entre dos puntos finales (emisor - receptor).

5. Capa de Sesión

Permite que los usuarios de diferentes máquinas puedan establecer sesiones entre ellos. A través de una sesión se puede llevar a cabo un transporte de datos ordinario, tal y como lo hace la capa de transporte, pero mejorando los servicios que ésta proporciona y que se utilizan en algunas aplicaciones.

Una sesión permite al usuario acceder a un sistema de tiempo compartido a distancia, o transferir un archivo entre dos máquinas.

6. Capa de Presentación

Se ocupa de todos los aspectos relacionados con el ***formato de los datos que se transmiten***. Por ejemplo, el código de representación ASCII, EBCDIC, ordenamiento de bytes en palabras de más de un octeto, encripción de datos, etc.

Esta capa proporciona independencia en la representación de datos (formato de los datos y niveles de señales) para las aplicaciones de los procesos.

7 Capa de Aplicación

La capa de aplicación proporciona el ***acceso al ambiente OSI para usuarios*** y además proporciona servicios de información distribuida.

Contiene una variedad de protocolos que se necesitan frecuentemente, algunos sistemas de archivos tienen diferentes convenciones para denominar un archivo, así como diferentes formas de representar las líneas de texto. Trabajos como éste, así como el de correo electrónico, la entrada de trabajo a distancia, el servicio de directorio y otros servicios de proposición general y específicos, también corresponden a esta capa.

1.2 ARQUITECTURA BASADA EN EL PROTOCOLO TCP/IP

La Arquitectura OSI es definida como un esfuerzo, más bien académico, orientado a guiar el desarrollo de los modernos Protocolos de Comunicaciones.

En la práctica, sin embargo, son otros los productos (arquitecturas y protocolos) más comunmente usados. Uno de ellos y uno de los más importantes, es el protocolo TCP/IP que tiene su origen en la Arquitectura ARPANET (descrita más adelante).

Características:

El Departamento de Defensa (DOD) ha producido estandares para un conjunto de protocolos de comunicación [STAL 90c] . Sus motivaciones son mayores que las mismas de la ISO y cualquier sistema consumidor de computadores. DOD necesita tener eficiencia y un costo efectivo de comunicaciones a través de computadores heterogéneos. DOD ha escogido desarrollar sus propios protocolos y arquitectura en lugar de adoptar los estandares internacionales. Entre las razones para esta decisión, podemos citar 3 de ellas:

2.- Los requerimientos para un DOD específico tienen un principal impacto en el diseño de protocolos y arquitectura.

3.- Existen diferencias filosóficas concernientes a la naturaleza apropiada de una arquitectura de comunicaciones y de su protocolo.

Existen cuatro diferencias fundamentales entre el modelo OSI y TCP/IP (Protocolos de Internet) :

1.- El concepto de jerarquía versus capas.

2.- La importancia de interred.

3.- La utilidad de servicios sin conexión.

4.- La aproximación al manejo de funciones.

La Arquitectura del protocolo TCP/IP está basada en una forma de comunicación que envuelve 3 agentes: Procesos, Procesadores ("hosts"), y Redes. Los ***procesos*** son entidades fundamentales que se comunican entre sí. Los procesos se ejecutan en ***"hosts"*** (estaciones) los cuales pueden frecuentemente soportar múltiples procesos simultáneos. La comunicación entre los procesos toma lugar a través de las *redes* hacia las cuales los "hosts" están unidos.

Estos tres conceptos dan un principio fundamental del protocolo TCP/IP: la transferencia de información hacia un proceso puede ser acoplada primeramente obteniéndolo del host en el cual reside el proceso y entonces obtener el proceso dentro del "host". Estos 2 niveles de demultiplexamiento pueden ser manejados independientemente. Entonces una red necesita solo tener conocimiento de los datos de la ruta entre los "hosts".

Enfatizamos aquí que el factor importante es el orden de la jerarquía de los protocolos. La designación de las capas es puramente para la explicación de las proposiciones. Una entidad en una capa puede usar los servicios de otra entidad en una capa más baja pero no en una capa adyacente.

Con los conceptos que hemos mencionado, es natural organizar los protocolos en cuatro capas.

1.- Capa de acceso a la red.

2.- Capa de internet

3.- Capa host-a-host.

4.- Capa proceso/aplicación

1.- La **capa de acceso a la red** contiene protocolos que proveen acceso para la red de comunicación. Los protocolos en esta capa están entre un nodo de comunicación y un "host" conectado a la red o su equivalente lógico. Una función de todos estos protocolos es la de ser el dato de ruta entre los hostales conectados a la misma red. Otros servicios que también provee son el control de flujo y el control de errores entre hostales, y varias características de servicio, ejemplos de esto último son la prioridad y seguridad.

Una entidad de la capa de red es invocada típicamente por una entidad de Internet hacia una capa de host-a-host, pero también puede ser invocada por la entidad de la capa proceso/aplicación.

2.- La **capa internet** consiste de procedimientos requeridos para permitir que los datos atraviesen múltiples redes entre "hosts'. Ella debe proveer una función de enrutamiento. Este protocolo (Internet) es usualmente implementado dentro de los hosts y "gateways". Un "gateway" es un

procesador conectado entre dos redes cuya función principal es retardar el dato entre las redes usando un protocolo de internet.

3.- La **capa "host-a-host"** contiene entidades de protocolos con la habilidad de deliverar el dato entre dos procesos en diferentes computadores "host". Una entidad del protocolo en este nivel puede (ó no) proveer de una conexión lógica entre entidades de alto nivel.

Otros posibles servicios de la capa host-a-host incluyen control de error y control de flujo y la habilidad de distribuir los datos con señales de control no asociados con una conexión lógica.

En este nivel necesitamos cuatro tipos generales de protocolos, un protocolo de datos orientado a conexión, un protocolo datagrama, un protocolo de lenguaje, y un protocolo de datos de tiempo-real. Cada uno tiene diferentes requerimientos de servicio.

- Un **protocolo formal de datos**, orientado a conexión se caracteriza por la necesidad de puntualidad de una secuencia deliverada de datos. Muchas aplicaciones de procesamiento de datos podrian usar dicho servicio por lo anteriormente mencionado.
- Un **protocolo datagrama** es un protocolo de baja sobrecarga de información, funcionalmente mínimo que es apropiado para el tráfico, particularmente en aplicaciones que prefieren implementar su propia funcionalidad orientada a conexión.
- El **protocolo de lenguaje** el cual se caracteriza por la necesidad de manejar un flujo estable de datos con una mínima variación de retardo.

- Un **protocolo de tiempo real** tiene las características demandadas de ambos protocolos: Protocolo orientado a conexión y el protocolo de lenguaje.

4.- La **capa de proceso / aplicación** contiene protocolos para recursos compartidos (computador a computador) y de acceso remoto (terminal a computador).

1.2.1 OPERACION DE TCP e IP

La figura 1.2 indica como estos protocolos están configurados para las comunicaciones. Para aclarar que el total de comunicaciones puede consistir fácilmente de múltiples redes, las redes que la constituyen son llamadas subredes.

Una clase de protocolo de acceso a la red, como Token Ring, es usado para conectar un computador a una subred. Este protocolo capacita al "host" para enviar datos a través de la subred al otro "host" o, en el caso de un host a otra subred, hacia una ruta.

IP se lo puede establecer en todos los extremos de los sistemas y en las rutas. Actúa como un amortiguador para mover un bloque de datos hacia un host, a través de una ó más rutas, al otro host. **TCP** puede establecerse sólo al final de los sistemas; mantiene el rastro de

los bloques de datos para asegurarse que todos ellos lleguen a la aplicación apropiada.

Para que una comunicación sea exitosa, toda entidad en un sistema completo debe tener una única dirección. Actualmente, se necesitan dos niveles de direccionamiento . Cada host en la subred debe tener una única dirección global de internet; esto permite que los datos sean conducidos hacia su propio "host". Cada proceso en un "host" debe tener una dirección única dentro del mismo; esto le permite al protocolo "host-a-host" conducir los datos a su propio proceso. Estas últimas direcciones son conocidas como **puertos**.

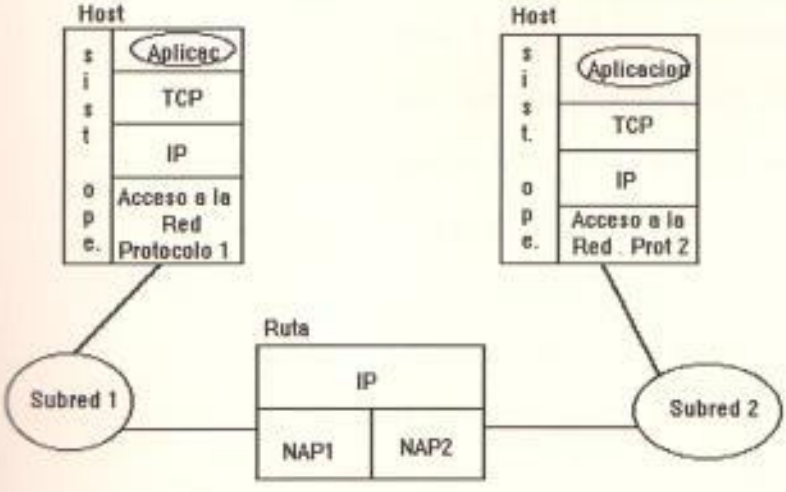


FIG. 1.3 Comunicación usando la arquitectura del protocolo TCP/IP

1.2.2 INTERFASES DEL PROTOCOLO

Cada capa en el protocolo TCP/IP interactúa con sus capas inmediatamente adyacentes. Si la capa de proceso se encuentra en la fuente, ella hace uso de los servicios de la capa host-a-host y provee de datos a la capa que está abajo de ella. Una relación similar existe en la interfase de las capas "host-a-host" e internet y hacia la interfase de internet y las capas de acceso a la red. En el destino, cada capa ingresa los datos hacia la siguiente capa superior a ella.

El uso de cada capa individual no es requerida para la arquitectura. Como sugiere la figura 1.4 , es posible desarrollar aplicaciones que directamente invoquen los servicios de cualquiera de las capas. La mayoría de las aplicaciones requieren un protocolo seguro de extremo a extremo. Algunas aplicaciones de proposición especial no necesitan de los servicios de TCP. Otras aplicaciones, como el simple protocolo de manejo de la red (SNMP), usa un protocolo de host a host conocido como protocolo datagrama usuario ("user datagram protocol": **UDP**); otros pueden hacer uso directamente de IP. Las aplicaciones que no envuelven a la red y que no necesitan de TCP han sido desarrolladas para invocar directamente a la capa de acceso a la red.

Es importante notar que el protocolo TCP/IP no está limitado a los 5 protocolos estandarizados, en lugar de ello, una variedad de aplicaciones y otros procesos pueden hacer uso de esta arquitectura.

La figura 1.4 muestra la posición de algunos de los protocolos fundamentales comunmente implementados como parte del protocolo TCP/IP.

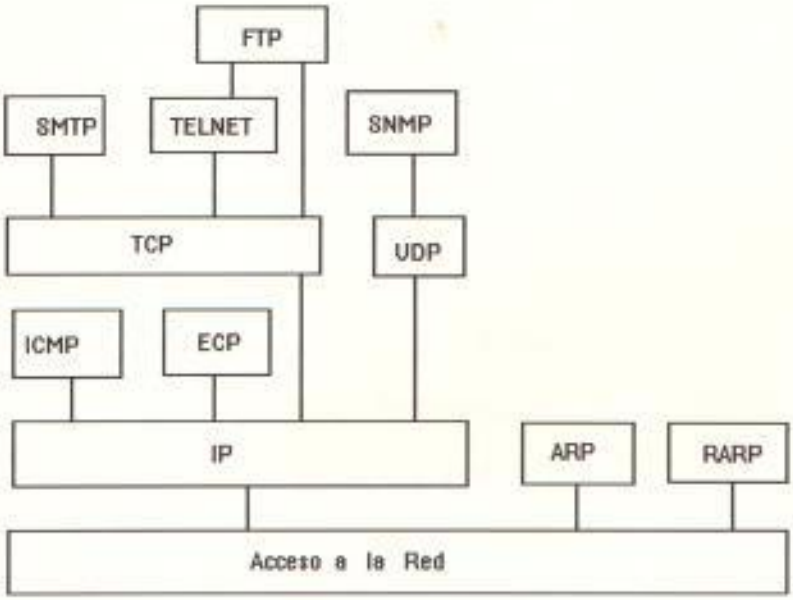


FIG. 1.4 Dependencias de protocolos

ARP = "address resolution protocol"

EG = " external gateway protocol" TCP = " transport control protocol"

FTP = " file transfer protocol" UDP = " user datagram protocol".

ICMP= " internet control message protocol"

IP = " internet protocol"

RARP= " reverse address resolution protocol"

SMTP= " simple mail transfer protocol"

NMP= " simple network management protocol"

1.3 PROTOCOLO SNA ("Systems Network Architecture")

El protocolo SNA está formado de siete capas:

- Control Físico
- Control del Enlace de Datos
- Control del camino ("path")
- Control de la Transmisión.
- Control del Flujo de Datos.
- Servicios de Presentación.
- Servicios de Transacción.

La figura 1.5 hace una comparación de SNA con los protocolos OSI y TCP/IP.

OSI		TCP/IP		SNA	
7	Aplicaciones	7	Proceso/ Aplicación	7	Servicios de Transacción
6	Presentaciones	6		6	Servicio de Presentación
5	Sesión	5	Host - a - Host	5	Control del Flujo de Datos
4	Transporte	4		4	Control de Transmisión
3	Red	3	Internet	3	Control del camino
2	Enlace de Datos	2	Acceso a la Red	2	Control del Enlace de Datos
1	Físico	1		1	Control Físico

FIG. 1.5. Comparación de arquitecturas de comunicaciones

Control Físico.

La capa de control físico corresponde a la capa uno del modelo OSI, ella especifica la interfase física entre los nodos. Esta capa tiene dos tipos de interfases: Uno para enlaces de comunicación serial y otro para enlaces paralelos. Las interfases para los **enlaces de comunicación serial** son usados en la mayoría de las conexiones nodo-a-nodo. Las redes SNA también

pueden incluir enlaces paralelos de alta velocidad entre un mainframe y un procesador de comunicación frontal-final.

Control de Enlace de Datos.

La capa de control de enlace de datos corresponde a la capa dos del modelo OSI. Esta capa da una referencia segura de los datos a través del enlace físico. El protocolo especificado para enlaces de comunicaciones seriales es SLDC (Control de Enlace de Datos Sincrónico), el cual es básicamente un subconjunto de HLDC (Control de Alto Nivel del Enlace de Datos). [STAL93].

Control de la Ruta:

Esta capa crea un canal lógico entre puertos finales, a los cuales nos referimos como ***unidades direccionables de la red*** (NAU: "network address units"). Un NAU es una entidad de la capa de aplicación, capaz de ser direccionada y de intercambiar datos con otras entidades. Las principales funciones de esta capa capa son: enrutamiento y control de flujo. El control de la ruta ("Path Control") está basado en conceptos del grupo de transmisión, de una ruta explícita y de una ruta virtual. Un ***grupo de transmisión*** es un conjunto de uno o más enlaces físicos entre nodos adyacentes de la red. Una ***ruta explícita*** es un camino entre puntos finales definidos como una secuencia ordenada de grupos de transmisión. Una ***ruta virtual*** es una conexión lógica entre dos puntos finales que está dinámicamente asignada a una ruta explícita.

La función principal de la subcapa de control del grupo de transmisión es hacer que el conjunto de enlace en dicho grupo aparezca en las capas superiores como un solo enlace físico. Las principales ventajas de esto son, el ***aumento de la seguridad y de la capacidad de transmisión*** entre nodos adyacentes. El protocolo en esta subcapa acepta las unidades de datos y las ubica en una pila FIFO (Primero en entrar, primero en salir).

Cada unidad de datos es enviada por turnos sobre el siguiente enlace físico disponible, usando SLDC. Debido a los errores o diferencias en el retardo de propagación, las unidades pueden arribar fuera de orden hacia el otro extremo del grupo de transmisión. Se usan secuencias numéricas para que el protocolo de la entidad que está recibiendo las unidades las reordene.

Otra función que se ejecuta en esta capa es el ***Bloqueo***. Cuando un grupo de transmisión consiste de un solo enlace, el protocolo de la entidad puede bloquear las unidades de datos que están llegando en una unidad grande antes de la transmisión. Esto puede incrementar la eficiencia, por ejemplo, reduciendo el número del canal de las operaciones I/O (entrada/salida) que el control de enlace de datos necesita ejecutar. La función de esta subcapa es inusual y parece convenir mejor como parte de la capa dos del modelo OSI.

La subcapa de control de la ruta explícita es responsable del enrutamiento, rutas explícitas son predefinidas en SNA y cada nodo mantiene su información en forma secuencial (X,Y) = (secuencia numérica explícita,

siguiente nodo). Así, cualquier dato que está llegando debe contener una ruta numérica explícita.

El criterio de este número es que el protocolo de la entidad selecciona el siguiente nodo y pasa el dato a la subcapa de control del grupo de transmisión.

La ***subcapa de control de la ruta virtual*** proporciona una *conexión lógica* cuyo tráfico de sesiones es multiplexado y aplicado en los mecanismos de control de flujo.
Esta subcapa tiene la habilidad de segmentar las unidades de datos de las capas superiores para probar su eficiencia. Las unidades de datos segmentados deben ser reensamblados en el otro extremo.

Control de Transmisión

El control de transmisión es la siguiente capa más alta de la arquitectura SNA, la cual corresponde aproximadamente a la capa cuatro del modelo OSI . La capa de control de transmisión es ***responsable de establecer, mantener, y terminar las sesiones SNA***. Una sesión, la cual corresponde a la conexión de transporte OSI es una relación lógica entre dos puntos finales (NAU's). La capa de control de transmisión puede establecer una sesión en respuesta a una solicitud de la siguiente capa más alta (control de flujo), desde un proceso de aplicación, o por sus propias proposiciones de control.

Esta capa está compuesta por dos módulos: el manejador del punto de conexión (CPMGR: connection point manager), el cual maneja la transferencia de datos individuales, y el control de la sesión, el cual maneja asuntos a nivel-sesión. El CPMGR ejecuta las siguientes funciones:

- **Enrutamiento**: El enrutamiento es esencialmente una función de demultiplexamiento. Las unidades de datos que están llegando son enrutadas a la entidad apropiada, la cual está en la misma capa o alguna superior.
- **Encapsulamiento**: Los mensajes de salida son encapsulados en una unidad de datos; que es la cabecera, la cual es añadida al dato, contiene información de control de la expeditada deliveración, "pacing", encripción, y otras funciones de control.
- **Control de Flujo:** El control de flujo es el mismo mecanismo usado en las rutas virtuales. En este caso, es usado solamente por los extremos, para controlar el flujo de las unidades de datos. El tamaño de ventana usado es fijo. El control de la sesión es invocado para activar o desactivar una sesión, además como una secuencia numérica faltante cuando CPMGR detecta un error en la sesión.

Control del Flujo de Datos:

Dentro de las conexiones lógicas llamadas sesiones que pueden ser establecidas entre aplicaciones , SNA ha escogido dividir el manejo de sesiones dentro de dos capas: la capa de control de transmisión, la cual es transmisión orientada y, como hemos dicho, corresponde a la capa cuatro del

modelo OSI; y, la capa de control de flujo de datos que es orientada al usuario-final y corresponde igualmente a la capa cinco del modelo OSI. Esta capa es responsable de ***proveer servicios relacionados a la capa de sesión*** que son visibles para los procesos usuario-final y terminales. Las funciones principales están en las siguientes categorías:

Modo Enviar/Recibir: Aquí especificamos tres modos: full duplex, flip-flop half duplex, y contención half-duplex.

Encadenamiento ("Chaining") : Es un mecanismo para delinear las secuencias de transmisión para las proposiciones de recuperación.

Corchetes ("Bracketing") : "Bracketing" contribuye con una secuencia de intercambios. Este concepto puede ser usado en la definición y control de las secuencias de transacción.

Opciones de Respuesta: Aquí se especifican tres modos de respuesta para cada unidad de datos: (1) no envía una respuesta, (2) envía una respuesta sólo en el caso de una excepción, y (3) siempre envía una respuesta.

Mantenerse/Apagarse: Esta categoría puede solicitar una espera temporal o permanente hacia el flujo de datos.

Servicios de Presentación:

Recientemente el tope de dos capas de la arquitectura SNA fueron consideradas como una sola capa llamada *capa de servicios del manejo de datos* (FMD: function management data). La capa de servicios FMD comprime un conjunto de funciones y servicios provistos para el usuario final. Esa capa corresponde a las capas seis y siete del modelo OSI. Las funciones están divididas en dos capas que son, servicios de presentación y servicios de transacción. La ***capa de servicios de presentación*** incluye los siguientes servicios:

- **Traslado de formato**: Este servicio permite a cada extremo tener una revisión diferente del intercambio de datos. Por ejemplo, puede ser usada para permitir a una aplicación manejar múltiples tipos de terminales.

- **Compresión y Compactación**: Los datos son comprimidos a nivel de bit o byte usando procedimientos especificados, para reducir el volumen de transmisión.

- **Soporte del programa de Transacción**: Este servicio controla la comunicación a nivel de conversación entre programas de transacción: (1) "Loading" (Cargando) e Invocando programas de transacción; (2) manteniendo los protocolos en modo de envío y recepción; e (3) imponiendo el uso correcto del parámetro verbo (comandos de comunicación que usa el protocolo) y restricciones de secuenciamiento.

La mayoría de estos servicios corresponden a la capa seis del modelo OSI, además algunas de las funciones de soporte del programa de transacción están más relacionadas con la naturaleza de la capa siete.

La Capa de Servicios de Transacción:

Esta capa está principalmente destinada a proveer servicios del manejo de la red, debido a esto, dichos servicios son directamente usados por un usuario final, como un manejador del sistema o un operador de la red. Entre los servicios del manejo de red están incluídos los siguientes:

- **Servicios de Configuración**: Estos servicios permiten a un operador empezar o reconfigurar una red, activando y desactivando enlaces.

- **Servicios de Operador de la Red**: Aquí se incluyen funciones del operador sin configuración, además de la colección y muestra de redes estáticas, y la comunicación de datos de usuarios y procesos al operador de la red.

- **Servicios de Sesión:** El servicio de sesión soporta la activación de una sesión en medio de los usuarios y aplicaciones. En efecto, esta es la interfase con el usuario hacia la capa del control de transmisión.

- **Servicios de Mantenimiento y Manejo:** Estos servicios proveen facilidades para las pruebas de la red y ayudan en la identificación y aislamiento de fallas.

1.3.1 ENCAPSULACION SNA

SNA no requiere del uso de una cabecera en cada capa de la jerarquía de encapsulamiento. La figura 1.6 muestra la estrategia de encapsulación SNA. La pieza básica del dato en la encapsulación SNA es la ***unidad de respuesta*** (RU), la cual contiene el dato del usuario o la información de control de la red. Un manejador de servicios crea RU's de los datos del usuario. La capa de servicios FMD (Función del Manejo de Datos/ "Function Management Data") puede ejecutar ciertas transformaciones en los datos para acomodar los servicios de presentación o puede añadir la información de control relacionada a series de RU's. Los últimos están contenidos en la cabecera FMD. Esta cabecera es opcional y, si es usada, puede sólo aparecer ocasionalmente (al inicio de la serie o "bracket") .

La capa de control de transmisión entonces añade una cabecera solicitud/respuesta (RH), conteniendo la información de control para sí mismo y para la capa de control del flujo de datos. En lugar de crear una cabecera separada, la capa de control de flujo de datos pasa parámetros a la capa de control de transmisión para incluirlos en la cabecera.

Las siguientes dos capas añaden sus propias cabeceras para sus requerimientos de información. El control del enlace de datos

también añade una cola ("trailer"); esto corresponde al "trailer" para HDLC.

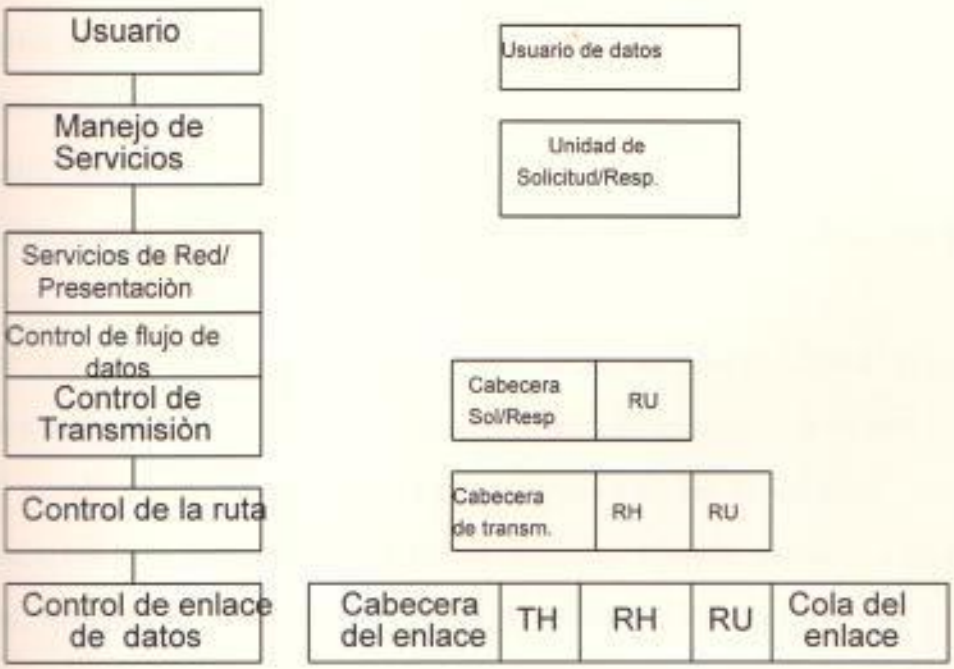


FIG. 1.6 Encapsulaciòn de datos SNA.

CAPITULO II:

2. PLATAFORMA CLIENTE-SERVIDOR

La computación bajo el ambiente Cliente-Servidor provee un ambiente abierto y flexible donde la unión e igualdad es la regla. Las aplicaciones en el servidor correrán predominantemente en PC's y en otras máquinas de escritorio que están familiarizadas con las LAN's (Redes de Area Local). Los servidores se sentirán familiarizados con las LAN's y conocerán exactamente cómo comunicarse con sus clientes PC.

Para que los mainframes tengan éxito como servidores, deben aprender a relacionarse con las PC's como si estuvieran bajo un mismo nivel en una LAN. En este mundo donde la igualdad es un factor principal, los mainframes no pueden tratar a los PC's como terminales tontos; necesitan soportar protocolos de igual-a-igual, interpretar los mensajes de las PC's, dar servicio de sus archivos a clientes PC's en sus formatos originales y proveerles de datos y servicios de la forma más directa.

2.1 ESQUEMAS TRADICIONALES.

Varios sistemas con diferentes arquitecturas han sido llamados Cliente-Servidor, algunos vendedores de sistemas, frecuentemente usan este término como si sólo fuera aplicado a sus paquetes específicos. Durante los últimos diez años la idea de dicho esquema es la de dividir una aplicación a lo largo de las líneas cliente/servidor para crear varias formas de soluciones de software de redes de Area Local. Cada una de estas soluciones se distingue

por la naturaleza del servicio que provee a sus clientes, como mostraremos en los siguientes ejemplos.

2.1.1 SERVIDORES DE ARCHIVOS

Con un servidor de archivos, el cliente (tipicamente un PC) hace solicitudes de registros de archivos sobre una red, como veremos en la figura 2.1. Los servidores de archivos son una forma muy primitiva de servicio de datos que necesita de muchos intercambios de mensajes sobre la red para encontrar el dato solicitado.

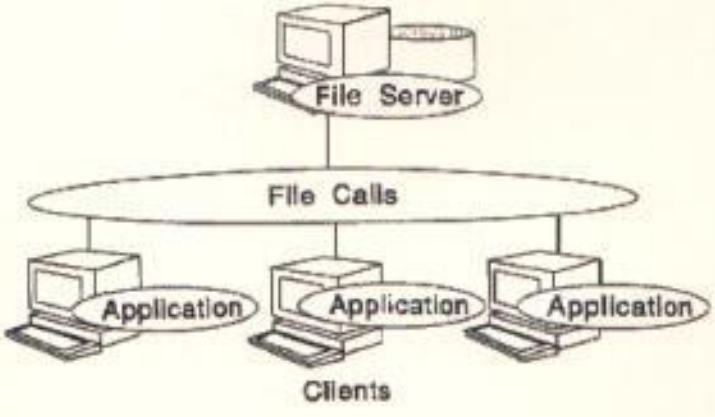


FIG. 2.1 Cliente/Servidor con Servidores de Archivos

2.1.2 SERVIDORES DE BASE DE DATOS.

Con un servidor de Base de datos, el cliente pasa las solicitudes SQL ("Structured Query Language") al servidor en forma de mensajes ,como veremos en la fig. 2.2. Los resultados de cada comando son rertornados sobre la red. El código que procesa la solicitud SQL y el dato residen en la misma máquina. El servidor usa su propio poder de procesamiento para encontrar el dato solicitado, en lugar de regresar todos los registros al cliente y permitirle encontrar su propio dato como fue el caso del servidor de archivos. El resultado es mucho más eficiente en cuanto al uso del poder de procesamiento distribuido.

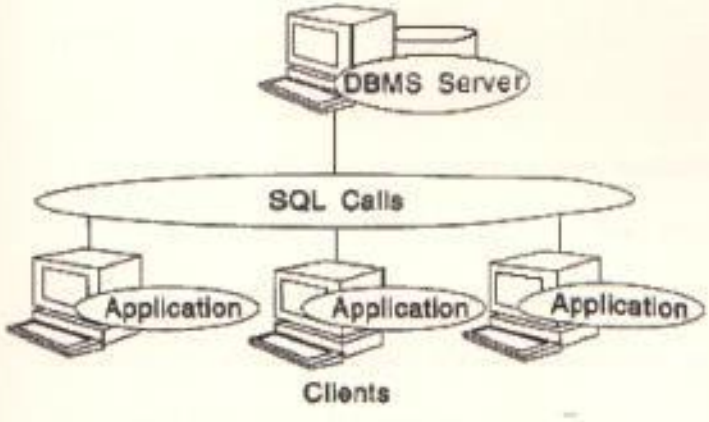


FIG. 2.2 Cliente/Servidor con Servidores de Base de Datos

2.1.3 SERVIDORES DE TRANSACCIONES.

Con un Servidor de transacciones, el Cliente invoca a los procedimientos remotos que residen en el servidor en una máquina con base de datos SQL, según lo ilustra la fig. 2.3. Estos procedimientos ejecutan un grupo de sentencias SQL; el intercambio en la red consiste de un solo mensaje del tipo solicitud/respuesta. Necesitaremos usar un protocolo de igual-a-igual para emitir la llamada al procedimiento remoto y obtener los resultados. Estas sentencias SQL pueden fallar o ser exitosas, como una unidad al ser agrupadas son llamadas **transacciones** . Se crea la aplicación cliente/servidor escribiendo el código para ambos componentes (cliente y servidor).

El componente **Cliente** usualmente incluye una interfase gráfica con el usuario, en cambio, el componente **Servidor** usualmente consiste de transacciones SQL en lugar de base de datos. Estas aplicaciones tienen un nombre, ***Procesamiento de Transacción en Línea*** **u OLTP** ("Online Transaction Processing").

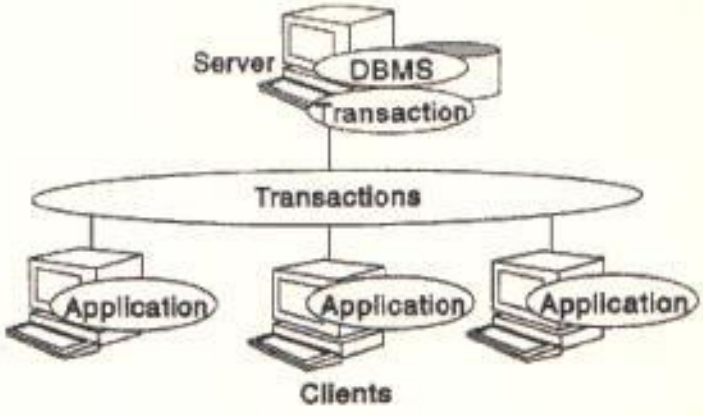
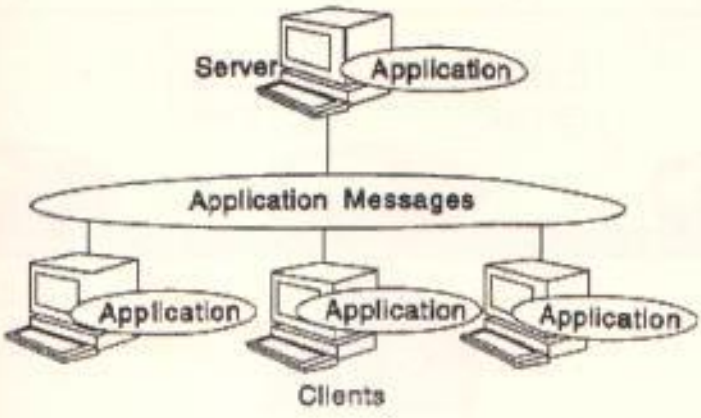


FIG 2.3. Cliente/Servidor con Servidores de Transacciones.

2.1.4 SERVIDORES DE APLICACIONES.

Con un Servidor de aplicaciones, podemos suplir el código para Cliente y Servidor, según lo muestra la fig. 1.4. Así como los servidores de transacción, los servidores de aplicaciones no son necesariamente una base de datos centralizada.

Los Servidores de Aplicaciones pueden también ser construidas en el tope de las bases de datos, para crear nuevos tipos de aplicaciones cliente/servidor. Por ejemplo: El producto "Note" de la Compañía Lotus, es un servidor de aplicación que maneja una información semi-estructurada (como textos y gráficos) en un ambiente de trabajo a manera de grupo boletín-pizarra. Los Servidores de aplicaciones de las Bases de Datos forman un principio de la nueva generación de imagen y documentos del flujo de trabajo de servidores.



2.4. Cliente/Servidor con Servidores de Aplicaciones.

2.1.5 ¿SERVIDORES BENEFICIADOS O CLIENTES BENEFICIADOS?

Hemos mostrado que los modelos cliente/servidor se distinguen por el servicio que ellos proporcionan. Las Aplicaciones Cliente/Servidor se diferencian en la manera de distribuir la aplicación entre el cliente y el servidor. El modelo del servidor beneficiado ubica más funciones en el servidor (FIG. 2.5). El modelo de cliente beneficiado hace lo contrario. Ejemplos de **servidores beneficiados** son los Servidores de transacciones y de aplicaciones . Servidores de bases de datos y de archivos son ejemplos de **clientes beneficiados**.

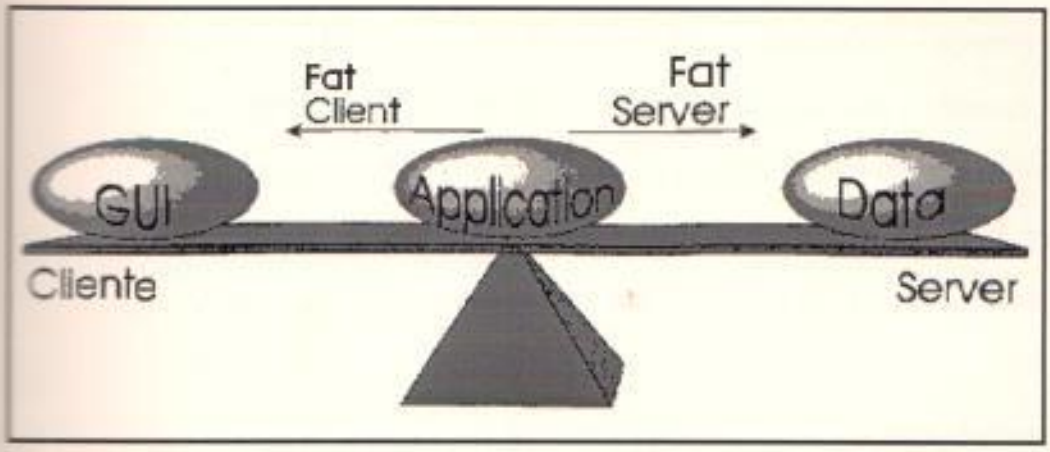


FIG. 2.5 ¿Clientes Beneficiados o Servidores Beneficiados?.

Los Clientes beneficiados son la forma tradicional de cliente/servidor. La mayor parte de las aplicaciones corren en el lado cliente de la ecuación. En ambos modelos, Servidores de archivos y base de datos, los clientes están concientes de cómo está organizado el dato y cómo está almacenado en el lado del servidor. Los Clientes beneficiados son usados como soporte de desiciones y software personal. Ellos ofrecen flexibilidad y oportunidades para crear herramientas frontal-final que permite a los usuarios finales crear sus propias aplicaciones finales.

Las aplicaciones en un Servidor Beneficiado ("Fat Server") son fáciles de manejar y desplegar en la red porque la mayoría de los códigos corren en los servidores. Los clientes beneficiados tratan de minimizar los intercambios en la red creando más niveles de servicio. Los Servidores de transacción, por ejemplo, encapsulan la base de

datos. En lugar de exportar datos originales, exportan los procedimientos (o métodos en trminología orientada a objetos) que operan en ese dato. El cliente en el modelo de servidores provee la herramienta GUI ("graphical User Interface": Interface Gráfica con el Usuario) e interactúa con el servidor a través de llamadas a procedimientos remotos.

Como podemos ver los servidores beneficiarios son más rápidos que los clientes beneficiados.

2.2. ESQUEMA CLIENTE-SERVIDOR

Las soluciones Cliente - Servidor dan como resultado bajos costos de mantenimiento y de desarrollo; además, brinda soluciones de alta productivad.

A través de estandares abiertos, el cliente-servidor puede además proveer de la menor juntura de integración de PC's y mainframes. El desarrollo de la aplicación Cliente-Servidor requiere se habilidades que incluyen **procesamiento de transacción, diseño de la base de datos, experiencia en comunicación, y una amigable interfase gráfica con el usuario.**

La mayoría de las soluciones cliente/servidor que hoy en día son implementadas en las PC LAN son personalizadas de acuerdo al grupo que las usa. Todo lo referente , desde directorios LAN para requerimientos de

seguridad debe ser propiamente configurado, frecuentemente por los mismos usuarios.

El Cliente/Servidor es la industria más sobresaliente. Nos provee de la oportunidad de crear nuestra propia definición. Como el nombre lo indica clientes y servidores son entidades lógicas separadas que trabajan juntas sobre una red para llevar a cabo una tarea. **¿Qué hace la diferencia de cliente/servidor de otras formas de software distribuido?** Se ha propuesto que todos los sistemas distribuidos tienen las siguientes características distintivas:

a) **Servicio**: Cliente/Servidor es primeramente una relación entre los procesos que están corriendo en máquinas separadas. El proceso en el servidor es un proveedor de servicios. El cliente es un consumidor de servicios. En esencia, cliente/servidor provee una clara separación de una función basada en la idea de servicio.

b) **Recursos Compartidos**: Un servidor puede atender a muchos clientes al mismo tiempo y regular su acceso a recursos compartidos.

c) **Protocolos Asimétricos**: Existe una relación de muchos a uno entre clientes y servidor. Los clientes siempre inician el diálogo solicitando un servicio. Los servidores están pasivamente esperando por las solicitudes de los clientes.

d) **Transparencia de la Localidad:** El servidor es un proceso que puede residir en la misma máquina como cliente o en una diferente a lo largo de la red. El software Cliente/Servidor usualmente disfraza la localidad del servidor para los clientes, redireccionando las llamadas de servicio cuando son necesitadas. Un programa puede ser un cliente, un servidor o ambos.

e) **Asociar e Igualar**: el software ideal cliente/servidor es independiente del hardware o software de plataformas de sistemas operativos ("operating system software platforms). Usted debe ser capaz de asociar e igualar las plataformas cliente/servidor.

f) **Intercambios basados en mensajes**: Clientes y Servidores están holgadamente acoplados a los sistemas, los cuales interactúan a través del mecanismo "passing-message" (transmitiendo -mensajes). El mensaje es el mecanismo de liberación para las solicitudes de servicio y respuestas.

g) **Encapsulamiento de Servicios**: El servidor es un "especialista". Un mensaje le dice al servidor que se solicita el servicio y entonces éste es puesto en el servidor para determinar cómo obtener el trabajo hecho. Los servidores pueden ser mejorados sin afectar a los clientes así como la interfase del mensaje publicado no es cambiada.

h) **Escalabilidad**: Un sistemas cliente/servidor puede ser escalado horizontalmente o verticalmente. Escalando horizontalmente significa

sumar o remover una estación de trabajo cliente con sólo un impacto insignificante en el rendimiento. Escalando verticalmente significa migración hacia una máquina servidora grande y rápida o multiservicio.

i) **Integridad**: El código servidor y dato servidor es mantenido centralmente , lo cual da como resultado un mantenimiento barato y la seguridad de la integridad de los datos compartidos. Al mismo tiempo, los clientes se mantienen personales e independientes.

1.3 VENTAJAS Y DESVENTAJAS

Ventajas:

"Downsizing" (Disminución de costos y tamaño): El proceso "downsizing" rompe hasta las aplicaciones de grandes "superminis" y tipos "mainframes" en módulos de programas que corren en uno o más servidores de la red. Las soluciones para el software Cliente/Servidor sobre el hardware estándar de bajo-costo es la fuerza manejadora detrás del "downsizing". También ellas permiten crear ambientes coherentes fuera de las autónomas máquinas de escritorio, es decir que la computación Cliente/Servidor combina lo mejor de los dos mundos: el costo efectivo y el poder adictivo de computadores de escritorio con acceso multiusuario para recursos y datos compartidos .

"Upsizing" (Aumento del tamaño): La vasta mayoría de PC's fueron originalmente conectadas a LAN's para compartir dispositivos periféricos de alto costo, como impresoras láser y "scanners".

Actualmente las LAN's son usadas primeramente en correo electrónico y para compartir bases de datos y archivos. Además se promete introducir una nueva generación del flujo de trabajo de software cliente/servidor , finamente modulada en niveles de interacción en trabajos de grupo.

"Rightsizing" mueve aplicaciones de la plataforma servidora más apropiada. Los clientes solicitan servicios sobre la red y además, el servidor adecuado para el trabajo requerido. En este modelo abierto un servidor puede ser un PC, un supermini, o un mainframe. "Rightsizing" mueve la información y el poder del servidor a los puntos de impacto. Iguala el trabajo del PC al del servidor sin tener que recurrir a las "islas de automatización".

El poder de cómputo cuesta menos. Resultan ser de bajos costos de mantenimiento y soluciones de alta productividad.

El acceso al poder de cómputo también cuesta menos. Un servidor puede atender a muchos clientes al mismo tiempo y compartir sus recursos.

El cliente/servidor gana cuando los requerimientos del poder de cómputo son altos y el número de terminales (estaciones de trabajo) es pequeño, como en el caso de un diseño auxiliado por computadora y operaciones de

manufactura empleando un puñado de ingenieros con requerimientos enormes de conocimientos de computación.

DESVENTAJAS:

Si una organización tiene muchas terminales y pocos requerimientos de poder computacional, quedarse con un mainframe tiene sentido. Por ejemplo, un sistema de reservaciones aéreas con miles de agentes y relativamente requerimientos modestos de computación trabaja bien en un ambiente de mainframes, como lo hace un centro de servicio al cliente que usa poder de cómputo para responder a preguntas sobre la facturación.

Si la desición basada en costos está cerca de un tris, los requerimientos de almacenamiento y los costos de soporte/mantenimiento pueden ser elementos de desempate con los de cliente/servidor. La regla de cajón aquí es que el cliente/servidor alcanza el borde de almacenamiento (almacenamiento cliente/servidor a dos mil dólares por Gigabyte (GB), es cerca de la mitad del costo de los dispositivos de almacenamiento de acceso directo). La computación de mainframes, por su parte , tiende a irle mejor en el área de costos de soporte.

Sin embargo, aun con el más bajo costo por GB, el almacenamiento en el mundo cliente/servidor no siempre resulta la mejor apuesta debido a que la tecnología no está sofisticada lo suficiente para explotar la ventaja del costo.

Mientras que en los sistemas de administración de discos mainframes permiten a las compañías implementar bases de datos que residen en decenas o aun cientos de unidades de discos, la tecnología de servidores requiere que la base de datos completa resida en una sola unidad de discos. Escencialmente, los datos están ahí, pero no pueden ser empleados muy fácilmente.

Mientras los costos de arrendamiento anual del software de mainframes pueden ser exhorbitantes, dependiendo del tamaño de la plataforma y de las tarifas del mejoramiento de la calidad ("upgrade"), no son necesariamente más altos que los costos del cliente/servidor. El software del cliente/servidor acumula costos de soporte que disminuyen su ventaja monetaria sobre los sistemas mayores.

El cliente/servidor es un área nueva, así que hay pocos profesionales experimentados en administrar y soportar las operaciones del cliente/servidor. Los costos de entrenamiento para hacer que la gente avance rápidamente son inmensos. Existen muchos productos actualmente que manejan áreas tales como automatización de sistemas, planeación del monitoreo de la capacidad, control de la programación de reprocesos, administración de bibliotecas, administración de cintas, administración de discos y seguridad.

En el mundo del cliente/servidor, dichas herramientas no existen o son demasiado incipientes para ser muy efectivos.

El cliente/servidor puede algunas veces entorpecer el canal de información debido al número de elementos de cómputo que requiere. Una demostración reciente de interoperabilidad entre cuatro diferentes plataformas de cómputo requirió 50 paquetes de software de comunicaciones.

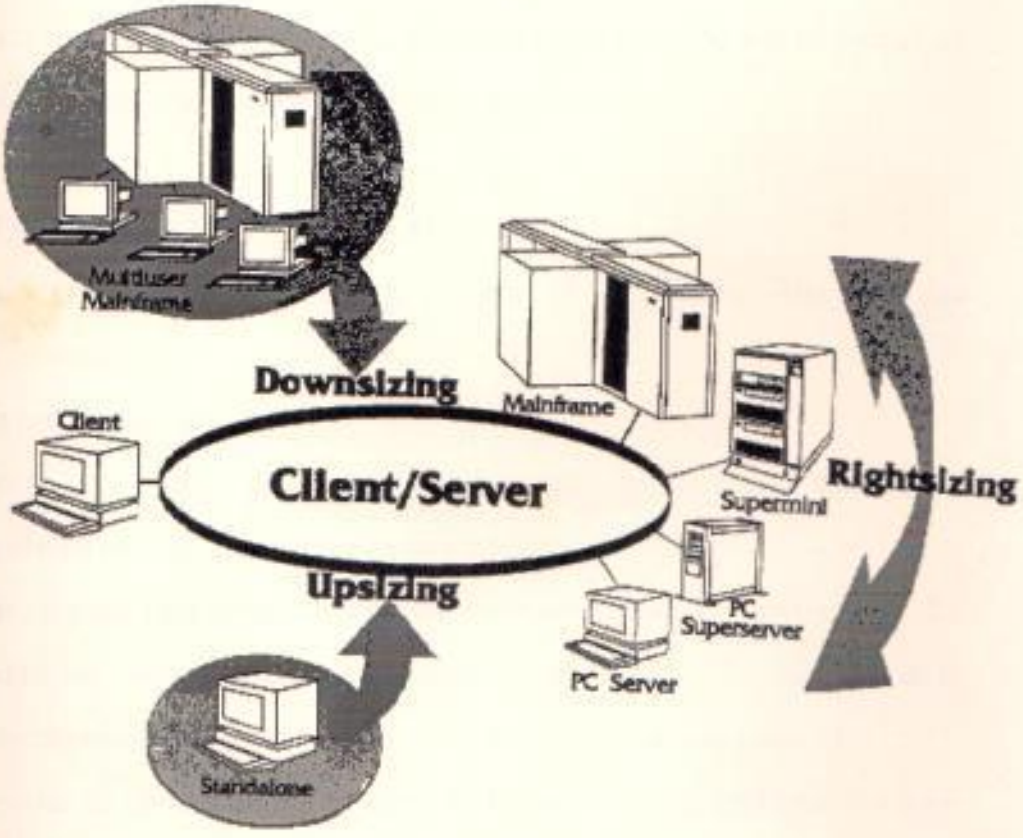


FIG 2.6 Mercado del Esquema Cliente- Servidor

CAPITULO III

3. TECNICAS DE PROGRAMACION CLIENTE-SERVIDOR

Comunicación entre procesos (IPC)

La comunicación entre procesos es el resultado de la multitarea. Actualmente las PC's pueden correr más de un programa al mismo tiempo, debido a la necesidad de intercambiar información y comandos entre los programas.

CARACTERISTICAS DEL IPC "IDEAL"

Durante la transferencia de datos entre procesos, el IPC "ideal" debe tener las siguientes características:

- Debe ser embebido ("built-in").
- Debe controlar el flujo de datos, de tal manera que el nuevo dato no se sobreescriba en el viejo, antes de que éste sea leído.
- Debe dar soporte para el intercambio de datos entre procesos sobre la red. La localidad de los procesos debe ser transparente al programador. La seguridad de la red y la recuperación de la misma debe estar integrada en un solo protocolo.
- Debe proveer de un mecanismo tal como la Llamada a Procedimientos Remotos (RPC), para facilitar la creación de transacciones basadas en aplicaciones entre procesos.
- Dar soporte para el intercambio de datos usando una variedad de estructuras de datos.
- Proveer de una interfase de programación que sea familiar, es decir, que no requiera aprendizaje de muchos comandos o de una metáfora de programación.

- Proveer de un protocolo que pueda ser implementado usando una mínima cantidad de códigos. Las comunicaciones entre procesos en un ambiente multitarea debería ser "natural" y fácil de usar.

Lo que hemos mencionado es lo que se espera de un protocolo bien diseñado para la transferencia de datos entre procesos.

3.1. PROGRAMACION CON NETBIOS

Antes de definir lo que es NETBIOS y su origen, describiremos brevemente ciertos conceptos que serán necesarios para su debida explicación.

Nodos: Son dispositivos que permiten las comunicaciones, conectados por enlaces de transmisión.

Red de Comunicación: Es una colección de nodos.

Redes "Broadcast": Son redes que comparten un mismo medio de transmisión. Las estaciones "oyen pero no escuchan"; sólo si los datos son para ellas los escuchan.

Datagrama: Es una técnica para el manejo de paquetes, en la cual, *no se establece una conexión lógica*, sino que cada paquete toma un tratamiento especial de acuerdo al tráfico de ellos. Los paquetes pueden llegar en diferente secuencia numérica y no necesariamente siguen la misma ruta.

Circuito Virtual: También es una técnica para el manejo de paquetes, la cual *establece una conexión lógica* entre los dos dispositivos que se están comunicando. Esta comunicación debe ser establecida antes de comunicarse. Los nodos no necesitan hacer ninguna conexión de ruta. Los paquetes

enrutados son encolados brevemente para verificar errores y tráfico muy cargado.

NETBIOS fue diseñado para un grupo de computadores personales, que comparten un medio común, "broadcast". Provee de ambas conexiones: Servicio orientado a conexión (Circuito Virtual) y un servicio sin conexión (Datagrama). Soporta "Broadcast"(todas las entidades en una red) y "Multicast" (grupo de entidades seleccionadas) .

La figura 3.1 muestra la relación de estos servicios, pero no cómo estos interactúan . En la mayoría de las implementaciones, un solo módulo provee de alguna forma de liberar el datagrama (similar a la capa IP en el protocolo TCP/IP) .

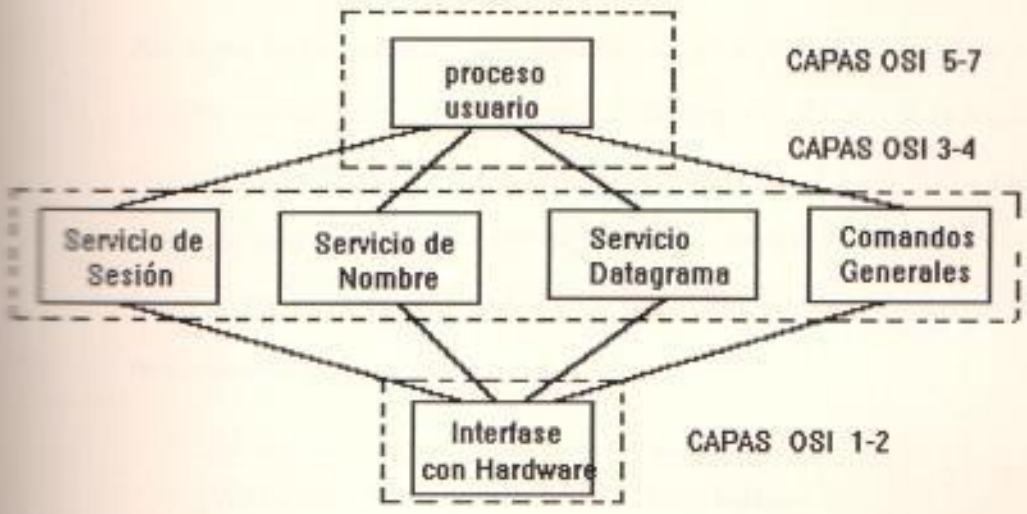


FIG. 3.1 RELACION ENTRE LOS SERVICIOS DE NETBIOS

Así como con la capa IP, con NetBIOS el proceso usuario no tiene acceso a ningún otro servicio. En muchos ambientes de "PC's" la aplicación que NetBIOS está usando es para archivos compartidos. En este caso, existe otra interfase de control sobre NetBIOS. Esta interfase es llamada Protocolo "Server Message Block" (SMB), Servidor de bloques de mensajes, y lo mostraremos en la figura 3.2.

FIG. 3.2 RELACION DE NetBIOS Y SMB EN EL MODELO OSI

ORIGEN DE NetBIOS

NetBIOS primero apareció en Agosto de 1984 con la tarjeta adaptadora de red IBM PC diseñado por Sytek Inc, de IBM. La red IBM PC fue primero IBM LAN, proveía de un rango de transmisión de datos de 2 Megabit por segundos a través de un cable4 coaxial, usando el método estándar de acceso de la popular industria "Carrier Sense Multiple Access Collision Detect" (CSMA/CD) que primero apareció con Ethernet IEEE 802.3.

NetBIOS está localizado en el adaptador de Red LAN IBM PC (LANA), el cual es un ROM BIOS extendido.

¿QUE ES EL "VERDADERO NetBIOS"?

Dentro de la línea de productos IBM, la versión actual de programas de soporte IBM LAN ofrece la implementación del "verdadero NetBIOS", porque provee de dispositivos "drivers" PC-DOS.

Una ventaja significativa de este programa es que permite a varios adaptadores LAN IBM's comunicarse con cada uno de los demás por medio de un PC intermediario o por medio de un PS/2 que esté corriendo en un programa de interconexión de la red IBM Token-Ring. Dicha ventaja capacita a las estaciones de trabajo IBM PC, de la red LAN, para comunicarse con otras estaciones de trabajo basadas en una red "Token-Ring". De tal manera podemos ver que sólo razones estratégicas dictan que la implementación de los programas de soporte IBM PC LAN sobrepasan la implementación original, como la industria estándar del "verdadero NetBIOS".

NetBIOS provee de 4 categorías de Servicios de Aplicación:

- Soporte del Nombre
- Soporte Datagrama
- Soporte Sesión
- Comandos Generales.

SOPORTE DEL NOMBRE

Los nombres son usados para identificar los recursos de NetBIOS, por ejemplo, para dos procesos que participan en una conversación, cada uno debe tener un nombre. El proceso Cliente identifica a un Servidor específico por su nombre, y el servidor puede determinar el nombre del cliente.

Un adaptador LAN individual de NetBIOS se distingue de los otros por uno o más nombres respectivos de red, los cuales permiten a las aplicaciones LAN dirigir sus mensajes hacia adaptadores específicos e indicar el adaptador que origina el mensaje. Cada nombre de red consiste de 16 caracteres, cada uno de los cuales es significativo , además las letras mayúsculas son diferentes de las minúsculas. Los nombres que usted puede crear no deben tener como primer caracter un valor binario de cero o un asterisco (*) . IBM reserva los valores de 00h a 1Fh para el 16o caracter, esto es porque se puede tener sólo 15 caracteres IBM PC LAN del nombre de la máquina.

La cantidad de nombres que puede usar (o está usando) un adaptador puede variar, así como también el número de adaptadores que puede usar un nombre dado, en cambio un adaptador puede usar cualquier nombre, adquirido convenientemente para registrarse en la LAN.

Hay 2 tipos de nombres: **Nombres Unicos y Nombres de Grupos**. Un nombre único debe ser único alrededor de la red. Un nombre de grupo no

tiene que ser único y todos los procesos que tienen el nombre del grupo dado le pertenecen (al grupo).

Existen 4 comandos adecuados para el servicio del nombre:

ADD_NAME	Añade un nombre único
ADD_GROUP_NAME	Añade un nombre de grupo.
DELETE_NAME	Borra el nombre
FIND_NAME	Determina si un nombre es registrado.

NetBIOS inicia las actividades de registración en respuesta a cualquiera de los dos tipos de comandos de NetBIOS: "**Add Name**" (Añadir nombre) y "**Add Group Name**" (Añade nombre del grupo). Un adaptador de red para los registros de un nombre primero transmite una petición de la red ("**a name-claim packet**": Un paquete de demanda del nombre) para poder usar el nombre. Luego, el tipo del paquete, Petición_de_Nombre o **Añade_Petición_de_Nombre_de_Grupo**, esto indica si el adaptador quiere registrar el nombre como un **nombre único** o como un **nombre de grupo** respectivamente.

Finalmente, una vez que el nombre es registrado exitosamente , cualquier nombre registrado excepto el primero puede ser borrado usando el comando "**Delete Name**" (Borrar Nombre). Los comandos del adaptador NetBIOS que sirven para borrar, borran la tabla del nombre NetBIOS (excepto el primer nombre), como en un sistema al resetearse (Ctrl-Alt-Del) y apagando la estación de trabajo.

El comando **FIND_NAME** fue añadido con la implementación de NetBIOS y determina si un nombre particular ha sido registrado por otro nodo de NetBIOS.

Nombres Unicos.

Si un adaptador trata de registrar un nombre como nombre único , entonces ningún otro adaptador que está operando en la LAN puede tener el mismo nombre registrado, porque falla al intentar registrarlo.

Nombre de Grupos.

Si un adaptador trata de registrar su nombre como un nombre de grupo, entonces no existe otro adaptador que puede estar usando este nombre como nombre único, ya que falla en el registro. Si el nombre no se está usando como nombre único, el adaptador tiene el derecho de usar el nombre en la LAN porque no es exclusivo, esto permite a otros adaptadores registrar el nombre como un nombre de grupo. Los nombres de grupo son muy usados para enviar mensajes de colecciones de estaciones de trabajo tal como un departamento o equipo.

La Tabla Nombre o Número del Nombre.

Si falla un intento en el registro del nombre , ésta es reportada a la estación de trabajo donde se encuentra la aplicación, para subsecuentes análisis retornando un código apropiado de error. En ausencia de las quejas de la red, el adaptador de NetBIOS ubica el nombre en una localidad que

mantiene reservada, una tabla interna conocida como tabla de nombres NetBIOS. Ella entonces reporta éxito en el registro del nombre a la aplicación LAN con el valor de **un-byte**.

El valor de un byte es un número sin signo referenciado al nombre NetBIOS llamado **número del nombre.** El número del nombre es secuencialmente usado en varios comandos de NetBIOS asociados con dicho nombre. NetBIOS asigna el valor del número del nombre en un incremento de módulo 255, usando la técnica de Round Robin ("Give everybody a turn": Cada estación en turno está dando una oportunidad para transmitir). El valor cero y 255 no son nunca asignados, y la primera entrada es permanentemente asignada por el adaptador basados en sus números seriales internos. Así los numeros son asignados en el orden 1,2,3,.....254,etc.

Una vez ubicado el nombre en la tabla de nombre, ésta autoriza al adaptador para que subsecuentemente contabilice las peticiones de registro de otros adaptadores que desean registrar nombres. Una vez añadidos, un nombre puede ser borrado de la tabla , permitiéndo usar algún otro nombre como nombre único.

Note que **la tabla de nombres** NetBIOS **es una tabla temporal** ubicada dentro de la memoria RAM y es reconstruida después de cada "boot" del sistema o adaptador reset. Debido a que cada adaptador tiene su propio nombre privado de la tabla, la resolución del nombre en NetBIOS es altamente autónoma alrededor de la LAN, sin requerir una administración central del nombre. Si un módulo de NetBIOS está soportando más de un

adaptador LAN dentro de una estación de trabajo, cada adaptador también tiene independientemente su propia tabla de nombres.

Nombre del Nodo Permanente

Todos los adaptadores LAN de IBM tienen un único número de seis-bytes asociado a ellos, garantizando ser único para todo adaptador IBM LAN y está contenido en un adaptador ROM. El número está referenciado por una variedad de nombres:

- Nombre del nodo permanente
- Diercción del nodo permanente
- Dirección "burned-in"
- Dirección universalmente administrada.
- Número de identificación de la unidad
- Dirección física
- Nombre del nodo local

Para todos los adaptadores LAN de IBM esta dirección se encuentra en el rango que está universalmente administrado por los comités estándares para LAN's y tiene dos bits de alto orden fijados a cero. Los valores de los dos bits de alto orden en el resto de los cuatro bytes varían por adaptador.

Nombres Simbólicos

Los nombres simbólicos son pseudónimos que están registrados en la tabla de nombres NetBIOS como nombres únicos o como nombres de grupos. Esto se hace porque es conveniente personalizar la dirección del adaptador LAN asociandolo a su nombre natural , por ejemplo: "Mònica".

Los adaptadores pueden recibir mensajes que son direccionados a ellos para usar solamente:

- Su dirección única de 48-bit derivada de su único número serial.
- Una indiscriminada dirección de transmisión general de X'FFFFFFFFFFFF'
- Una dirección funcional de bit-mapeado.
- Una dirección del grupo del valor-mapeado.

Una vez que el nombre simbólico ha sido resuelto en una dirección apropiada de 48-bits, NetBIOS necesita solamente esa dirección para conducir la comunicación. El nombre usado para hacer la asociación no es esencial hasta que se necesite resolver otra asociación, éste puede ser una dirección diferente de 48-bits, al nombre simbólico. Los nombres simbólicos pueden ser registrados y tambièn borrados.

SOPORTE DATAGRAMA Y DE SESION

Una vez que el adaptador està activo en una red, los programas de aplicación dentro de la estación de trabajo pueden usar NetBIOS para comunicarse con otras aplicaciones que están residiendo en la mismas o diferentes estaciones

de trabajo. Las aplicaciones pueden comunicarse usando datagramas o sesiones.

SOPORTE DATAGRAMA

Los Datagramas son mensajes cortos cuyos tamaños varían por la implementación de NetBIOS y no garantizan que el adaptador vaya más allá de su "mejor esfuerzo". Es indiferente en cuanto a que los mensajes arriben de manera segura, no provee de ninguna indicación de la recepción de mensajes por NetBIOS. La máquina receptora relacionada puede:

1.- No existir.

2.- Estar apagada.

3.- No estar pendiente de un datagrama.

En estos ejemplos, y en el caso de los problemas de red, el datagrama puede nunca ser recibido por ninguna estación de trabajo. La comunicación datagrama es una comunicación del tipo "envíe y ruegue", a menos que la aplicación que la está recibiéndo tome una acción explícita para transmitir un reconocimiento de recepción. La **principal ventaja** de la comunicación datagrama es que **puede consumir menos recursos** (estaciones de trabajo) que los que puede consumir por medio de la comunicación por sesión.

Existen dos tipos de comunicación datagrama : Datagramas "Broadcast" y Datagramas "plain" (sencillos). En ambos casos, los comandos de transmisión datagrama hacen referencia a un número existente del nombre local NetBIOS, asi como el nombre del nodo permanente, que sirve como

nombre original del datagrama. Este número del nombre puede estar asociado con un nombre único local o de grupo. Finalmente, los datagramas sencillos transmitidos hacia los grupos de nombres y datagramas "broadcast" tienen un nivel muy bajo en la seguridad de datos, debido a que ellos con un pequeño esfuerzo pueden ser interceptados .

a) Datagramas Broadcast

Los datagramas broadcast son totalmente **datagramas indiscriminantes** transmitidos con el comando de NetBIOS ***"Send Broadcast Datagram"*** (Enviar Datagrama Broadcast). Cualquier adaptador (incluyendo al que está transmitiendo) puede recibir un datagrama "broadcast" si ha utilizado previamente el comando ***"Receive Broadcast Datagram"*** (Recibir Datagrama "Broadcast") .

En general, la comunicación entre datagramas "broadcast" puede ser destruida, porque dos aplicaciones dentro de una misma estación de trabajo pueden fácilmente recibir datagramas "broadcast" que están relacionados con otra aplicación. Además, las aplicaciones que se ejecutan en las estaciones de trabajo que están corriendo el programa de IBM PC LAN han sido prevenidas específicamente frente al uso de esta comunicación.

b) Datagramas "Plain"

Los Datagramas "plain" son **datagramas discriminantes** transmitidos con el comando ***"Send Datagram"*** (Enviar Datagrama).. Así mismo, con los comandos de NetBIOS de Enviar Datagrama "Broadcast" , las

aplicaciones especifican un nombre del receptor de NetBIOS con el comando "Send Datagram" . Cualquier adaptador, incluyendo el adaptador que está transmitiendo, puede recibir un datagrama si previamente ha añadido el apropiado nombre del receptor y ha utilizado el comando ***"Receive Datagram"*** (Recibir Datagrama) a NetBIOS referenciandolo al número del nombre especificado en el comando.

Si una aplicación especifica el número del nombre FFh en un datagrama receptor, la aplicación puede recibir un datagrama de cualquier nombre que se encuentre en la tabla de nombres NetBIOS. Esto se lo hace referenciadolo con ***"receive_any datagram"*** (Recibir cualquier Datagrama). De cualquier manera, los comandos de recepción de datagramas para un número específico del nombre tienen prioridad sobre los comandos "Receive_Any Datagram" (Reciba_cualquier Datagrama)

Los Datagramas "plain" (sencillos) pueden ser transmitidos hacia los adaptadores usando el nombre como único nombre , o para grupos de adaptadores que comparten un nombre de grupo.

Soporte de Sesión

La Comunicación por Sesión crea una conexión segura para la comunicación de datos de dos-vías que puede existir por períodos extensivos. Dichas conexiones algunas veces son referenciadas a los Circuitos Virtuales.

Las Aplicaciones de comunicaciones pueden residir dentro de la misma estación de trabajo (sesión local) o en diferentes estaciones de trabajo (sesiones remotas). Cada aplicación constituye una mitad o un lado de la sesión.

La principal ventaja de este tipo de comunicación (sesión) sobre la del tipo datagrama es que el lado que está transmitiendo recibe siempre un estado del mensaje que envía, es decir que la comunicación datagrama proporciona el estado de la transmisión del mensaje.

La seguridad de la comunicación por sesión viene con un ligero "overhead" (encabezamiento) de las sesiones que se están creando y manteniendo además del protocolo de reconocimiento del paquete entre adaptadores.
Los siguientes comandos proveen el servicio de Sesión:

CALL	Call-Active open
LISTEN	Listen-Pasive open
SEND	Send session data
SEND_NO_ACK	Send Session Data
RECEIVE	Receive session data
RECEIVE_ANY	Receive session data
HANG_UP	End Session,
SESSION_STATUS	Retrieve session status.

NetBIOS requiere de un proceso para el cliente y otro para el servidor. El servidor primero utiliza una apertura pasiva con el comando LISTEN. El

cliente entonces se conecta al servidor cuando el cliente ejecuta el comando CALL.

a) Creando Sesiones

Las sesiones son creadas cuando una aplicación produce un comando ***"Listen"*** (Escuchar) referenciando a un nombre en la tabla de nombres NetBIOS. La aplicación puede usar un nombre existente en la tabla, así como el nombre del nodo permanente, o añadir uno de su propiedad.

El comando ***"Listen"*** también especifica el nombre remoto que una petición de la aplicación debe usar para calificar una sesión compañera, y puede usar un asterisco (*) como primer caracter del nombre remoto. En este caso, el resto de los 15 caracteres son ignorados y el NetBIOS local permite a la segunda aplicación usar cualquier nombre para calificar una sesión compañera.

Una segunda aplicación entonces genera un comando de NetBIOS llamado ***"Call"*** (Llamar), el cual se refiere al nombre en la tabla de NetBIOS que la primera aplicación está esperando como nombre del compañero. El comando "Call" también hace referencia al nombre de la primera aplicación refiriéndose a su vez a su propia tabla de nombres.

El doble nombre se asemeja totalmente al criterio de ambas aplicaciones para crear una sesión y los comandos pendientes de "Listen" y "Call" entonces se completan.

Note que la secuencia: Primero Escuchar ("Listen"), entonces Llamar ("Call") no puede ser invertida exitosamente.

Cada aplicación entonces recibe una notificación del establecimiento de la sesión y un valor de un-byte sin signo que se refiere al número local de la Sesión de NetBIOS (LSN) que el adaptador asocia con la sesión. El LSN es analógico al manejo de archivos de PC-DOS.

NetBIOS asigna el valor LSN de manera incremental, módulo 255, de la forma ROUND-ROBIN. Los valores cero y 255 no son nunca asignados. También si ambos lados de la sesión son sesiones locales, le son asignados dos números, uno para cada lado. En este caso, cualquier aplicación puede usar cualquier LSN. En general, no hay restricción en que dos LSN tengan el mismo valor, igualmente si ambas sesiones son locales. El procedimiento para la creación de una sesión está resumido en la figura 3.3

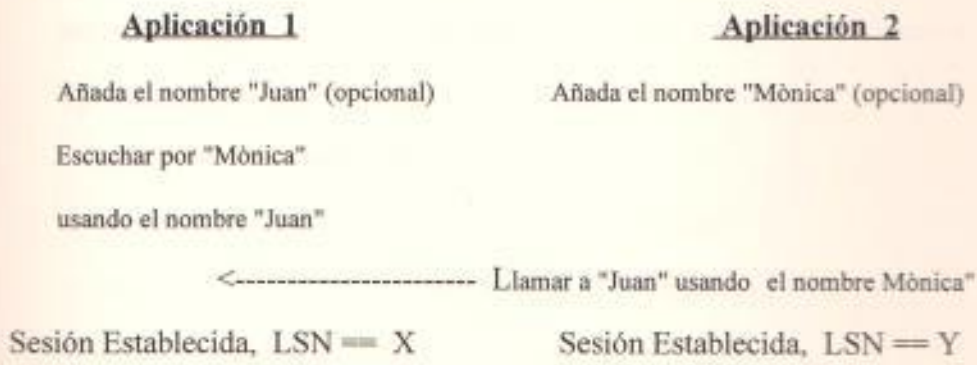


FIG. 3.3 Establecimiento de la Sesión

b) Características de los Comandos de Recepción

Después de establecer una sesión, ambos lados , cliente y servidor, pueden producir comandos NetBIOS ***"Send"*** (Enviar) y ***"Receive"*** (Recibir) para transferir los datos. Si un nombre dado es usado para crear varias funciones, una aplicación puede también producir un comando de NetBIOS ***"Receive-Any-For-Especified-Name"*** (Reciba-cualquiera-de-nombre-especificado), el cual proporciona los datos recibidos de cualquier sesión asociada con un nombre específico. Generalmente, la aplicación puede producir un comando ***"Receive-Any-for-Any-Name*** (Receive-Any-Any)" (Recibe-cualquiera-de Cualquier-Nombre), el cual proporciona los datos recibidos de cualquier sesión existente que el adaptador ha establecido activamente.

En el evento , el mensaje que arriba puede satisfacer mayormente que uno de estos tipos de recepciones NetBIOS, entonces se observa la siguiente jerarquía:

1.- Recibir (alta prioridad)

2.- Recibir-Cualquiera-de-un-Nombre-Especificado

3.- Recibir-Cualquiera-de-Cualquier-Nombre (baja prioridad)

c) Características del Comando "Send" (Enviar)

Las aplicaciones de NetBIOS producen comandos ***"Send"*** para transferir datos hacia otra aplicación. El comando "Send" permite a la aplicación enviar mensajes en rangos de tamaños desde cero bytes a 64K menos 1

byte de datos. El dato debe estar en memoria continua. La aplicación puede también producir un comando "Chain Send" (Cadena de Envío) que permite a los datos residir en buffers localizados en dos áreas diferentes de almacenamiento.

Con un comando ***"Chain Send"***, los datos dentro de cada buffer deben estar en memoria continua, sabiendo que los dos buffers en sí mismos no tienen que ser continuos. Más aún, cada bloque de datos puede tener un rango desde cero bytes a 64K menos 1 byte, permitiendo hasta 128K menos 2 bytes para transferir con un comando "Chain Send".

d) Consideraciones para Enviar y Recibir

Primero notemos que el comando "Chain Send" existe pero "Chain Receive" no existe. NetBIOS permite a las aplicaciones recibir parte de una transmisión y utilizar subsecuentes "Receives" para recibir el resto del mensaje. Esto es cierto para mensajes que se originan desde ambos comandos ***"Send"*** y ***"Receive"*** . Un solo comando "Receive" puede usualmente recibir mensajes transmitidos con un comando "Chain Send" proveyendo de un mensaje que no es tan largo. En cualquier evento, la aplicación de recepción no puede detectar si un mensaje fue transmitido con un "Send" versus "Chain Send" a menos que el tamaño total del mensaje exceda los 64K menos 1 byte. Esto es porque los datos del comando "Chain Send" que se originan en dos buffers separados siempre arriban de la misma manera sin indicación de las características del buffer original.

La única estipulación de una aplicación que parcialmente recibe un mensaje es que no retarde "mucho tiempo" para recibir el mensaje completo. Específicamente cuando la sesión es establecida, cada lado especifica los períodos de *"time-out"* del punto de partida de *"Send" y "Receive"*. Si el período del punto de partida de "Send" se excede antes que el mensaje sea completamente recibido, el "time-out" de "Send" y la sesión entera es terminada por el adaptador que está enviando. En este ejemplo, ambos lados de la sesión son notificados de las consecuencias del retardo del receptor.

e) Terminando las sesiones

Las sesiones son terminadas por uno o por ambos lados produciendo un comando ***"Hang Up"*** que especifica el LSN de la sesión para ser terminada. La otra aplicación es notificada sobre el final de la sesión cuando ésta utiliza un comando de sesión subsecuente. Una aplicación puede producir un comando que indicará el estado de la sesión: Existiendo o Cancelada.

El servicio de sesión consiste de:

1.- El protocolo

2.- El nombre de la Fuente.

3.- El número de la sesión fuente.

4.- Nombre del destino

5.- El número de la sesión destino.

Ejemplo:

{ Nbs, JOESXT, 4, PRINTER, 7}

COMANDOS GENERALES

Existen 4 comandos generales:

RESET	Reset NetBIOS.
CANCEL	Cancela un comando asincrónico.
ADAPTER_STATUS	Busca el estado del adaptador

El comando RESET limpia el nombre NetBIOS y las tablas de sesión y, además, sale de cualquier sesión existente.

El comando CANCEL asume que los comandos de NetBIOS pueden ser usados asincrónicamente por un proceso usuario. Esto es, el proceso del usuario inicia un comando pero no espera a que se complete. Se requiere de algún método para que el proceso sea notificado cuando el comando se completa o el proceso chequee si un comando específico es o no hecho.

El comando STATUS retorna el estado de una interfase_específica asociada con otro nombre local o remoto. Adicionalmente, retorna el nombre de la tabla para el nodo NetBIOS.

3.2 PROGRAMACION CON "NAMED-PIPES"

Named-Pipes es un mecanismo de IPC de comunicación bidireccional. Dos procesos pueden usar Named Pipes para efectuar intercambio de información, mientras un proceso escribe (envía datos), el otro lee (recibe datos), y viceversa. Bajo el sistema operativo OS/2, éste es es el único IPC que permite que los procesos de comunicación sean distribuídos en diferentes computadores de la red.

Named Pipes requiere de los dos componentes tradicionales: El **Servidor**, el cual "crea" el Named Pipe y el **Cliente** que se conecta al Servidor. El cliente debe conocer el nombre del pipe para conectarse. Cada lado obtiene un manejador (**"handle"**) del pipe. Con este "handle" se realizarán todas la operaciones de envío y recepción de datos a través del pipe.

El nombre del Named Pipe es su característica más importante. El cliente puede ser un proceso que está corriendo en una máquina diferente en la red. Este cliente debe conocer tanto la identificación del servidor (Machine Id) como el nombre del Pipe al cual desea conectarse.

¿ ES "NAMED PIPES" EL IPC IDEAL?

"Named Pipes" es el mecanismo de IPC más cercano al protocolo ideal de transferencia de datos entre procesos. Además, "Named Pipes" es el único mecanismo de IPC que puede transferir datos a través de la red. Su API de programación simula un archivo, usando un Named Pipe los procesos

pueden intercambiar datos como si leyeran o escribieran a un archivo secuencial.

"Named Pipes" son especialmente apropiados para la implementación de programas en el servidor, los cuales requieren de **"pipelines"** del tipo muchos-a-uno. Una aplicación en el servidor puede fijar un "pipeline" donde el receptor final del pipe pueda intercambiar datos con muchos procesos clientes, y tener el manejo de todos los planes de trabajo así como la sincronización de las producciones. Ellos además proveen de una plataforma muy robusta para el desarrollo de aplicaciones Cliente-Servidor. Sus principales características son:

1.- Un Método para el intercambio de datos y para el control de la información entre procesos que está corriendo en diferentes computadoras.

2.- Simulan la interface de red.

3.- Proveen de llamadas API , optimizadas, para la implementación de llamadas a procedimientos remotos, los cuales son un par de mensajes Requerimiento/Respuesta que son usados para construir aplicaciones Cliente-Servidor usando llamadas como en los procedimientos. La invocación de un procedimiento remoto daría como resultado un intercambio de Requerimiento/Respuesta sobre el Named Pipe". La solicitud que está llegando puede ser usada como salida hacia un procedimiento en el Servidor y pasar los parámetros requeridos. Los

resultados son retornados por el servidor en un mensaje de respuesta sobre el mismo "pipe"

CICLO DE VIDA DE UN NAMED PIPE

El named pipe es creado en el lado del Servidor por medio de ***DosMakeNmPipe***. El Servidor entonces, utiliza ***DosConnectNmPipe*** para esperar que el cliente se conecte al Pipe.

El cliente se conecta al extremo del Pipe utilizando la función **DosOpen**. Después de que ambos lados están conectados, ellos pueden empezar el diálogo con ***DosRead y DosWrite***.

Cuando el diálogo se completa, el servidor llama a ***DosDisconnectNmPipe***, y ejecuta una operación ***DosClose***. El Servidor puede entonces ejecutar otra llamada DosConnectNmPipe y esperar por el siguiente cliente. La figura 3.4 ilustra este ciclo.

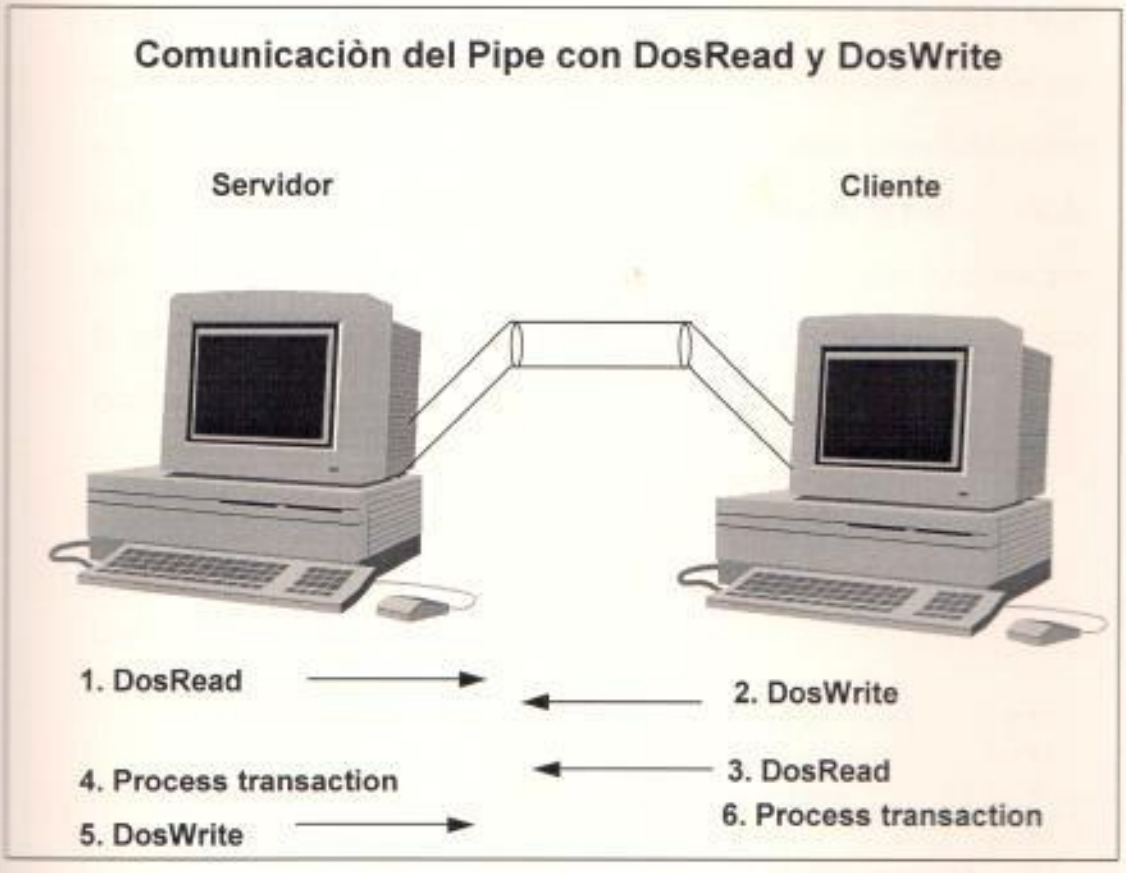


FIG. 3.4 EL CICLO DE VIDA DEL NAMED PIPE

USANDO "NAMED PIPES"

Para establecer un "Named Pipe, un proceso Servidor debe:

Primero Crear el "pipe" usando ***"DosMakeNmPipe"***, el cual retorna un manejador ("handle") que identifica el Pipe. Un "Named Pipe" es identificado por un nombre único . Los **Pipes locales** se nombran como: **\PIPE\name**, y son referenciados por ese nombre ("name") . Los **Pipes remotos** son referenciados como **\\server\PIPE\name**, donde el "server"

(servidor) es el nombre de la máquina servidora de la red. Un pipe local puede ser usado solamente por procesos de la misma máquina; mientras que un "pipe" remoto puede ser usado por procesos que están conectados sobre una red de área local. Esta ***capacidad de nombramiento de la red*** es una de las características más importantes de los "Named Pipes", esto significa que la aplicación Cliente puede comunicarse por medio del "Named Pipe" con cualquier máquina que se encuentre dentro de la red.

Los parámetros de la función ***DosMakeNmPipe*** son los siguientes:

```
unsigned far Pascal
DosMakeNmPipe(
    char far * name,          /* nombre del pipe                */
    unsigned far * handle,    /* retorna el manejador           */
    USHORT  omode,            /* modo de apertura               */
    USHORT  pmode,            /* modo del pipe                  */
    USHORT  size1,            /* tamaño del buffer de salida    */
    USHORT  size2,            /* tamaño del buffer de llegada   */
    long timeout              /*  timeout de DosWaitNmPipe      */
    )
```

Errores Retornados

Si DosMakeNmPipe es exitoso retorna cero (0); si no retorna uno de los siguientes códigos de error.

Error Retornado	Significado
ERROR_INVALID_PARAMETER	Uno de los argumentos contiene valores ilegales
ERROR_NOT_ENOUGH_MEMORY	Insuficiente memoria del sistema para crear buffers de pipes.
ERROR_OUT_OF_STRUCTURES	Insuficiente memoria del sistema para crear las estructuras internas usadas para Track named pipes
ERROR_PATH_NOT_FOUND	Invalido pathname del named pipe. Asegúrese que el pathname empieze con \PIPE\ y que contiene solamente caracteres legales del archivo del sistema
ERROR_PIPE_BUSY	Un named pipe con este nombre ya existe, y todas las instancias disponibles ya estan siendo usadas.

DosMakeNmPipe crea el Named Pipe. Si éste es usado en un ambiente distribuído, el pipe debe ser creado por el proceso servidor. El Named Pipe existe hasta que el proceso que lo ha creado existe.

El argumento **name** se parece al nombre del archivo del sistema. Debe empezar con **\PIPE**. El resto del nombre puede usar cualquier caracter válido del archivo del sistema, incluyendo "backslash" ("\").

Cada aplicación puede codificar el nombre de la misma en el Named Pipe. Todos los "named pipes" usados por programas bajo el servicio LAN MANAGER empiezan con \PIPE\LANMAN\. Por ejemplo, los named

pipes creados por el Servicio Netrun usan el nombre \PIPE\LANMAN\NETRUN.

Cuando el pipe es creado, se retorna un manejador del archivo en el parámetro *"handle"*. Las operaciones subsecuentes en el pipe, incluyéndo DosRead y DosWrite, lo usan para identificar el "pipe".

Si ocurren múltiples instancias del pipe, cada llamada a DosMakeNmPipe con el mismo nombre retorna un "handle" diferente, representando una instancia diferente del pipe.

El argumento **omode** es el modo abierto ("open"), un conjunto de bits que identifican el modo de acceso, una bandera de herencia, y una bandera "write-behind". Los valores para omode son los siguientes:

BITMASK	VALOR	SIGNIFICADO
NP_ACCESS_DUPLEX	0x0002	El pipe es bidireccional. Cliente y Servidor pueden leer y escribir el pipe
NP_ACCESS_INBOUND	0x0000	El pipe es unidireccional. El servidor puede leer y el cliente puede escribir.
NP_ACCESS_OUTBOUND	0x0001	El pipe es unidireccional. El servidor puede escribir y el cliente puede leer
NP_INHERIT	0x0000	El manejo del named-pipe debe ser heredado por el proceso hijo.
NP_NO_INHERIT	0x0080	El manejo del named pipe será heredado por el proces hijo.
NP_NOWRITEBEHIND	0X0000	Un retorno exitoso desde DosWrite indica que el dat está en el buffer del named pipe en el computador rem
NP_WRITEBEHIND	0x4000	En operaciones remotas un retorno exitoso a DosWrite puede ocurrir antes de que el dato sea transportado sobre la red.

El argumento **omode** puede especificar uno de los bitmasks de modo de acceso, uno de los bitmasks de herencia, y uno de "writebehind". Los valores por default son los siguientes:

NP_ACCESS_INBOUND | NP_INHERIT | NP_NOWRITEBEHIND.

Como mínimo, la mayoría de las aplicaciones debe especificar NP_ACCESS_DUPLEX. Notemos que la aplicación cliente debe abrir el named-pipe con un modo de archivo equivalente al modo de acceso usado en DosMakeNmPipe:

MODO ARCHIVO CLIENTE	MODO ARCHIVO SERVIDOR
OPEN_ACCESS_READONLY	NP_ACCESS_OUTBOUND
OPEN_ACCESS_READWRITE	NP_ACCESS_DUPLEX
OPEN_ACCESS_WRITEONLY	NP_ACCESS_INBOUND

El argumento **pmode** es un conjunto de bits que indican el contador de instancias, el modo de lectura, el tipo, y la bandera de espera. Los valores para pmode son los siguientes:

BITMASK	VALOR	SIGNIFICADO
NP_WAIT	0x0000	Las llamadas a DosRead no retornarán hasta que el dato esté disponible, y las llamadas a DosWrite no retornarán hasta que exista un "room" para el dato en el pipe. Esto es llamado bloqueo.
NP_NOWAIT	0x8000	DosRead y DosWrite retornarán inmediatamente si no pueden completar sus operaciones. Esto es el modo sin bloqueo.
NP_READMODE_BYTE	0X0000	El modo actual del pipe es leído en bytes.
NP_READMODE_MESSAGE	0x0100	El modo actual del pipe es leído en mensajes.
NP_TYPE_BYTE	0x0000	El pipe fue creado como flujo de bytes.
NP_TYPE_MESSAGE	0x0400	El pipe fue creado como flujo de mensajes.
NP_UNLIMITED_INSTANCES	0x00FF	Pueden existir un número ilimitado de instancias del pipe.

El argumento pmode debe especificar una de las opciones de espera ("wait"), una de las opciones de modo, una de las opciones tipo, y un contador de instancias. Si solo es especificado un contador de instancias (1, por ejemplo), por default, el resto de pmode son los siguientes:

NP_WAIT | NP_READMODE_BYTE | NP_TYPE_BYTE | 1

El contador de instancias puede ser desde 1 hasta 254 o el valor especial NP_UNLIMITED_INSTANCES. Un contador de instancias de 0 es un valor reservado y no debe usarse.

Los argumentos **size1 y size2** son avisos al Sistema Operativo acerca de los requerimientos de buffer para la salida y llegada de datos en el pipe. Consideremos una aplicación en la cual el cliente está haciendo escrituras de bloqueo en el pipe (esto es, cuando la opción NP_WAIT es fijada al cliente):

Cliente	**Servidor**
Write 2K	Read 1K. proceso
	Read 1K, proceso

Si el buffer que está llegando del servidor es más pequeño que 2KB, el proceso cliente debe bloquearse hasta que el servidor finalice su procesamiento. O consideremos un caso en el cual el buffer que está llegando es de una longitud de 128 bytes:

Cliente	**Servidor**
Write 2K	
	Read 2K.

El servidor leerá los primeros 128 bytes del buffer y dejará bloqueado al cliente, esperando a que finalice su escritura. Podría tomar 16 llamadas DosRead para todos los datos a ser transferidos. Si esto fue realizado en un ambiente remoto (el cliente y servidor en diferentes computadores), podría tomar 16 viajes por separado a través de la red. Si ambos argumentos, size1 y size2 son lo suficientemente grandes para que el dato pase a través de la

red, entonces, cualquier extremo puede escribir al named pipe y continuar procesando mientras el dato es almacenado en el named pipe.

El argumento **timeout** es usado cuando el proceso cliente usa el API DosWaitNmPipe. Este API permite que las aplicaciones esperen por la siguiente instancia del named pipe, hasta un período especificado de timeout. Si DosWaitNmPipe no especifica un timeout, usará el timeout de DosMakeNmPipe.

ESTABLECIENDO LA CONEXION

Una vez que el Pipe es creado en el servidor, éste se encuentra en el estado "Escuchar" ("listening") antes que un cliente pueda conectársele. El servidor hace esto usando ***"DosConnectNmPipe."*** El Pipe está ahora en el estado **"LISTENING"** esperando porque un proceso cliente lo abra. En este punto, cualquier proceso Cliente que conoce el completo calificativo del "path name" del "Named Pipe" puede abrir el "pipe" utilizando ***"DosOpen".***

Si "DosOpen" es exitoso, retorna el manejo del "pipe" y entonces pasa a un estado **"CONNECTED"** (conectado). Los parámetros de la función son los siguientes:

unsigned far pascal

DosConnectNmPipe(unsigned handle /* pipe handle */)

Errores Retornados:

Si es exitoso, DosConnectNmPipe retorna 0. Si no lo es, retorna uno de los siguientes códigos de error:

Errores Retornados	Significado
ERROR_INVALID_HANDLE	El argumento "handle" es inválido
ERROR_BAD_PIPE	El "handle" no es un "handle" del named-pipe
ERROR_INVALID_FUNCTION	El "handle" es para el cliente del named-pipe
ERROR_PIPE_NOT_CONNECTED	El pipe está en modo sin-bloqueo, y el cliente aún no ha ejecutado DosOpen.
ERROR_PIPE_BROKEN	El cliente final ha cerrado o está cerrando el pipe. El servidor final debe ejecutar una lla-mada DosDisconnectNmPipe antes de DosConnectNmPipe.
ERROR_INTERRUPT	DosConnectNmPipe fue interrumpido mientras estaba esperando por el cliente final para eje-cutar DosOpen.

DosConnectNmPipe es usado por el servidor del named pipe para determinar cuando el cliente se ha conectado al pipe con DosOpen. La labor de ésta llamada depende de si el pipe está en modo de bloqueo o sin-bloqueo.

Si la llamada falla , retorna ERROR_PIPE_BUSY, el cliente puede utilizar ***"DosWaitNmPipe"*** lo cual bloqueará el proceso hasta que la primera instancia del "pipe" esté disponible. *¿Qué pasa cuando muchos clientes están esperando por el mismo "pipe"?* el sistema operativo despertará al proceso que ha esperado por más tiempo, cuando una instancia del pipe esté disponible. El proceso desbloqueado debe entonces utilizar ***"DosOpen"*** para conectarse al Pipe.

Los parámetros de la función DosWaitNmPipe son los siguientes:

```
unsigned far pascal
DosWaitNmPipe(
        char   far *name;       /* nombre del pipe          */
        long   timeout          /*  maximo tiempo de espera */
        )
```

Errores Retornados:

Si es exitoso, DosWaitNmPipe retorna 0. Si no lo es , retorna uno de los siguientes códigos de error:

Error Retornado	Significado

ERROR_FILE_NOT_FOUND	El "named-pipe" no existe
ERROR_SEM_TIMEOUT	No hay una instancia disponible del named-pipe para abrir dentro de milisegundos del timeout.
ERROR_INTERRUPT	Una interrupción ocurrió en la mitad de una espera por una instancia del named pipe.

DosWaitNamePipe debe ser llamado por una aplicación del cliente cuando DosOpen retorna ERROR_PIPE_BUSY. Existe una condición posible de contienda entre el tiempo que retorna DosWaitNmPipe y el tiempo que una nueva llamada DosOpen sea hecha. El siguiente fragmento de código muestra cómo usar DosWaitNmPipe en un programa:

```
err = DosOpen( pipename, .
while (err == ERROR_PIPE_BUSY)
        {
        err = DosWaitNmPipe( pipename, timeout);
        if (err != 0)
        break;  /*  inserte el código para procesar el error */
        err = DosOpen (pipename, .....
        }
```

INTERCAMBIO DE INFORMACION

Una vez que el "pipe" está en estado **"CONNECTED"** (CONECTADO), los procesos servidor y cliente pueden usar las llamadas a archivos del subsistema de OS/2, para comunicarse sobre el como sigue:

- **DosWrite** o **DosWriteAsync** son usados para escribir el dato en el "pipe".
- **DosRead** o **DosReadAsync** son usados para leer el dato del "pipe".
- **DosBufReset** es usado para sincronizar los diálogos de lectura y escritura. Hace esto por el bloqueo del proceso que se está llamando en un extremo del "pipe" hasta que todos los datos que ha escrito hayan sido leídos en el otro extremo del "pipe".

PROCESAMIENTO DE LA TRANSACCION USANDO "Named Pipes"

"Named Pipes soporta 2 funciones para el procesamiento de una transacción: **DosTransactNmPipe** y **DosCallNmPipe.**

Estas llamadas pueden solo ser usadas en Pipes tipo **duplex**, es decir, que permite tanto el envío como la recepción de mensajes.

DosTransactNmPipe: Esta llamada API envía un mensaje y recibe una respuesta en una sola operación (ver FIG. 3.5). El par Requerimiento/Respuesta es usado para inicializar una transacción ó una

Llamada al Procedimiento Remoto. Esta llamada es típicamente utilizada sobre conexiones **"persistentes"**. En este tipo de conexión, muchas transacciones son usualmente utilizadas sobre un Pipe abierto antes de que termine la conexión.

La conexión persistente es apropiada para **pipes remotos** , debido a su alto costo para establecer las conexiones. A continuación mostraremos el siguiente ejemplo:

```
unsigned  far pascal
DosTransactNmPipe{
        unsigned      handle;        /* manejador del pipe*/
        char          far * inbuf;   /* buffer de datos de entrada */
        USHORT        inlen;         /* longitud de datos de entrada */
        char far      *outlen;       /* buffer de datos de salida*/
        USHORT        outlen;        /* longitud del buffer de salida*/
        USHORT far    *bread;        /* bytes leìdos */
        }
```

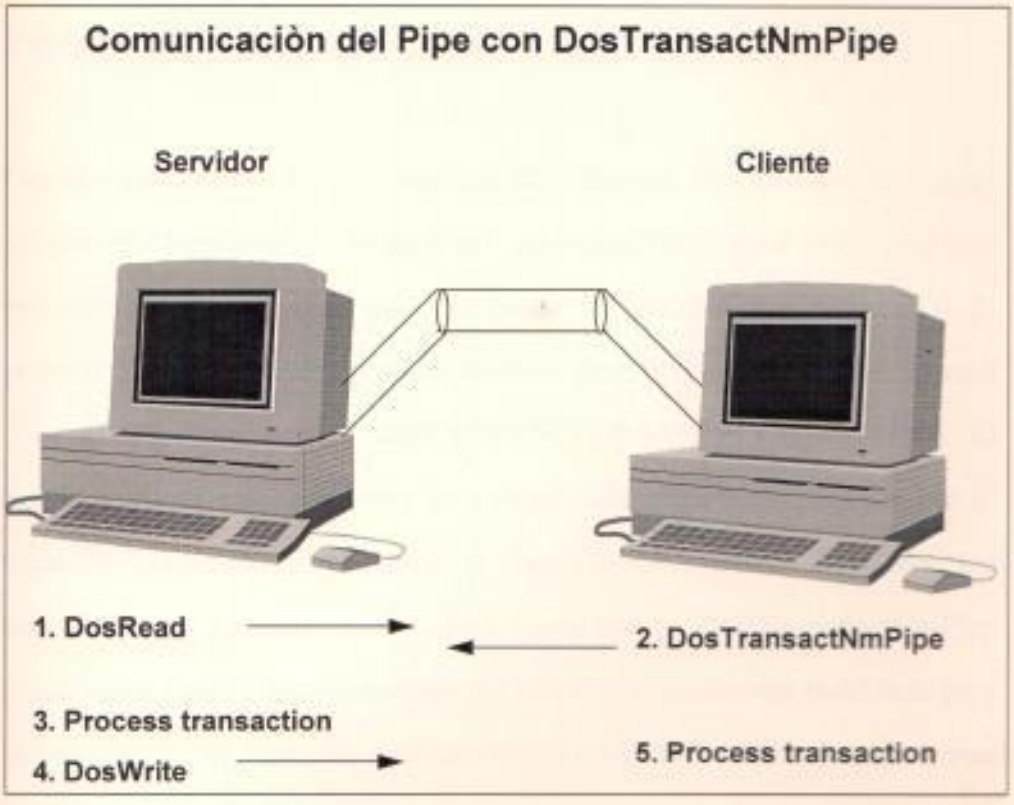


FIG. 3.5 COMUNICACION DEL PIPE CON DosTransactNmPipe

DosCallNmPipe: Esta llamada API combina **DosOpen**, **DosTransactNmPipe**, y **DosClose** en una sola operación. Es usada en **conexiones no-persistentes** que se han establecido y terminado solamente para utilizar una sola llamada a un Procedimiento Remoto. Las conexiones no-persistentes son ideales para simples transacciones Solicitud/Respuesta.

TERMINANDO UN DIALOGO DE "Named Pipe"

Cuando un proceso Cliente está usando "Named Pipe", cierra el "pipe" utilizando ***"DosClose"***. La cual es usada por OS/2 para cerrar archivos regulares. El "pipe" está ahora en el estado **"CLOSING"** (CERRADO). El proceso Servidor puede entonces utilizar ***DosDisConnectNmPipe***, el cual pone al "pipe" en estado **"DISCONNECTED"** (DESCONECTADO). El Servidor puede entonces usar otro ***DosConnectNmPipe*** y esperar por el siguiente cliente ,o puede cerrar el "Named Pipe" usando ***DosClose***. Un servidor puede también utilizar Dos Close sin un DosDisconnectNmPipe precedente. Esto liberará el manejo del "pipe" y lo mantendrá habilitado para que el cliente lea cualquier dato que queda en el buffer. De tal manera que, un "pipe" cerrado no puede usarse nuevamente sin utilizar nuevamente ***DosMakeNmPipe***.

Si el Pipe es abierto cuando DosDisConnectNmPipe es usado, él es forzado a cerrar y el cliente obtiene un código de error en la siguiente operación. El forzar al cliente final para que se cierre prematuramente, puede causar que el dato sea descartado sin haber sido aún leído por el cliente.

Un cliente que es forzado a apagar el "pipe" con DosDisConnectNmPipe debe utilizar DosClose para liberar el manejo del recurso.
DosDisconnectNmPipe invalida el manejo del cliente , debido a que no libera el recurso. Cualquier "thread" que es bloqueado en el "pipe" son despertados

por ***DosDisconnectNmPipe*** . Un "thread" bloqueado en el "pipe" por DosWrite retorna ERROR_BROKEN_PIPE. Un "thread" bloqueado en el pipe por DosRead, retorna EOF. En general, las desconexiones forzadas deben ser destruidas, proveyendo de un protocolo con el usuario para apagar el "pipe".

Usando Semáforos con "Named Pipes"

Los semáforos del sistema pueden ser usados en unión con un "Named Pipe" local para controlar el acceso al "pipe". Un proceso de lectura puede usar **DosSemWait o DosMuxSemWait** para esperar a que el dato arribe en uno o más "pipes". Esto destruye muchos métodos costosos, dedicando un **"thread"** (hilo) para cada "pipe" que está llegando o consultando ("polling") a cada "pipe" con un modo de no-espera.

La llamada **DosSetNmPipe** es usado para asociar un semáforo del sistema con una instancia del "Named Pipe" local. Si el intento es hecho para conectar un semáforo hacia un **pipe remoto**, retorna **"ERROR_INVALID_FUNCTION"** . Hasta dos semáforos pueden ser conectados a un "pipe", uno para el servidor y otro para el cliente. Si está listo uno de los semáforos asociados con un extremo del "pipe", el antiguo semáforo es reemplazado. OS/2 limpiará el semáforo cuando el nuevo dato está disponible, siempre y cuando el espacio del buffer esté disponible para escritura, o si cualquiera de los dos lados cierra el "pipe". Una aplicación es

notificada de este evento a través de llamadas a semáforos regulares (Ejemplos: DosSemRequest ó DosMuxSemWait). Se puede asociar un semáforo con más de un "pipe" manejable. Un parámetro identifica cual "pipe" maneja lo justificado.

Un proceso puede chequear el estado de los semáforos "Named Pipe" utilizando **DosQNmPipeSemState**. Esta llamada retorna la información acerca de los "Named Pipes" locales para especificar un semáforo del sistema, así como la infomación adicional acerca de I/O (Entrada/Salida) que pueden ser ejecutados en el conjunto de "pipes". Esta información puede ser buscada para determinar cuales "pipes" pueden leerse o escribirse. Además, esta llamada proporciona toda la información acerca del evento que ha aclarado un semáforo asociado con el manejo de un "pipe" en particular. (a travéz de DosSetNmPipeSem).

Ejemplo:

unsigned far pascal

```
DosQNmPipeSemState (
        long  semhandle,        /* manejador del semàforo *        */
        char  far * infobuf,    /* estructura _PIPESEMSTATE        */
        USHORT  infobuflen,     /* longitud de informac. del buffer */
        )
```

Obteniendo y Modificando la Información del Pipe

Un proceso puede observar el contenido de un "Named Pipe", pero sin removerlo, utilizando **DosPeekNmPipe**. Esta llamada también retorna el estado de un "pipe" . Por ejemplo:

```
unsigned far pascal
DosPeekNmPipe(
        unsigned handle;        /* manejador del pipe      */
        char far *buffer;       /* buffer de datos         */
        USHORT buflen;          /* longitud del buffer     */
        USHORT far *bread;      /* bytes leidos            */
        USHORT far *bavail;     /* bytes disponibles       */
        USHORT far *status;     /* estado del pipe         */
        )
```

Errores retornados:

Si es exitoso, DosPeekNmPipe retorna 0. Si no lo es, retorna uno de los siguientes códigos de error:

Error Retornado	Significado
ERROR_INVALID_HANDLE	El argumento handle es inválido
ERROR_BAD_PIPE	El handle no es un handle del named pipe
ERROR_PIPE_NOT_CONNECTED	El cliente no ha sido forzado a apagarse con DosDisconnectNmPipe.
ERROR_PIPE_BUSY	Otro thread está leyendo o escribiendo el pipe concurrentemente

DosPeekNmPipe nunca bloqueará, a menos que esté en modo de bloqueo.

Un proceso puede obtener información general acerca del "pipe" utilizando ***DosQNmPipeInfo***. La información retornada incluye: la entrada y salida de los tamaños del buffer, el número máximo de instancias de "pipes" permitidas, el número actual de instancias del "pipe", y el nombre del "pipe" incluyendo el nombre del computador servidor, si es que el Pipe es **remoto**.

Utilizando ***DosQNmPHandState***, un proceso puede obtener información del estado actual del "pipe", el número actual de **instancias** (o solicitudes) del "pipe", su **dirección**, y el **tipo**.

Por ejemplo:

```
unsigned far pascal
DosQNmHandState{
        unsigned    handle;          /* manejador del pipe*/
        USHORT      far *pmode       /*ubicaciòn del modo del pipe
        */

        }
```

Errores retornados:

Si es exitoso, DosQNmHandState retorna 0; si no, retorna uno de los siguientes códigos de error:

Error Retornado	Significado
ERROR_INVALID_HANDLE	El argumento handle es invalido
ERROR_BAD_PIPE	Handle no es un handle del named pipe
ERROR_INVALID_FUNCTION	El handle es para el l cliente del named pipe

Un proceso puede modificar el estado del "pipe" a través de la llamada **DosSetNmPHandState.** Esta llamada es utilizada para cambiar el modo del

"pipe" de un flujo de bytes a flujo de mensaje o viceversa, y cambiar el modo wait-no/no-wait (esperar-no/no-esperar). Esta llamada sería usada por clientes para modificar el DosOpen que es por "default", el cual siempre abre los "Named Pipes" con el modo de "wait" (esperar) habilitado y fija el "pipe" en modo de flujo. A continuación tenemos el siguiente ejemplo:

DosSetNmPHandState fija el modo del named-pipe

```
unsigned far pascal
DosSetNmPHandState {
        unsigned handle;        /* manejador del pipe */
        USHORT pmode;           /* nuevo modo del pipe*/
}
```

Errores Retornados:

Si este API es exitoso, retorna 0; si no, retorna uno de los siguientes códigos de error:

Error Retornado	Significado
ERROR_INVALID_HANDLE	El argumento handle es inválido
ERROR_BAD_PIPE	El handle no es del named pipe
ERROR_PIPE_NOT_CONNECTED	El cliente no ha sido forzado a apagarse con DosDisconnectNmPipe.
ERROR_INVALID_FUNCTION	El argumento pmode trató de cambiar el modo del pipe dentro del modo de lectura-mensaje.

DosSetNmPHandState cambia algunos parámetros del modo del pipe. Solamente el modo de lectura y el estado de bloqueo/sin-bloqueo pueden ser cambiados.

Para cambiar el modo de lectura ("read"), usamos las siguientes llamadas a las funciones:

DosQNmPHandState(hdl, &pmode);

hdl: manejador del pipe, de tipo unsigned.

DosSetNmPHandState(hdl, (pmode &NP_NOWAIT) |

NP_READMODE_MESSAGE);

Usamos (pmode &NP_READMODE_MESSAGE) para preservar el modo de lectura existente.

ANATOMIA DEL PROGRAMA CON NAMED PIPES

Esta sección resume como son utilizados los "Named Pipes" describiéndo la anatomía de un típico programa:

1.- **Crear el "Named Pipe"**. El Servidor de una aplicación crea un "Named Pipe" utilizando ***DosMakeNmPipe***. , este parámetro de instancias especifica cuántos clientes pueden usar el "pipe" concurrentemente.

2.-**Esperar hasta que los clientes se conecten**: La aplicación crea tantos "threads" como número de clientes simultáneos quieran servirse. Cada "thread" utiliza ***DosConnectNmPipe*** ,se bloquea esperando por un cliente para conectarse al otro extremo de la instancia del "pipe".

3.-**Estableciendo la conexión con los clientes**: Los clientes se conectan a su extremo del "pipe" utilizando ***DosOpen ó DosCallNmPipe*** (el cual ejecuta el DosOpen automáticamente). Cada DosOpen exitoso retorna un manejador exclusivo para el "pipe". En el lado del "servidor", DosConnectNmPipe tiene éxito.

4.-**Manteniendo la comunicación**: Después de que ambos lados del "pipe" están conectados, pueden empezar a negociar usando uno de los dos estilos de diálogos: ***Manteniendo una conexión persistente del "pipe",*** aquí el cliente y el servidor conducen múltiples intercambios de mensajes sobre el "pipe", esto es como llevar una larga conversación sobre el

telèfono. Un conversaciòn tipicamente usa llamadas como *DosWrite, DosRead y DosTransactNmPipe..*, intercambian Solicitud/Respuesta después de que sus respectivas partes se desconecten de sus extremos del "pipe". Un diálogo transaccional es conducido usando llamadas como ***DosCallNmPipe*** en el lado del cliente y ***DosTransactNmPipe*** en el lado del servidor.

Los ***"Pipes" No-persistentes*** tienen que pagar un costo que es el tiempo de "setup", pero el beneficio es que la instancia del "pipe" puede ser reusada serialmente por otros clientes. Esto es muy usado en situaciones donde un servidor puede requerir hablar a cientos de clientes.

5.- **Desconectando y tomando la siguiente llamada**: Cuando el negocio se completa, el servidor del "pipe" utiliza ***DosDisconnectNmPipe*** y el cliente utiliza ***DosClose*** (DosCallNmPipe ejecuta automáticamente DosClose). El servidor puede entonces ejecutar otro DosConnectNmPipe y esperar por la solicitud del siguiente cliente.

6.- **Terminando el "pipe"**. Cuando el "thread" servidor quiere parar de aceptar las llamadas de clientes, utiliza ***DosClose*** para borrar esa instancia del "Named Pipe". Cuando la última instancia es cerrada, el "pipe" deja de existir.

Esquema de Funcionamiento del Servidor de Consultas de Clientes para Plataforma

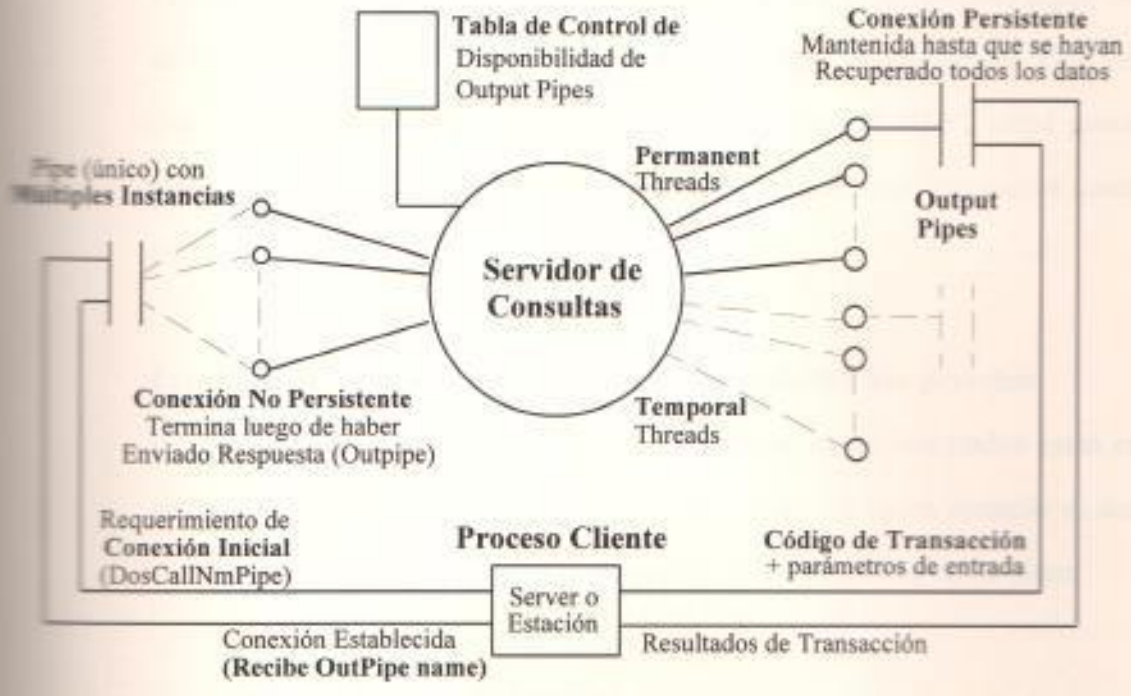


FIG 3.6 ESQUEMA DEL FUNCIONAMIENTO DE LA PLATAFORMA CLIENTE-SERVIDOR

ESCENARIOS DE NAMED PIPES

Los escenarios que siguen ayudarán a demostrar el uso de "Named pipes" en ambientes que requieren intercambios de datos entre interprocesos. Una vez que se apodere de los mecanismos del uso de "Named Pipe", usted puede crear su propio repertorio de secuencias API usando sus escenarios como punto de inicio.

Escenario 1: Una Transacción Simple que envuelve dos procesos

El escenario 1 consiste de una simple transacción cuyos resultados están en un sólo intercambio de datos Solicitud/Respuesta. Un típico ejemplo es una simple transacción de banco para preguntar sobre su cuenta como sigue:

Cuál es el Balance de Roberto?

<---------- $300

Implementamos esta simple transacción de tres formas diferentes:

- La primera implementación del Escenario 1 usa exclusivamente llamadas a "Named Pipe" orientadas a archivos.

- La segunda implementación del Escenario 1 refina al cliente introduciendo el uso de **DosTransactNmPipe** (Ver FIG. 3.7)

- La tercera implementación del Escenario 1 es la más elegante y económica forma para diseñar simples diálogos transaccionales. DosCallNmPipe fue diseñado específicamente para un solo intercambio

Solicitud/Respuesta sobre una conexión no-persistente. Esta llamada algunas veces es referenciada hacia la literatura de OS/2 como una Llamada a Procedimiento Remoto. Lo que signifiica esto es que DosCallNmPipe puede ser usada para intercambios solicitud/respuesta usando la semántica de la llamada a un procedimiento. Para obtener la cuenta de Bob, debería llamar a un fragmento del procedimiento local con "Bob" como argumento. El fragmento local, en turno, crea un mensaje y utiliza DosCallNmPipe. En el lado del servidor, el arribo del mensaje da como resultado la invocación del procedimiento del servidor que calcula el balance de Bob y retorna el resultado: $300. El resultado es retornado al fragmento del cliente sobre el "Named Pipe". El fragmento del cliente no se bloquea y retorna el resultado al que lo llama. El efecto de esta Llamada a Procedimientos Remotos es introducir semánticas parecidas a los procedimientos para la comunicación de interprocesos. Desde el punto de vista de "Named Pipes", no hay manera de que la interfase sea presentada al usuario o que el dato haya pasado en el "pipe". El "pipe" es simplemente un mecanismo deliberadamente super-eficiente que puede ser usado ya sea "novato" o a través de formas de presentación como Llamadas a Procedimientos Remotos.

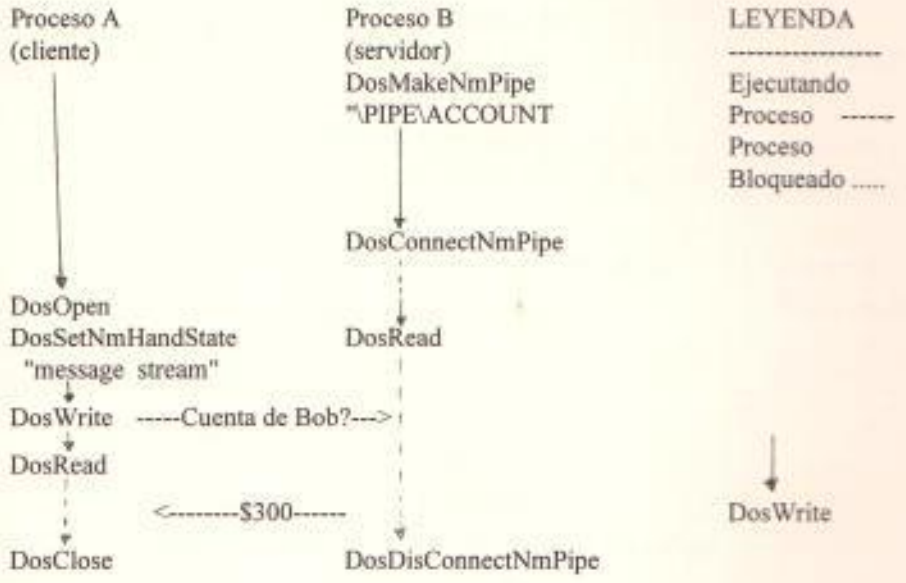


FIG. 3.7 Escenario 1a: Una simple Transacción Usando Llamadas a Named Pipes de Servicios de Archivos

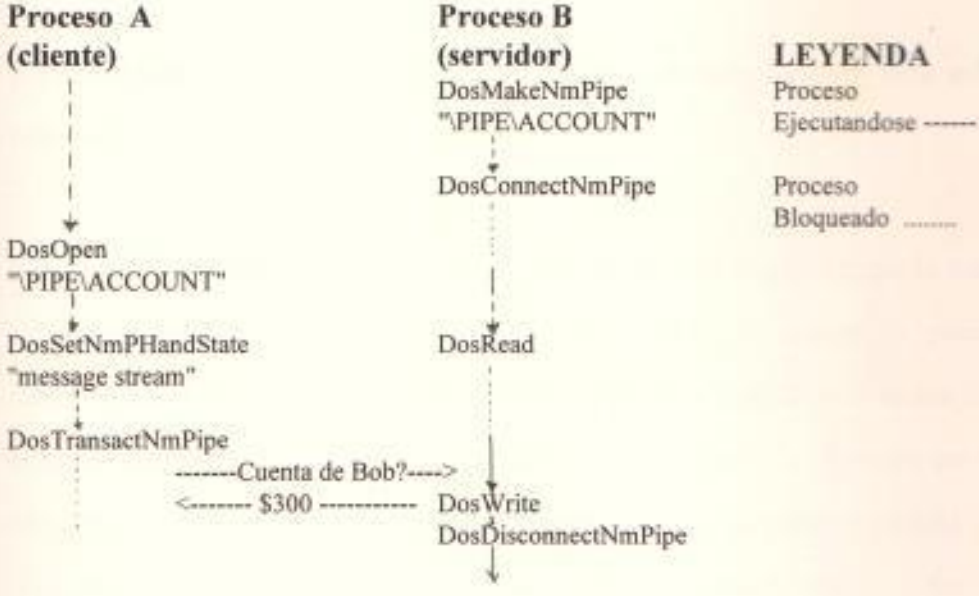


FIG. 3.8 Escenario 1b: Una Simple Transacción Usando DosTransactNmPipe

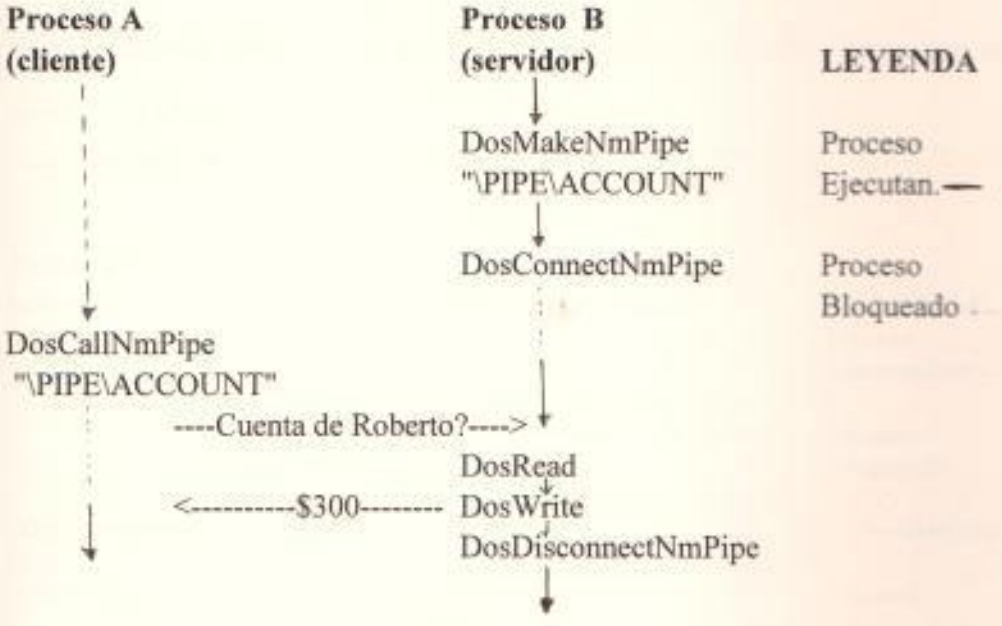


FIG. 3.9 Escenario 1c: Una Transacción Simple Usando DosCallNmPipe

ESCENARIO 2: Un Servidor con mùltiples clientes usando una sola instancia del pipe.

El Escenario 2 muestra còmo una sola instancia del pipe puede ser usada para servir serialmente a dos clientes (ver FIG. 3.10). El escenario puede fácilmente ser expandido para "n" clientes. Podemos asumir que todos los clientes estarían ejecutando simples transacciones, como la descrita en el escenario 1. El punto clave en el escenario es como el servidor reconecta el "Named Pipe" para permitir que otros clientes puedan usarlo. En el Escenario 2, el servidor sirve al proceso 1 seguido por el proceso 2. Asumimos que los valores de "time-out" en **DosCallNmPipe** son lo

suficientemente grandes para mantener un proceso bloqueado hasta que el servidor lo libere.

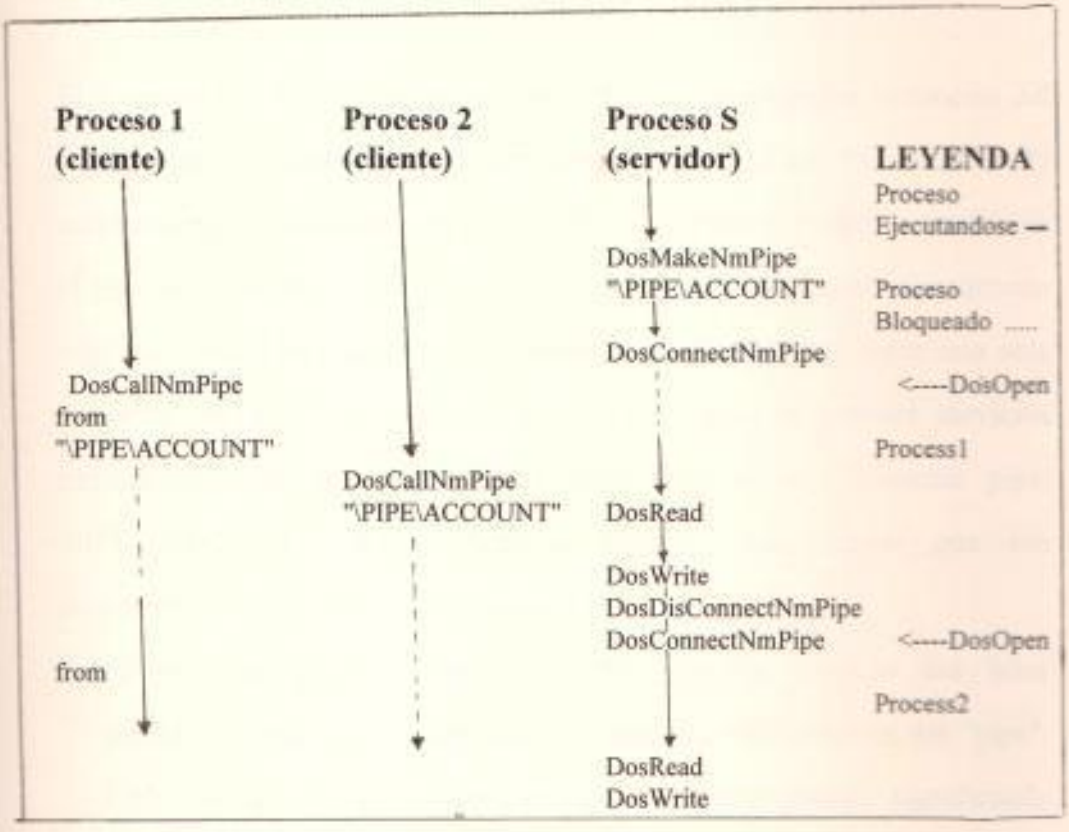


FIG. 3.10 Escenario 2: Rehusando Serialmente una Instancia de "Named Pipe"

ESCENARIO 3: Un Servidor "Multithread" usando mùltiples instancias del pipe

El Escenario 3, muestra como un solo "pipe" con múltiples instancias del pipe pueden ser usadas para acomodar dos clientes (Ver FIG. 3.12). El escenario puede fácilmente ser extendido a "n" clientes. Podemos asumir en el escenario que las transacciones del cliente son persistentes, significando esto que múltiples transacciones simples serían utilizadas sobre una sola conexión del "pipe". El servidor debería ser capaz de proveer servicios simultáneos para solo hasta dos clientes sobre el bien conocido pipe: \PIPE\ACCOUNT. Aquí tenemos algunas observaciones que son pertinentes en cómo juega este escenario:

1.- El escenario empieza con el proceso servidor creando dos hilos servidores, cada uno de los cuales manejaría una instancia del "pipe". Cada instancia sería usada para servir a un proceso cliente, significando esto que hasta dos clientes pueden abrir simultáneamente \PIPE\ACCOUNT.

2.- Cada hilo servidor utiliza DosMakeNmPipe para crear el "pipe": \PIPE\ACCOUNT con dos instancias especificadas. OS/2 retorna un manejo separado para cada instancia. El "thread" servidor usaría el manejo retornado para manejar su instancia del "pipe" y conectar al cliente a través de **DosConnectNmPipe**.

3.- En el Escenario 3, todos los "threads" se ejecutan concurrentemente. En cualquier momento, sólo un hilo está corriendo y sólo un mensaje está yendo a través del "pipe". OS/2 trataría de hacer un DosOpen, pero hay que considerar que solamente soportará hasta 53 "threads" por proceso. Para sistemas que necesitan soportar un gran número de clientes, se podría querer implementar un piscina de "threads" servidores no-persistentes además de la piscina de "threads" servidores persistentes .

4.- Un hilo no-persistente es similar al "thread" servidor del Escenario 2. El conduce la comunicación con el cliente y se desconecta tan pronto como sea capaz de hacerlo.

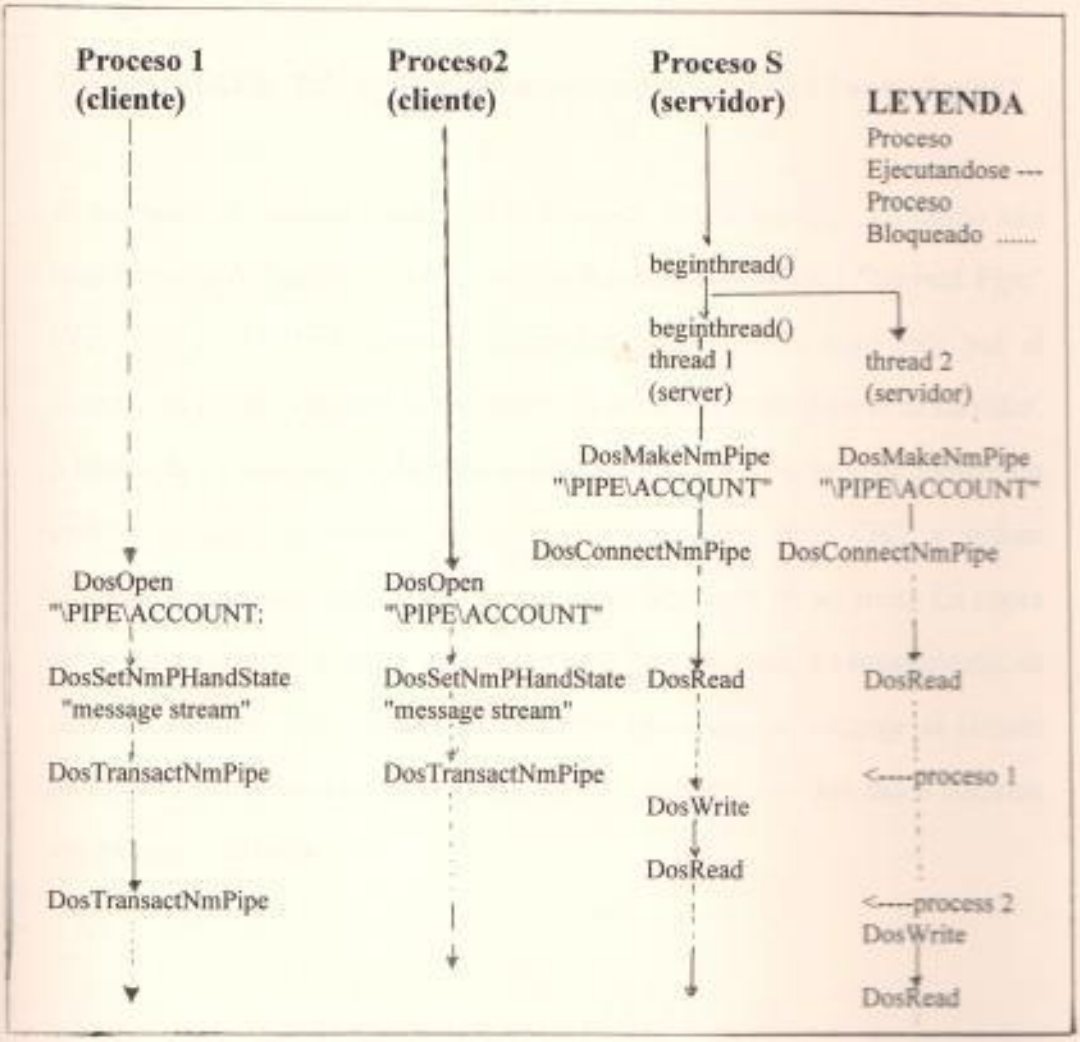


FIG. 3.11. ESCENARIO 3: Un Servidor Multhread usando Named Pipe con 2 instancias

ESCENARIO 4: Enviando datos masivamente sobre un "named pipe"

El escenario 4 muestra como mover datos en forma masiva , tal como una transferencia de archivos, puede ser implementada sobre un "Named Pipe" (Ver FIG. 3.13.). Inicialmente, el servidor es bloqueado esperando por el cliente. El cliente empieza la transferencia del archivo diciéndole al servidor, a través de la transacción **DosTransactNmPipe** que es acerca de recibir un archivo y darle un nombre y tamaño. El servidor dice "OK" y utiliza **DosRead** esperando por que arribe el primer segmento de archivo. La copia del archivo se hace a través de una serie de intercambios. Eventualmente, el servidor finaliza consumiendo el archivo y enviando un mensaje al cliente diciéndole que el archivo obtenido aquí OK después sus dos lados rompen sus extremos del pipe.

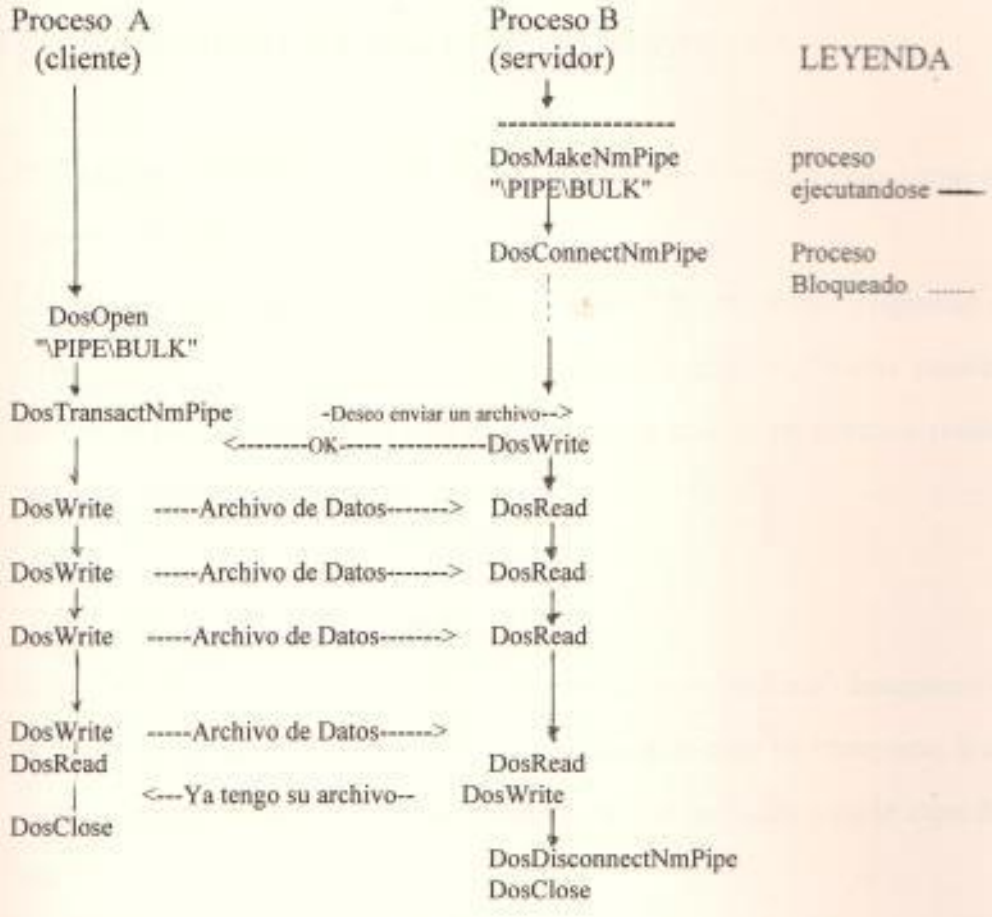


FIG. 3.12 Escenario 4: Moviendo la Capacidad de los Datos sobre un Named Pipe

3.3. PROGRAMACION CON "SOCKETS" DE TCP/IP

El protocolo TCP/IP provee un API de programación conocido como la "Interface Socket".

Los sockets permiten a un programa comunicarse con otros programas a través de las redes, por ejemplo, usted puede usar estas rutinas cuando escriba un programa cliente que debe comunicarse con un programa servidor que está corriendo en otro computador.

TCP/IP soporta 3 tipos de "sockets" API:

Flujo ("stream"), Datagrama, y Natural ("raw"). Los "sockets" datagrama y "stream" hacen una interfase con los protocolos de la capa de transporte, y el "socket" natural ("raw") hace una interfase con los protocolos de la capa de red.

Conceptos de Programación "Socket".

Antes de la programación debemos considerar algunos conceptos importantes:

Concepto de "socket".

Un "socket" es un punto de entrada para la comunicación. Es representado por un número entero llamado, descriptor del "socket".

Tipos de "Sockets".

La interface del "socket" ("SOCK_STREAM") tipo flujo define un servicio formal orientado-a-conexión. El dato es enviado sin errores o duplicaciones y es recibido en el mismo orden que es enviado. La interfase del "socket" tipo datagrama ("SOCK_DGRAM") define un servicio de conexión menor ó sin-conexión. Los datagramas son enviados como paquetes independientes. El servicio no provee de garantías; los datos pueden perderse o duplicarse, y los datagramas pueden arribar fuera de orden. El tamaño de un datagrama está limitado al espacio que puede enviarse en una sola transacción.

La interfase del "socket" tipo "raw" (SOCK_RAW) permite un acceso directo a protocolos de la capa más baja como con IP y el Protocolo de Control de Mensajes por Internet (ICMP). Esta interfase es frecuentemente usada para probar nuevas implementaciones de protocolos o ganar acceso a algunas facilidades más avanzadas de un protocolo existente.

Los tipos de "sockets" son definidos en el archivo de cabecera <SYS\SOCKET.H>.

La interfase de socket puede ser extendida; es decir, que puede definir nuevos tipos de sockets para proveer de servicios adicionales. "Sockets" tipo transacción, no son soportados por TCP/IP para OS/2.

GUIA PARA EL USO DE LOS TIPOS DE "SOCKETS"

Las siguientes consideraciones nos ayudarán a escoger el tipo de "socket" apropiado según la aplicación que se va a utilizar.

Si se está comunicándo con una aplicación existente, debe usarse el mismo protocolo de ella, por ejemplo, si se hace una interfase con una aplicación que usa TCP, se debe usar "sockets" de tipo "stream" (flujo). Para otras aplicaciones, se debe considerar los siguientes factores:

- **Formalidad:** sockets de tipo "stream" proveen la conexión más formal, ya que Datagrama o "raw" son informales, porque los paquetes pueden despacharse incorrectamente, o duplicarse durante la transmisión, esto puede ser aplicable si la aplicación no requiere formalidad.

- **Rendimiento**: La cabecera asociada con formalidad, control de flujo, reensamblaje de paquetes, y el mantenimiento de la conexión degradan el rendimiento del "socket" tipo flujo debido a que ellos no rinden tan bien como con los de tipo datagrama.

- **Cantidad de datos a transferirse**: Los "sockets" de tipo datagrama imponen un límite en la cantidad de datos que se van a transferir. Si usted envía menos de 2048 bytes al mismo tiempo, debe usar "sockets" tipo datagrama. A medida que la cantidad de datos en una sola transacción aumenta se hace más sensato el uso de "sockets" tipo flujo.

Si se está escribiéndo un nuevo protocolo en el tope de IP, o se desea usar el protocolo ICMP, entonces se debe escoger "sockets" tipo "raw".

- **Familias de Direcciones:** Las familias de direcciones definen estilos de direcciones o dominio de comunicaciones . Todos los "hosts" de la misma familia de direcciones usan el mismo esquema de direcciones de los puntos finales de los sockets TCP/IP para OS/2, soporta una familia de dirección: AFINET, cuyo dominio define la dirección en el dominio Internet. AFINET es referido también como PF_INET. Ambos son equivalentes.

  Las familias de direcciones están definidas en el archivo de cabecera <SYS\SOCKET.H>

- **Dirección del "Socket":**

  Una dirección del "socket" está definida por la estructura **sockaddr** en el archivo de cabecera <SYS\SOCKET.H>. Tiene dos campos, como muestra el siguiente ejemplo:

```
struct sockaddr
{
    u_short      sa_family;      /*dirección de la familia*/
    char         sa_data[14];    /*hasta 14 bytes de dirección
directa*/
};
```

El campo sa_family contiene la dirección de la familia . Cada dirección de la familia define su propia estructura, la cual puede ser colocada sobre la estructura sockaddr.

Direccionamiento dentro del dominio de Internet

Una dirección del "socket" en una dirección Internet comprende 4 campos:

1) La dirección de la familia (AF_NET),

2) Una dirección de Internet,

3) Un puerto, y,

4) Un arreglo de caracteres.

La estructura de la dirección de un "socket" de internet está definida por la estructura sockaddr_in, la cual se encuentra en el archivo de cabecera <NETINET\IN.H>:

```
struct in_addr
{
        u_long s_addr;
};
struct sockaddr_in
{
      short  sin_family;
      u_short   sin_port;
```

```
    struct  in_addr sin_addr;
    char    sin_zero[8];
};
```

El **campo sin_family** es fijado a AF_INET. El **campo sin_port** es el puerto usado por la aplicación, en orden de bytes de la red. El campo **sin_addr** es la dirección de internet de la interface de red usada por la aplicación. Además también va en orden de bytes de la reds. El campo sin_zero debe fijarse a cero.

2. DIRECCION DE INTERNET

Las direcciones de internet tienen cantidades de 32-bits que representan la interface de red. Toda dirección de internet dentro de un dominio administrado AF_INET debe ser única. Un acuerdo en común es que todo hostal debe tener solamente una dirección de internet. En efecto, un "host" tiene muchas direcciones Internet así como interfaces de red.

3. PUERTOS

Un puerto es usado para diferenciar las diversas aplicaciones que están usando el mismo protocolo (TCP o UDP). Además es una cantidad adicional (fisicamente un entero de 16 bits), usada por el software del

sistema, para obtener el dato para la correcta aplicación. Algunos de ellos son reservados para aplicaciones particulares y son llamados puertos "bien-conocidos".

4. "Network Byte Order"

Los puertos y direcciones son usualmente especificados para las llamadas usando el byte de ordenamiento de la red. Este es también conocido como el "gran indio" de ordenamiento de bytes. Esta técnica permite usar diferentes arquitecturas en el intercambio de información (o datos) entre hostales.

La interfase de sockets no maneja aplicaciones acerca de la diferencias del ordenamiento de bytes de datos. Los escritores de las aplicaciones deben manejar estas diferencias por ellos mismos o usar interfases de alto nivel, como Llamadas a procedimientos Remotos (RPC) o Sistemas para Redes de Computadoras (NCS).

LLAMADAS AL "SOCKET" PRINCIPAL

Con llamadas a pocos "sockets", se puede escribir una muy poderosa aplicación de la red.

1.- Primero, una aplicación debe ser inicializada con "sockets" utilizando la llamada **sock_init()**.

Para mayor información ver en el **Apéndice A**.

```
int rc;
int sock_init();
.
.
rc = sock_init();
```

FIG. 3.13 APLICACION USANDO UNA LLAMADA SOCK_INIT()

El fragmento de código en la FIG. 3.13 inicializa el proceso con la librería de "socket", y chequea si está corriendo INET.SYS.

2.- La aplicación debe obtener un descriptor de "socket" usando la llamada **socket()**, como en el ejemplo de la FIG. 3.14. Para más detalles ver el apéndice A.

```
int socket(int domain, int type, int protocol);
.
.
int s;
.
.
s = socket(AF_INET, SOCK_STREAM, O);
```

FIG. 3.14. APLICACION USANDO LA LLAMADA socket().

El fragmento de código en la figura 3.14 ubica un descriptor de socket **s** en la dirección de la Familia de internet. El parámetro dominio ("domain"), es una constante que especifica el dominio donde toma lugar

la comunicación . Un **dominio**, es la colección de aplicaciones que están usando la misma convención de nombramiento. TCP/IP para OS/2 soporta una familia de dirección : AF_NET. El parámetro tipo ("type"), es una constante que especifica el tipo de "socket", el cual puede ser SOCK_STREAM, SOCK_DGRAM, o SOCK_RAW. El parámetro protocolo ("protocol") es una constante que especifica el protocolo que se va a usar,el cual es ignorado a menos que el tipo sea fijado a SOCK_RAW. Si la constante es 0 escoge el protocolo por "default". Si es exitoso, socket() retorna un entero positivo que es el descriptor del "socket".

3.- Una vez que la aplicación tenga el descriptor del socket, puede explícitamente ligar **[bind()]** un nombre único al "socket", como en el ejemplo de la figura 3.15. Para un mayor detalle ver apéndice A.

```
int rc;
int s;
struct sockaddr_in  myname;
int  bind(int s, struct sockaddr *name, int namelen);
/* Limpia la estructura para asegurarse que el campo sin_zero está
limpio*/
memset(&myname,0,sizeof(myname));
myname.sin_family = AF_INET;
myname.sin_addr = inet_addr(*129.5.24.1*);  /*interface especifica*/
myname.sin_port  = htons(1024);
:
rc= bind(s, (struct sockaddr *) &myname, sizeof(myname));
```

FIG. 3.15. Una aplicación que usa la llamada bind()

Este ejemplo liga ("binds") myname al "socket" **s**. El nombre especifica que la aplicación está en el dominio (AF_INET) en la dirección de internet 129.5.24.1, y es confinado al puerto 1024. Los servidores deben ligar un nombre, el que llega a ser el más accesible de la red . El ejemplo en la figura 3.15 muestra 2 rutinas muy utilizadas:

1.- **inet_addr()** toma una dirección internet en forma de puntos decimales y retorna a él en orden de byte de la red. Para una descripción detallada, ver el apéndice A: inet_addr().

2.- **htons()** toma el número del puerto en orden del byte del hostal y retorna el puerto en la red en orden de byte. Para mayor detalle , ver apéndice A.

La figura 3.16. muestra otro ejemplo del llamada bind() en el lado del servidor.. Usa una rutina de gran utilidad de la red, **getservbyname()** para encontrar un número del puerto bien-conocido para un servicio especifico del archivo TCPIP\ETC\SERVICES. La figura 3.16 también muestra el valor "wildcard" INADDR_ANY. Si un hostal tiene muchas direcciones de redes, los mensajes se envían a cualquiera de las direcciones que serían deliberadas hacia el socket.

```
int   rc;
int   s;
struct sockaddr_in  myname;
```

```
int bind(int s, struct sockaddr_in name, int namelen);
struct servent *sp;
:
sp = getservbyname("login","tcp"); /* obtiene una aplicación*/
                    /* específica de un puerto*/
                    /* bien conocido */
/* limpia la estructura para asegurarse que el campo sin_zero esté
limpio*/
memset(&myname,0, sizeof(myname));
myname.sin_family  AF_INET;
myname.sin_addr.s_addr =  INADDR_ANY;
myname.sin_port =  sp-> s_port;
:
rc = bind(s,(struct sockaddr *)&myname, sizeof(myname));
```

FIG. 3.16 Llamada bind() usando la llamada getservbyname()

4.- Después de ligar un nombre al socket, un servidor usando un flujo de "sockets" debe indicar su disposición para aceptar conexiones de clientes. El servidor hace esto con la llamada listen() como muestra en la figura 3.17

```
int   s;
int backlog;
int rc;
int listen (int  s, int  backlog);
:
rc = listen(s,5);
```

FIG. 3.17 Una aplicación utilizando la llamada listen()

La llamada listen() le dice al software de TCP/IP que el servidor está listo para empezar a aceptar la conexión y que un máximo de cinco conexiones solicitan que pueden ser encoladas por el servidor. Las

solicitudes adicionales son ignoradas. Para mayor información, ver Apéndice A.

5.- Los clientes que están usando el flujo de "sockets" inician una conexión solicitada por la llamada connect(), como muestra la figura 3.18.

```
int  s;
struct  sockaddr_in servername;
int rc;
int connect(ints, struct sockaddr *name, int name_len);
:
memset(&servername, 0, sizeof(servername));
servername.sin_family = AF_INET;
servername.sin_addr = inet_addr(*129.5.24.1*);
servername.sin_port = htons(1024);
:
rc= connect(s, (struct sockaddr *) &servername, sizeof(servername));
```

FIG. 3.18 Una aplicación utilizando la llamada connect()

La llamada connect() pretende conectar al "socket **s** hacia el servidor con el nombre servername. Esto podría ser el servidor que fue usado en el ejemplo previo con bind(). El que llama opcionalmente se bloquea hasta que la conexión es aceptada por el servidor. En el retorno exitoso, la conexión del "socket" **s** está asociado con el servidor. Para mayor descripción ver en el Apéndice A: connect().

6.- Los servidores que están usando flujo de "sockets" aceptan una solicitud de conexión con la llamada accept(), como se muestra en la FIG. 3.19.

```
int clientsocket;
```

```
int   s;
struct sockaddr clientaddress;
int   addrlen;
int accept (int s,struct sockaddr *addr,int addrlen);
:
addrlen =  sizeof (clientaddress);
:
clientsocket=accept(s, &clientaddress, &addrlen);
```

FIG. 3.19 Una aplicación usando la llamada accept()

Si no hay solicitudes de conexión pendientes en el socket s, la llamada accept() opcionalmente bloquea al servidor. Cuando la solicitud de conexión es aceptada en el socket s, el nombre del cliente y la longitud del nombre del cliente son retornados con un nuevo descriptor del "socket". El nuevo descriptor del "socket" está asociado con el cliente que ha inicializado la conexión y s es nuevamente habilitada para aceptar nuevas conexiones. Para mayores detalles ver Apéndice A.

7.- Los Clientes y Servidores tienen muchas llamadas que pueden escoger para la transferencia de datos. Las llamadas **readv() y writev(), y send() y recv()** pueden ser usadas solamente en "sockets" que se encuentran en "estado conectado". Las llamadas **sendto() y recvfrom()** pueden ser usadas en cualquier momento. El ejemplo en la FIG. 3.20 ilustra el uso de send() y recv().

```
int bytes_sent;
int  bytes_received;
char data_sent[256];
char data_received[256];
```

```
int send (int socket, char *buf, int buflen, int flags);
int recv (int socket, char *buf, int buflen, int flags);
int s;
:
bytes_sent = send(s,data_sent, sizeof(data_received),0);
:
bytes_received = recv(s, data_received, sizeof(data_received),0);
```

FIG. 3.20. Una aplicación usando las llamadas send() y recv()

El ejemplo en la figura 3.20 muestra un aplicación que está enviando datos en un socket conectado y recibiendo datos como respuesta. Los campos "flags" son usados para especificar opciones adicionales para send() o recv(), así como para enviar datos fuera de banda. Para mayor información de las rutinas readv(), recv(), send() y write(), ver el Apéndice A.

8.- Si el "socket" no está en un estado de conectado, una información adicional de dirección debe ser pasada hacia sendto() y puede ser opcionalmente retornada desde recvfrom(). Un ejemplo del uso de estas llamadas se muestran en la FIG. 3.21.

```
int bytes_sent;
int bytes_received;
char data_sent[256];
char data_received[256];
struct sockaddr_in to;
struct sockaddr_from;
int addrlen;
int sendto(int socket, char *buf, int buflen, int flags, struct sockaddr *addr,
int addrlen);
```

```
int recvfrom(int socket, char *buf, int buflen, int flags, struct sockaddr
*addr, int *addrlen);
int s;
:
to.son_family = AF_INET;
to.sin_addr = inet_addr("a29.5.24.1");
to.sin_port = sendto(s,data_sent, sizeof(data_sent),0, &to, sizeof(to));
:
bytes_sent = sendto(s,data_sent, sizeof(data_sent),0,&to,sizeof(to));
:
addrlen =sizeof(from); /*must be inicialized*/
bytes_received = recvfrom(s,data_received, sizeof(data_received),0,
&from, &addrlen);
```

FIG. 3.21. Una aplicación usando las llamadas sendto() y recvfrom()

Las llamadas **sendto() y recvfrom()** toman parámetros adicionales que permiten al que llama especificar el recipiente del dato ó ser notificado al que envía el dato. Usualmente sendto() y recvfrom() son usados por "sockets" tipo datagramas, y send() y recv() son usados por "sockets" tipo flujo.

9.- Las llamadas writev() y readv() proveen de características adicionales de datos recogidos y dispersados. Los datos dispersados pueden estar localizados en múltiples buffers de datos. La llamada writev() recoge los datos dispersados y se los envía al buffer. La llamada readv() recibe el dato y los recoge en múltiples buffers.

10. Las aplicaciones pueden manejar múltiples "sockets". En dichas situaciones, usa la llamada select() para determinar los "sockets" que tienen datos para ser leídos, aquellos datos que están listos para ser

escritos, y los "sockets" que tienen condiciones excepcionales pendientes. Un ejemplo de cómo es utilizada la llamada select() es mostrada en la FIG. 3.22.

```
#define BSD_SELECT

fd_set readsocks;
fd_set writesocks;
fd_set exceptsocks;
struct timeval timeout;
int number_of_sockets;
int number_found;
:
/* fija los bits en lectura escritura excepto los bits de la máscara. Para fijar la
máscara de un*/
/* descriptor s use readsocks = fd_set(s);
*
* fija el número de "sockets" a ser chequeados
* number_of_sockets = x;
*/
:
number_found = select(munber_of_sockets, &readsocks, &writesocks,
&exceptsocks, &timeout);
```

FIG. 3.22 Una aplicación que usa la llamada select().

En este ejemplo, la aplicación fija los bits de la máscara ("mask") para indicar que los "sockets" están siendo probados para ciertas condiciones y además indicar un "time-out". Si el parámetro "time-out" es NULL, la llamada no espera por ningún "socket" que va a estar listo en estas condiciones . Si este parámetro es diferente de cero, select() espera hasta esta cantidad de tiempo para que al menos un "socket" llegue a estar listo en las condiciones indicadas. Esto es muy usado para aplicaciones que

sirven a múltiples conexiones que no pueden proporcionar bloqueo, y están esperando por el dato en una conexión.

11. Además de select(), las aplicaciones pueden usar la llamada ioctl() para ayudar a ejecutar operaciones asincrónicas de "sockets" (no_bloqueadas).

```
int s;
int dontblock;
char buf[256];
int rc;
int  ioctl (int s, unsigned long command, char *command_data,int
datasize);
:
dontblock =1;
:
rc = ioctl (s, FIONBIO, (char *) &dontblock, sizeof (dontblock));
:
if (recv(s,buf, sizeof(buf),0) == -1 && errno == EWOULDBLOCK)
   /*  no data available */
else
  /* either got data or some other error ocurred */
```

FIG. 3.23. Una aplicación que usa la llamada ioctl().

Este ejemplo causa que el socket s sea ubicado en modo sin bloqueo. Cuando este "socket" es pasado como parámetro para llamadas que podrían bloquear, como recv() cuando el dato no está presente, esto causa que la llamada retorne un código de error, y el valor global errno es fijado a EWOULDBLOCK. Fijando el modo del "socket" a no bloqueo permite que una aplicación continúe procesando sin llegar a bloquearse.

12. Un descriptor del "socket", s, es desubicado con la llamada soclose(). Un ejemplo de esta llamada se muestra en la FIG. 3.24.

```
:
/* cierre el socket */
soclose(s);
:
```

FIG. 3.24. Una aplicación que usa la llamada soclose()

3.4 METODO RPC ("Remote Procedure Call")

La Librería de RPC proporciona las rutinas que capacitan a los programas locales a ejecutar procedimientos en computadores remotos. Estas rutinas transfieren solicitudes y respuestas entre clientes (los programas están llamando a los procedimientos) y servidores (los programas están ejecutando los procedimientos). Cuando escribimos una aplicación distribuída, usualmente no necesitamos usar directamente rutinas RPC. En lugar de ello, podemos crear una definición de la interfase en el Lenguaje de Definición de la Interfase de Red y usar el compilador NIDL (es un compilador que sirve como herramienta para el desarrollo de aplicaciones) para generar las rutinas requeridas de RPC.

3.4.1. LIBRERIA DE LAS LLAMADAS A PROCEDIMIENTO REMOTOS

La librería de RPC nos capacita para hacer una llamada a un procedimiento que no reside en en espacio de dirección del proceso que lo está llamando. El proceso puede estar en el mismo computador, (como un proceso que está llamando) o en uno diferente. Una llamada a un procedimiento remoto es análogo a un salto remoto hacia una llamada de una subrutina. La Fig. 3.24 muestra como el hilo ("thread") de control de la aplicación local de la llamada al procedimiento, es pasada sobre el computador remoto donde el

procedimiento es ejecutado y entonces, el control es retornado a la aplicación local. El mecanismo RPC provee de la semánticas usada en llamadas a funciones para la comunicación entre procesos locales o remotos.

La librería RPC libera los mensajes usando un transporte que provee la comunicación de una aplicación a otra. En el caso de RPC, el transporte es el que permite la comunicación entre mecanismos RPC que están en diferentes computadores, además, puede regular el flujo de información.

Un problema al pasar los argumentos y resultados de RPC es que pueden existir diferencias entre las arquitecturas del computador local y del remoto. Es decir, que el procedimiento remoto puede no interpretar apropiadamente los valores que le son pasados, o la aplicación local puede no interpretar apropiadamente los valores que le son retornados.

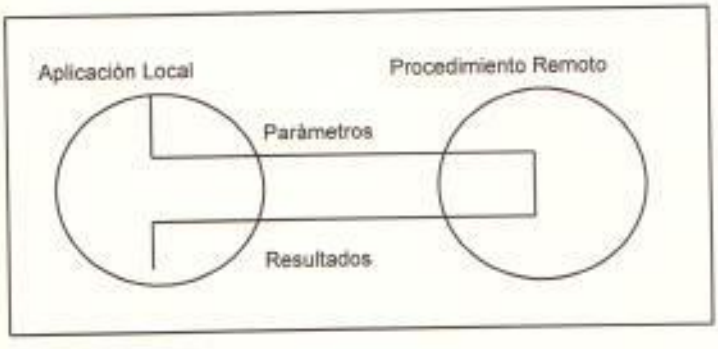


FIG. 3.25 Aplicación Local versus Procedimiento Remoto

3.4.2 APLICACIONES DISTRIBUIDAS

La librería RPC da la capacidad de escribir aplicaciones distribuidas, que consisten en un conjunto de procedimientos, de los cuales no todos residen en un solo computador. Algunos de los procedimientos típicamente residen en diferentes computadores que están interconectados por una red de comunicación. Los procedimientos remotos son invocados usando la librería RPC. Una aplicación distribuida es heterogénea si los procedimientos remotos están corriendo en diferentes computadores con diferentes arquitecturas y/o diferentes sistemas operativos, desde los computadores donde residen las llamadas o procedimientos.

La librería RPC es usada para : ejecutar procedimientos remotos, procesamiento de imagen distribuída, manejo de la red, servicio de información backup, y mucho más. El número de aplicaciones distribuidas y sus diversidades es un testamento a las utilidades de las aplicaciones distribuidas en el área industrial.

3.4.3 MODELO CLIENTE/SERVIDOR

La librería RPC usa el modelo Cliente/Servidor, el cual es utilizado en aplicaciones distribuidas. En este modelo, el servidor ofrece servicios a la red donde el cliente puede accesar. Una aplicación puede ser un

cliente y servidor. Otra forma de entender el modelo es que los servidores proveen de recursos, mientras que los clientes los consumen. Los términos cliente y servidor no necesariamente denotan computadoras; podríamos imaginarlos como un proceso(s) cliente y proceso(s) servidor(es). Existen dos tipos de servidores: ***Servidores "poco declarativos" y Servidores "muy declarativos"***. Un servidor "Poco declarativo" no necesita mantener ninguna información (llamada Estado) de ninguno de sus clientes en orden de que funcionen correctamente. Un servidor muy declarativo (esto es, uno que tiene un estado) mantiene la información del cliente desde una llamada a un procedimiento remoto a la siguiente.

La figura 3.25 muestra un simple archivo servidor muy declarativo que tiene los procedimientos "open y read". El cliente solicita que el servidor abra un archivo, pasando el nombre del mismo y el modo asociado con él. El servidor retorna un descriptor del archivo al cliente, entonces el cliente puede usarlo (al descriptor) en una subsecuente operación de lectura. Note que el servidor está manteniendo el estado del archivo abierto del cliente, el cual incluye el nombre del archivo, el modo de apertura, y la posición actual dentro del archivo. El servidor debe ser capaz de tomar el descriptor del archivo especificado en un comando de lectura y mapearlo al estado apropiado que está asociado con el archivo abierto.

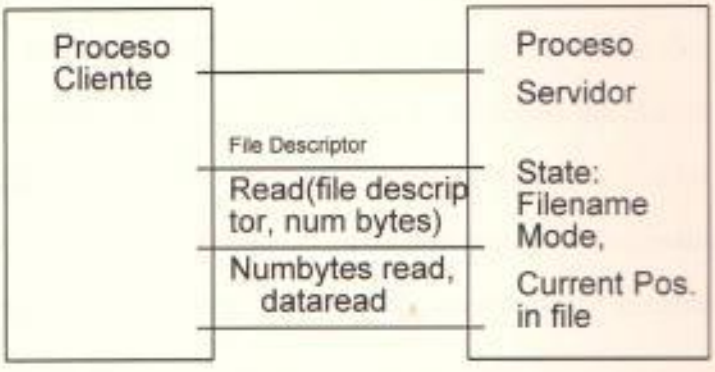


FIG. 3.26 Un simple archivo servidor muy declarativo

La figura 3.26 muestra un simple ***archivo servidor poco-declarativo*** que tiene solamente el procedimiento ***"read"***; pero no tiene un procedimiento de apertura (open). El cliente pasa toda la información dentro de una solicitud de lectura (read) que necesita el servidor, para localizar el dato solicitado, incluyendo el nombre del archivo, la posición dentro del archivo para empezar la lectura, y el número de bytes a leerse. El servidor entonces retorna el número de bytes leídos (read), la nueva posición del archivo, y el dato leído. Note que el estado del archivo está ahora localizado en el cliente.

En el caso de una falla, los servidores poco-declarativos tienen distintas ventajas sobre los servidores muy-declarativos. Con los servidores poco-declarativos, un cliente necesita solo reintentar una solicitud hasta que el servidor responda; éste no necesita conocer que el servidor ha fracasado o que la red temporalmente se ha caído. El cliente de un servidor muy-declarativo por otro lado, necesita ya sea

detectar el fracaso del servidor y reconstruir el estado del mismo cuando regresa o la causa de que falle la operación del cliente . Si el cliente fracasa, entonces el servidor puede sostener por mucho tiempo el estado que no es válido . Los servidores muy declarativos son tipicamente más eficientes que los servidores poco-declarativos y proveen de un fácil paradigama de programación.

El uso de servidores poco-declarativos o muy-declarativos depende de la aplicación que se está usando. Durante el desarrollo de una aplicación distribuida, es mejor examinar los aspectos positivos y negativos de ambos enunciados.

3.4.4 VENTAJAS DEL USO DE RPC

La librería de RPC simplifica el desarrollo de aplicaciones distribuidas proveyendo un modelo de programación muy familiar. Este modelo efectivamente esconde los detalles de programación de la red, permitiéndole enfocarse en resolver los problemas direccionados por la aplicación. Este es un modelo flexible que puede soportar una variedad de diseños de aplicaciones. Estructurando una aplicación distribuida como un conjunto de procedimientos que definen un servicio, la librería RPC puede usarse para construir bloques de aplicaciones sofisticadas. Estos procedimientos remotos pueden usarse en diferentes aplicaciones por muchos clientes, y servidores que pueden también llamar a otros servidores para acoplar su trabajo. Un

servicio de red puede ser visualizado como una librería de procedimientos que están disponibles en ella. Los beneficios de esta librería son: añadir flexibilidad, debido a que las rutinas están dinámicamente limitadas al tiempo de corrida, mientras que las rutinas de librería pueden ser compartidas y distribuídas.

La inherente naturaleza modular de las aplicaciones distribuidas puede ser usada como una ventaja. Una colección de servicios de red específicos, juntos, ejecutan las funciones que necesita para obtener todo perfectamente y que así tienda a ser más fácil de manejar que una gran entidad de red como en un sistema operativo distribuido. Las piezas modulares hacen fácil implementar aplicaciones distribuidas heterogéneas porque los módulos son más fáciles de portar que los sistemas operativos enteros.

La librería RPC esconde las dependencias del sistema-operativo. El único requerimiento para portar una aplicación basada en RPC es que el ambiente de observación del computador provea de una librería RPC que es compatible con la provista por el computador en cuya aplicación fue originalmente desarrollada. El mecanismo RPC de Sun no puede comunicarse con otros mecanismos de RPC que no emplean un protocolo RPC de Sun basado en mensajes.

El problema existe porque no hay un protocolo standard de RPC . En otras palabras, no hay preescritas un conjunto de reglas que gobierne la interacción de mecanismos RPC. Además, no hay reglas que especifiquen la interface de programación de una aplicación (API: Aplication Programming Interface) para la librería RPC.

3.4.5 PROTOCOLO RPC

La librería RPC usa un esquema de pasar-mensajes para manejar la comunicación entre el servidor y el cliente. Para que un cliente pregunte al servidor si puede ejecutar un procedimiento remoto, el cliente debe enviar un mensaje alrededor de la red hacia el servidor; Similarmente, un servidor debe indicar que ha ejecutado el procedimiento solicitado, él debe enviarle un mensaje de regreso al cliente que lo está solicitando. El protocolo RPC define estos mensajes, de los cuales hay dos tipos, Mensajes de llamada y Mensajes de respuesta. Los clientes envían mensajes de llamada hacia los servidores, un mensaje de llamada solicita la ejecución de un procedimiento remoto en particular y contiene los argumentos del mismo. Después de la ejecución de un procedimiento remoto para un cliente, el servidor le envía un mensaje de respuesta ; éste contiene los resultados de la ejecución del procedimiento remoto. Los campos en los mensajes de llamada y de respuesta son codificados de acuerso al estandard XDR, permitiendo a clientes y servidores correr en diferentes arquitecturas de computadoras. Note que usted puede

especificar su propia representación para argumentos y resultados de otros campos dentro de los mensajes que son siempre codificados usando XDR.

El protocolo especifica solo el formato de mensajes de llamada y respuestas y la interpretación de los campos en estos argumentos. Los implementadores son libres para:

a.- Transmitir mensajes RPC vía cualquier transporte: el protocolo está libre de transportar dependencias.

b.- Proveer de cualquier interface de programación para el software del cliente y servidor. Los programas clientes pueden o no conocer que están ejecutando procedimientos remotos; los cuales pueden ser bloqueados hasta que uno de ellos retorne un resultado, o pueden también correr en paralelo. Un procedimiento remoto puede o no estar siendo llamado alrededor de la red, y los procedimientos remotos pueden ser ejecutados en un servidor por un proceso, por un número fijo de procesos, o por procesos creados dinámicamente para cada mensaje de llamada.

c.- Implementar el protocolo RPC dentro o fuera del kernel del sistema operativo o en ambos. Note que las implementaciones de RPC en una red dada variarán radicalmente de un computador a otro, ellos pueden actuar como clientes o como servidores, o como ambos , simplemente observando el protocolo RPC.

3.4.5.1 MENSAJE DE LLAMADA

Una implementación RPC puede ser conceptualmente dividida en un lado cliente y el otro como servidor. Para ejecutar una llamada a un procedimiento remoto, el cliente envía un mensaje de llamada al lado del servidor. Todos los campos en el mensaje son del tipo standad XDR. Un mensaje de llamada empieza con un campo de identificación de transacción (XID); el lado cliente típicamente inserta un número secuencial en este campo.

El campo XID es principalmente usado para igualar los mensajes de contestación, para destacar los mensajes de llamada.

El campo del tipo de mensaje distingue los mensajes de llamada de los de contestación. Los mensajes de llamada fijan el campo a cero. Seguido del tipo de mensaje viene una versión numérica de RPC. Nótese que el lado del servidor no necesita soportar todas las versiones del protocolo RPC, pero debe rechazar los mensajes de llamada que especifican una versión de protocolo que no soporta. Seguido a este campo viene el programa remoto, versión y número de procedimientos. Después hay dos campos, credenciales del cliente y verificador

del cliente, que identifican al usuario cliente para la aplicación distribuída.

Debido a que el protocolo RPC es independiente del servicio de transporte en el cual es diseñado, el protocolo no puede definir la longitud máxima de la llamada y mensajes de contestación. Para definir los mensajes que son compatibles con un grupo de transportes, los desarrolladores de la aplicación distribuida deben asegurarse que sus mensajes no excedan de la longitud máxima más pequeña especificado por cualquier transporte.

3.4.5.2 MENSAJES DE CONTESTACION

Dos mensajes de contestación están disponibles y son: Respuestas a las Llamadas exitosas y Respuestas a Llamadas fracasadas.

La Respuesta a una Llamada"Exitosas" es definida desde el punto de vista del servidor de la librería RPC, y no del procedimiento Remoto. Una respuesta exitosa significa que el lado del Servidor ha encontrado algo errado con el mensaje de llamada. Si un procedimiento remoto rechaza un mensaje de llamada, retorna una respuesta exitosa, usualmente con un código de error como resultado del mismo.

XID= Identificador de la Transacción

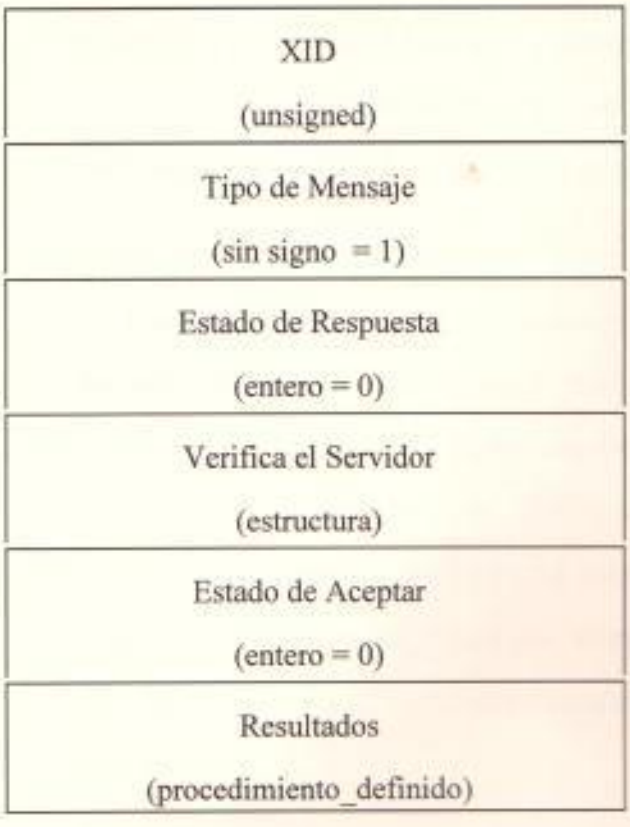


FIG. 3.27 FORMATO DE UN MENSAJE DE RESPUESTA EXITOSA

El formato empieza con la transacción ID (XID) del mensaje de llamada correspondiente seguido por el tipo del mensaje fijado a 1, lo que lo identifica como respuesta. El campo estado de la respuesta y el campo estado de aceptación distinguen una respuesta exitosa de una no exitosa. El campo verificador del

Servidor es usado para su autentificación. El campo final en una respuesta exitosa contiene los resultados retornados por el procedimiento remoto. El número y tipo de los resultados son definidos por los que desarrollan los procedimientos remotos, pero todos son codificados como tipos estándares XDR.

El lado del servidor RPC es responsable de rechazar los mensajes de llamada que violan el protocolo RPC. El lado servidor no puede detectar malos argumentos o credenciales inadecuadas, detectarlas es responsabilidad de los procedimientos remotos. Entonces, un mensaje a una respuesta exitosa no necesariamente indica que el procedimiento remoto fue ejecutado. Las siguientes condiciones pueden ser retornadas:

1.- El procedimiento remoto se ha ejecutado exitosamente.

2.- El programa remoto, la versión, o número del procedimiento especificado en el mensaje de llamada no es válido en el servidor. Si la discrepancia es el número de la versión, la respuesta contiene el más bajo y el más alto número de las versiones disponibles.

3.- El procedimiento remoto no puede decodificar los argumentos.

Los Mensajes a contestaciones no exitosas tienen el mismo formato que las respuesta exitosas hasta el campo del estado de la respuesta, el cual, es fijado a 1. El formato del resto del mensaje depende de las condiciones que hacen que la llamada sea fallosa. Estas condiciones son descritas a continuación:

1.- El lado del servidor RPC no soporta la versión del protocolo RPC especificado en el mensaje de llamada. La respuesta contiene el número de versión más alto y el más bajo soportados por RPC.

2.- Las credenciales del cliente o verificador están impropiamente formadas; La respuesta contiene un valor auth_stat que describe el problema.

3.4.6 EL PROTOCOLO DE SERVICIO DE LA RED "PORTMAP"

Un cliente que necesita hacer una llamada a un procedimiento remoto para un servicio debe ser capaz de obtener la dirección de transporte del servicio. El proceso de traslación del nombre del servicio a su dirección de transporte es referida como **ligamento** al servicio.

Una mejor aproximación permitiría un ligamento dinámico, proveyendo de algún mecanismo que traslade el nombre del servicio en su dirección de transporte. En sistemas distribuidos, los nombres y direcciones de transporte de los servicios actualmente disponibles

están almacenados en un nombre del servidor. Un servicio registra su nombre con el nombre del servidor, cuando éste inicia primero.

Un cliente desea hacer un RPC, entonces, presenta el nombre del servicio en una pregunta (query) al nombre del servidor. Este, a su vez, mira el nombre del servicio en sus tabla internas y retornan la dirección del transporte del servicio, con tal que el servicio se haya registrado a si mismo con el nombre del servicio.

El servicio "portmap" es un servicio de la red que proporciona un camino estandard para que un cliente mire el número de puerto de cualquier programa remoto del servidor, el cual ha sido registrado con el servicio.

El programa "portmap" es un servicio de ligamento. Debido a que puede ser diseñado en cualquier transporte que provea el equivalente de puertos, el servicio "portmap" da una solución a un problema general que trabaja para clientes, servidores y la mayoría de las redes. Un "portmap" es una lista del número de correspondencia puerto-a-programa/ versión en un computador, es decir, cada versión diferente de un programa remoto puede usar su propio puerto.

Un servicio "portmap" es también llamado "portmapper", porque mantiene las entradas de sus "hosts portmap". Todo computador que

soporta servicios basados en RPC corren una implementación del "portmapper". El "portmapper" es tipicamente inicializado automáticamente cuando el computador es booteado.

Para encontrar el puerto de un programa remoto, el programa cliente envía un mensaje de llamada RPC al "portmapper" del servidor; si el programa remoto es registrado en el servidor, el "portmapper" retorna el número del puerto reelevante en un mensaje de respuesta RPC. El programa cliente puede entonces enviar mensajes de llamada RPC al puerto del programa remoto. Un programa cliente puede minimizar sus llamadas al "portmapper" escogiendo los números de puertos de las recientes llamadas a Procedimientos Remotos.

CAPITULO IV

4. DESCRIPCION DE LAS APLICACIONES DEMOSTRATIVAS DESARROLLADAS.

4.1. APLICACION BASADA EN NETBIOS

Esta aplicaciòn usa el protocolo OS/2 NetBIOS LAN, la interfase està empaquetada como una Libreria Dinàmica de Enlace (Dynamic Link Library: DLL). la cual està creada en el programa FXNETB.C .

La comunicaciòn DDL de NetBIOS permite transferir archivos usando FXREQ.C y FXSVR.C ,dichos programas cliente/servidor son archivos de transferencia genèricos, incluidos en el apèndice B.

Las funciones de NetBIOS son definidas por un conjunto de comandos ò verbos, los cuales van a ser ejecutados en un especifico NCB (Network Control Block), es decir, API NetBIOS consiste de una sola llamada de OS/2.

El còdigo presentado en esta secciòn no contiene el procedimiento principal main(). El procedimiento principal es provisto por el programa que llama a las funciones DDL, esto significa que FXNETB.C no es un programa separado, sino que contiene una lista de funciones que son empaquetadas en un DDL.

FXNETB.C.

El listado contiene toda la miscelànea de programaciòn que debe ser usada por los procedimientos FX., esto incluye las declaraciones usuales de constantes,

variables globales, y declaraciones de funciones. Note que en la cabecera incluimos todas las estructuras de NetBIOS de OS/2.

Las funciones incluìdas en este mòdulo tienen 2 categorias: Funciones FX y funciones de ayuda. Las funciones de ayuda son usadas para proveer servicios comunes para ese mòdulo, las cuales estàn descritas en la funciòn print_error().

Estructuras de Datos utilizadas para NCB:

a) Estructura general para comandos NCB:

```
typedef estruct
{
    BYTE    command;          /* Còdigo del comando NetBIOS*/
    BYTE    retcode;          /* Còdigo de retorno */
    BYTE    lsn;              /* Nùmero de la sesiòn local */
    BYTE    num;              /* Nùmero del nombre de la aplicaciòn*/
    PBYTE   _Seg16 buffer_address  /* Direcciòn del buffer del mensaje*/
    USHORT  length;           /* longitud del mensaje del buffer */
    BYTE    callname [16];    /* Nombre del destino */
    BYTE    name[ 16 ];       /* Nombre de la fuente */
    BYTE    rto;              /* timeout de recepciòn */
    BYTE    sto;              /* timeout del emisor */
    PBYTE   _Seg16 post_address; /* direcciòn de la rutina anterior */
    BYTE    lana_num;         /* nùmero del adaptador */
    BYTE    cmp_cplt;         /* estatus del comando */
    BYTE    reserve[14];      /* Reservado (excepto RESET) */
```

}CMD_NCB;

Estructura reset de NCB usada en DDL

typedef estruct

{

```
BYTE      command;          /* Còdigo del comando NetBIOS */
BYTE      retcode;          /* Còdigo de retorno */
BYTE      lsn;              /* 0 = Solicitud  1= Liberar Recursos*/
BYTE      num;              /* No es usado */
BYTE      _Seg add_name_address   /* No usado */
USHORT    length;           /* No usado */
BYTE      req_sessions;     /* No usado */
BYTE      req_commands;     /* Nùmero de sesiones solicitadas */
BYTE      req_names;        /* Nùmero de comandos solicitados */
BYTE      req_name_one;     /* Nùmero de nombres solicitados */
BYTE      not_used_2[12];   /* No usado */
BYTE      act_sessions;     /* Nùmero de sesiones obtenidas de DLL*/
BYTE      act_commands;     /* Nùmero de comandos obtenidos de DLL */
BYTE      act_names;        /* Nùmero de nombres obtenidos de DLL */
BYTE      act_name_one;     /* Nùmero del nombre de una resp. de DLL */
BYTE      not_used3[4];     /* no usado */
BYTE      load_session;     /* Nùmero de sesiones llamadas de DLL */
BYTE      load_commands;    /* Nùmero de comandos llamados de DLL*/
BYTE      load_names;       /* Nùmero de nombres llamados de DLL */
```

```
    BYTE        load_stations;      /* Nùmero de estaciones llamadas de DLL*/
    BYTE        not_used4[2];       /* No usado */
    BYTE        load_remote_names; /* Nùmero de nombres remotos de DLL */
    BYTE        not_used5[5];       /* No usado */
    USHORT      dd_id;              /* No usado */
    BYTE        lana_num;           /* Nùmero del adaptador */
    BYTE        not_used6;          /* No usado */
    BYTE        reserve[14];        /* Informaciòn del error NCB */
}RESET_NCB;
```

Descripciòn de las funciones utilizadas por FXNETB.C

1.- fx_Initialize( ).

La funciòn fx_Initialize lee el archivo de la configuraciòn del protocolo especificado para obtener los paràmetros requeridos para inicializar el sistema. El nombre del usuario especificado del archivo de configuraciòn para NetBIOS es fxnetb.cfg. A continuaciòn tenemos un ejemplo de dicho archivo:

```
PIPE_NAME = NETSVR
FILE_BUFFER_SIZE  =  512
```

El significado de estos dos paràmetros es el siguiente:

PIPE_NAME. Este paràmetro es usado para derivar el nombre de la red del cliente y del servidor. El archivo de configuraciòn dado arriba resulta con una màquina cliente llamada NETSVR.REQ y una màquina servidora llamada NETSVR.SVR.

FILE_BUFFER_SIZE: Este paràmetro define el tamaño del buffer del mensaje, el cual es usado para lectura y escritura del dato hacia el disco y para intercambio de mensajes a travèz de la LAN ò el IPC. El espacio de memoria para este paràmetro debe ser ubicado dinàmicamente. La funciòn fx_Initialize() retorna un puntero al buffer y el tamaño de la aplicaciòn.

2.- fx_MakePipe( ).

La funciòn fx_MakePipe( ) es usada por el servidor de un pipe FX para establecer el pipe. Al usar NetBIOS, esto provee de las siguientes funciones.

2.1. El comando **NetBIOS RESET** para asignar recursos de la red al proceso, dicho comando es ejecutado por la funciòn de ayuda netbios_reset().

2.2. ADD NAME sirve para ubicar el nombre del servidor en la red, dicho comando es ejecutado por la funciòn de ayuda netbios_add_name( ).

3. netbios_reset( ).

El comando RESET es usado de dos maneras:

Para solicitar recursos de NetBIOS.

Para liberar recursos de NetBIOS.

La funciòn netbios_reset() ejecuta el reset dependièndo de los valores del paràmetro request_release que recibe. netbios_reset() realiza lo siguiente:

1. Limpia el NCB.
2. Llena los campos de solicitud de NCB para este comando.
3. Utiliza la llamada a un API de NetBIOS en el modo de espera.
4. Retorna el contenido del campo retcode.

4. netbios_add_name( )

La funciòn netbios_add_name() añade un nombre especifico a la red. Esto lo hace limpiando primeramente el NCB, entonces llena los campos de NCB para este comando usando el tipo ncb.cmd_ncb. El contenido del campo retcode es retornado al que està llamando.

5. fx_ConnectPipe( )

La funciòn fx_ConnectPipe( ) es usada para establecer una conexiòn en el servidor del pipe FX. Al usar NetBIOS se provee de una funciòn utilizando el comando LISTEN, esto se lo hace llamando al comando netbios_listen().

6. netbios_listen( )

La rutina netbios_listen() espera por un caller_name remoto para llamar a station_name local. Esto lo hace limpiando primero NCB,

entonces llena los campos de solicitud NCB para este comando usando el tipo ncb.cmd_ncb. La llamada se completa cuando la llamada es recibida de una estaciòn remota, entonces la sesiòn es establecida. Finalmente el contenido del campo retcode es retornado al que està llamando, asi como un puntero al campo numèrico de la sesiòn local retornada. lsn contiene un nùmero que identifica unicamente la sesiòn.

7. fx_DisconnectPipe( )

La funciòn fx_DisconnectPipe( ) es usado para desconectar al servidor del pipe FX. NetBIOS proporciona esta funciòn utilizando el comando HANG_UP, esto se lo hace utilizando netbios_hang_up( ).

8. fx_Open( )

La funciòn fx_Open( ) es usada para establecer el lado del cliente del pipe FX, usando las fucniones siguientes:

1. El comando RESET para obtener los recursos de NetBIOS. Esto se lo hace invocando a la funciòn netbios_reset( ).
2. Usando el comando ADD NAME para añadir el nombre del cliente a la red, esto se lo hace invocando a la funciòn netbios_add_name( ).
3. Utilizando el comando CALL para establecer una sesiòn con el servidor, invocando a netbios_call( ).

9. **netbios_call( )**

La funciòn netbios_call( ) utiliza el comando CALL. ella establece una sesiòn entre un call_name remoto y una station_name local.

4.2. APLICACION BASADA EN NAMED PIPES

La aplicación desarrollada para demostrar el uso de Named- Pipes incluye un conjunto de Transacciones cuyos objetivos son: Recuperar el Saldo de un Cliente (Consulta de Saldos), y Registrar el pago de cheques y de Depósitos, grabando dicha información en un Archivo de Movimientos.

Estas transacciones serán ejecutadas por un programa que desde ahora llamaremos Servidor (S) . Podrán ser invocadas básicamente por cualquier programa Cliente. Estas transacciones han sido divididas en dos grupos principales:

- **Recuperación** de datos (desde las Bases de Datos personales, Saldos y Movimientos del cliente)
- **Actualización** (Ingreso de movimientos y Actualización de las Bases de Saldos y Movimientos del Cliente)

PROTOCOLO DE COMUNICACION ENTRE PROGRAMAS

Para la comunicación entre el programa Servidor y sus Clientes, se usarán una serie de convenciones, tanto para el establecimiento de la comunicación,

como para el paso de datos entre aplicaciones. El gráfico 4.1 ilustra el modelo de esta aplicación.

A fin de aislar la aplicación Cliente de la mayoría de los detalles de estas convenciones, se proveerá de un conjunto de Funciones, una para cada transacción, que tendrán que ser embebidas en los programas Clientes, los cuales dispondrán tanto del código fuente, como del ejecutable de estas funciones.

De esta forma se provee una interfase más fácil de usar, aunque esto no impide que se ignore esta capa adicional de software, y que se empleen directamente las siguientes convenciones:

- Los programas se comunicarán a través de **Pipes**. Para establecer la **comunicación inicial**, se utilizarán Pipes con **múltiples instancias** de formas que los **Clientes del Servidor sólo necesitarán conocer un único nombre del Pipe.** El Servidor crea varias instancias del mismo Pipe, y los clientes intentan una conexión a alguna de las instancias de este Pipe, pero sin necesidad de conocer a cual.

- Una vez que se obtiene la comunicación inicial, el Cliente debe conectarse al hilo que realmente atenderá su requerimiento (lo llamaremos hilo de proceso). Para ello el Cliente **obtiene del Servidor** (durante la conexión inicial) el **nombre del respectivo Pipe**.

- Los hilos de Proceso del Servidor son de dos tipos: **Permanentes** (estáticos) y **Temporales** (dinámicos). Los Hilos permanentes esperan durante un cierto tiempo (TimeOut configurable) por la conexión (**open**) del Cliente. Si ésta no se produce, el hilo el **liberado** de este estado de espera, y es **rehabilitado** para que atienda requerimientos de nuevos clientes. Los Hilos Temporales también esperan por la conexión del cliente, pero si ésta no produce, o luego de haber atendido el requerimiento, terminan su ejecución.
- En el proceso de comunicación de datos entre Servidor y Clientes, **todos los datos serán de tipo texto**. Es decir, aún los campos definidos como **int**, serán transmitidos como de **tipo char.** Esto facilita enormemente este proceso de comunicación.
- Todos los campos tipo fecha tendrán el siguiente formato: mm/dd/aa.
- Tanto los datos de entrada como los de salida de las transacciones serán ubicados posicionalmente en los buffers de comunicación. En otras palabras, la documentación de las transacciones tiene **listas de parámetros** de entrada y de salida; se debe respetar el orden de aparición de éstos parámetros.

- Los **datos enviados** en una transacción deben formar un solo buffer contínuo con el siguiente formato:

99999	Código de Transacción.
X(n)	Datos de entrada de transacción.

- Los **resultados de una transacción** tendrán un formato similar:

99999	Status de transacción
X(n)	Información solicitada

- Para el establecimiento de la conexión inicial, el respectivo cliente deberá enviar el código: **99999** (no se necesiata ningún datos adicional).
- Si **no se logra establecer** esta conexión inicial, el Servidor retorna: 99990. **Caso contrario** retornará 99991, seguido del nombre del Pipe al cual deberá enviar el requerimiento deseado.
- Una vez que se inicia una transacción, ésta puede retornar cualquiera de los siguientes estatus:

00001	**OK**
01011	**CLIENTE_NO_EXISTE**
01021	**TRAN_ERR**
02011	**TRAN_NO_EXISTE**
03010	**NO_HAY_FONDOS**
04010	**NO_INSERTO_REG**
05010	**TRAN_FINAL**
04020	**NO_ACTUALIZO_REG**
-00009	**Tran_Pendiente**

Si la transacción se ejecutó correctamente, los datos que vendrán serán los resultados de la transacción. Si se produjo algún error, en algunos casos vendrá acompañado de un mensaje.

Junto a los archivos fuentes fuentes y ejecutables de este programa Servidor de servicios al cliente, se proveerá el correspondiente archivo **.H** con los respectivos #define's de estos códigos.

- Las Funciones provistas para llamar a cada una de las Transacciones, cumplirán las siguientes tareas:
  a) Realizar la conexión inicial y obtener el nombre del Pipe definitivo.
  b) Conectarse a este Pipe, e iniciar la respectiva transacción.
  c) Recibir todos los resultados que ésta arroje.
  d) Devolver los resultados al Cliente en buffers de memoria.

Estas funciones sólo necesitarán conocer el **nombre del Pipe** (con múltiples instancias) para la conexión inicial. Tal como se ha mencionado, durante esta conexión inicial el Servidor determina este nombre de Pipe y lo retorna al respectivo Cliente.

Es indispensable que el programa Cliente recupere toda la información disponible, aún cuando no la necesite, ya que una transacción iniciada en el servidor se queda en estado de Pendiente hasta que toda la información haya sido recuperada.

Si se iniciará una nueva Transacción sin haber terminado la anterior, recibirá el código de error Tran_Pendiente (-9). Sin embargo, las funciones encargadas de ejecutar cada transacción, realizarán ésta tarea e impedirán que se produzca esta clase de error.

Cuando la transacción se ha realizado con éxito retornará: TRAN_FINAL, en caso contrario retornará: TRAN_ERR.

TRANSACCION DE RECUPERACION:

GET_SALDOS_CLIE

Cod. Trans: 01000

Descripción: Obtiene todos los datos relacionados con la cuenta del cliente, es decir:

sus datos personales, Saldo anterior y saldo actual, y los movimientos realizados en ella.

Input: char cuenta[8] // Número de cuenta

Output: char data_out[OFFSET+ BUFF_MOVIMS]
// Bloque de memoria con resultados.

Los resultados son primero almacenados en un área de memoria de donde son transmitidos al proceso Cliente. Esta área tiene la siguiente estructura: (Ver estructura dat_clie, saldo_clie, mov_cta)

Funciones invocadas por la Transacción GET_SALDOS_CLIE:

a) ***int OBTIENE_DATOS_CLIENTE***

Descripción: Es una función de tipo entera, la cual retornará un código que indicará si la búsqueda ha sido ó no exitosa, los cuales son los siguientes:

0	NO_EXISTE_CTA
1	ENCONTRO_CLIENTE

Además esta función devolverá los datos personales del cliente comprobando primeramente si la cuenta ingresada existe en la Base Saldos del cliente (saldo_clie), luego recupera el saldo de su cuenta.

Entrada:	char	*num_cta	// Número de la cuenta
Salida:	dat_clie	*datos_clie	// Datos personales del cliente
	saldo_clie	*saldo	// Saldo de la cuenta

b) ***int*** ***Obtiene_Movs***

Descripción: Esta función es de tipo entera, la cual retornará los siguientes códigos:

Exito:	TRAN_OK_FIN
Fracaso:	NO_HAY_MOVIMS

Su función principal es comprobar si existen movimientos, si ellos existen retorna todos los movimientos que se han registrado, de lo contrario retornará cero como número de movimientos.

Entrada:	char	*num_cta

Salida: mov_cta *data_movs // Datos de los movimientos

int num_movs // Número de movimientos

INSERTA_DATOS_MOVIMS

Cod_Trans: 02000

Descripción: Esta función comprueba primero si la cuenta del cliente existe, luego recupera el tipo de transacción que desea realizar, si el tipo es PAGO DE CHEQUES, primero debe comprobar si el saldo en la cuenta es mayor que la cantidad del cheque que se va a pagar, si lo es llama a la función INSERT_ACTUALIZA_DATOS descrita más adelante, de lo contrario retornará que no hay sufientes fondos en dicha cuenta.

Si el tipo de transacción es depósito (DEPO) llama directamente a la función INSERT_ACTUALIZA_DATOS. En ambos casos debe actualizar la fecha del último movimiento.

a) ***int*** ***INSERT_ACTUALIZA_DATOS***

Input: char *data_in // Buffer de datos que envía el cliente

entre ellos tenemos los siguientes datos, que deben seguir el mismo orden:

char cuenta[8] numero de la cuenta

char tipo_tran tipo de la transacción

			(DEPO / CHEQ)
	char	fecha_mov[8],	fecha del movimiento
	char	valor [12],	valor de la transacción
	char	num_doc[8],	número del documento.
Salida:	char	data_out[7]	// Bloque de memoria con resultados.

Los cuales pueden ser:

CLIENTE_NO_EXISTE

TRAN_FINAL

NO_ACTUALIZO_REG

NO_INSERTO_REG

TRAN_ERR

Funciones utilizadas por INSERTA_DATOS_MOVIMS

b) ***int*** ***INSERTA_DATOS***

Descripción: Esta función ingresa los datos del movimiento realizado al Archivo de Movimientos de cuentas de Clientes ("movims.txt"), los cuales son inmediatamente grabados

Entrada:	char	*n_cuenta	// número de la cuenta
	char	tip_mov	// tipo de movimiento
			(1) CHEQUE
			(2) DEPOSITO

char *fecha // fecha del movimiento

char *valor

char *doc // número del documento.

Salida: retorna un entero que indicará:

éxito: TRAN_OK_FIN

fracaso: TRAN_ERROR

c) *int* ***ACTUALIZA_SALDO***

Descripción: Esta función actualiza los datos del archivo de Saldos del cliente, reemplazandolos con los que ya han sido calculados en la transacción principal. Esta función retornará un número entero que indicará: ACTUALIZA_OK, si la función ha sido exitosa, ó TRAN_ERROR si no lo hizo.

Entrada: char *n_cuenta // número de la cuenta

char *sal_act // saldo actual

char *sal_ant // saldo anterior

char *fecha // fecha del último movimiento

DEFINICION DE LAS ESTRUCTURAS DE DATOS Y DISEÑO DE LOS REGISTROS DE ARCHIVOS

```
typedef struct {
    char    num_cta[8];     // Número de la cuenta
    char    tip_mov;        // Tipo del movimiento:
```

```
                                                    1: Pago de Cheques
                                                    2: Depositos
        char        fec_mov[8];         // Fecha del movimiento
        char        monto[12];          // Valor del documento
        char        num_doc[8];         // Número del documento
        } MOV_CTA;
typedef struct {
        char        num_cta[8];         // Número de la cuenta
        char        sal_actual[12];     // Saldo actual en la cuenta
        char        sal_ant[12];        // Saldo anterior de la cuenta
        char        fecha_ult_mov;      // Fecha del último movimiento
        char        num_ced[11];        // Número de la cédula
        char        status_cta;         // Estado de la cuenta: cerrada(c),
                                           abierta (a)
                                           bloqueada (b)
        } SALDO_CLIE;

typedef struct {
        char        ced_cli[11];        // Cédula del cliente
        char        nombre[30];         // Nombre del cliente
        char        direccion[30];      // Dirección del cliente
        char        telefono[6];
        } DAT_CLIE;
```

4.3 APLICACION BASADA EN SOCKETS

La aplicación basada el sockets de TCP/IP presenta una implementación de la capa de comunicación del protocolo FX utilizando el protocolo LAN de Sockets de TCP/IP bajo OS/2. DLL es creada por el programa FXTCPIP.C usando FX.

FXTCPIP.C: LOS SOCKETS FX DLL DE TCP/IP

El código de este programa no presenta un programa principal main( ). Los DLL FX son simplemente funciones que son enlazadas a un archivo (.EXE) en el tiempo de corrida. El procedimiento principal es provisto por el programa que llama a las funciones DLL, esto significa que FXTCPIP:C no es un programa separado, sino que contiene una lista de funciones que son empaquetadas en DLL. Las funciones provistas por este módulo caen en 2 categorías.: Funciones FX y funciones de ayuda..

FXTCPIP.C. Descripción de las funciones y estructuras.

En esta sección se incluyen las declaraciones usuales de constantes, variables globales, declaraciones de funciones, y la descripción de las estructuras de datos.

DESCRIPCION DE LAS FUNCIONES UTILIZADA

1. fx_Initialize( ).

La función lee el archivo de configuración del protocolo especificado para obtener los parámetros requeridos para inicializar el sistema. El nombre del archivo de configuración especificado para sockets es: **fxtcpip.cfg, por** ejemplo:

```
SERVER_PORT      = 1027
HOST_ADDRESS     = 2.0.0.1
FILE_BUFFER_SIZE = 4096
```

- **SERVER PORT** especifica el número de puerto del servidor.
- **HOST_ADDRESS** especifica la dirección internet del servidor.
- **FILE_BUFFER_SIZE** especifica el tamaño del buffer del mensaje, el cual va a ser usado para lectura y escritura de datos al disco y para intercambio de mensajes a través de la LAN ó el IPC. La función fx_Initialize() retorna un puntero al buffer y su tamaño a la aplicación.

sock_init ( ).

La función API sock_init() inicializa la estructura de datos del socket y chequea si está corriendo TCP/IP. Esta función debe ser utilizada al inicio de cada programa que usa sockets, no tiene parámetros asociados a esta llamada.. El prototipo para esta llamada es:

```
sock_init(VOID);
```

Los valores retornados son: 0 que indica éxito, y , 1 que indica un error.

fx_MakePipe( )

La función fx_MakePipe() es usada por el servidor de un pipe FX para establecer el pipe. Esta función introduce 4 comandos de sockets que son: socket(), bswap(), bind(), y listen(), descritas a continuación.

socket( ).

La llamada API socket( ) crea un socket y retorna un descriptor del socket( ), debe especificarse la familia del protocolo que va a ser usado con el socket. El prototipo para esta llamada es:

```
socket ( SHORT   domain,    /* Dirección del dominio que debe estar en AF_INET */
         SHORT   type,      /* Tipo del socket creado */
         ULONG   protocol); /* Protocolo de Transporte */
```

Una descripción de estos parámetros es la siguiente:

- El parámetro domain especifica el dominio de la comunicación, la cual debe ser determinada por el formato de la estructura de la dirección de internet.
- El parámetro type especifica el tipo del socket creado, que puede ser: SOCK_STREAM, SOCK_DGRAM y SOCK_ROW.
- El paràmetro protocolo especifica el protocolo de transporte que va a ser usado en dicho socket. Este campo es fijado a 0, si el sistema selecciona el nùmero del protocolo por default para el dominio y el tipo del socket solicitado.

bswap( )

La llamada API bswap( ) hace un "swap" de los bytes en un entero corto sin signo.bswap() es usada para las conversiones entre Intel "little indian" y "big indian" de TCP/IP. El prototipo para esta llamada es la siguiente:

```
bswap( USHORT   number); /* Nùmero de bytes que serán hechos swapping */
```

La descripciòn de los paràmetros es la siguiente:

- **number**: es un entero corto cuyos bytes van a ser hechos swapping. La llamada retorna el entero corto trasladado.

bind( )

La llamada bind asocia una direcciòn local ùnica (nùmero del puerto) al socket. Esto es importante para los procesos servidores que necesitan especificar un puerto bien conocido para su servicio.

listen( )

La llamada listen( ) escucha por la conexiòn del socket, ella dice si ¿Hay algùn proceso en el socket listo para aceptar la conexiòn?. La llamada establece una pila del flujo de socket para aceptar las solicitudes de conexiòn que van llegando.

fx_ConnectPipe ( )

La funciòn fx_ConnectPipe es usada para establecer una conexiòn en el lado del servidor del pipe FX.

accept( )

La llamada accept( ) es usada por un servidor para aceptar una solicitud de conexiòn del cliente. Esta llamada acepta la primera conexiòn en una pila de conexiones pendientes. La llamada accept crea un nuevo descriptor del socket y retorna dicho valor al que lo està llamando. El socket original se mantiene abierto y puede aceptar nuevas conexiones.

fx_DisconnectPipe( )

La funciòn fx_Disconnect( ) es usada para desconectar el pipe por el lado del servidor.

soclose( )

La funciòn API soclose( ) apaga el socket, liberando sus recursos y cerrando la conexiòn TCP.

fx_Open( )

La funciòn fx_Open( ) es usada para establecer el pipe en el lado del cliente. Esta funciòn provee de dos llamadas a sockets: socket( ) y connect().

connect( )

La funciòn API connect() establece una conexiòn en un flujo de socket disponible. Esta funciòn puede ser usada con datagramas para especificar el destino, hacièndo posible transferir los datos sin especificar a cada momento el destino.

fx_Write ( )

La funciòn fx_Write( ) es usada para enviar informaciòn en el pipe. Dicha funciòn usa una nueva llamada, send( ).

send( )

La llamada API send( ) envìa un paquete en el socket conectado.

fx_Read( )

La funciòn fx_Read( ) es usada para leer la informaciòn del pipe.

recv( )

La llamada API recv( ) recibe datos en un socket conectado y los almacena en el buffer, ella retorna la longitud del mensaje que està llegando. Si los datos no estàn disponibles en el socket , la llamada se bloquea esperando por el arribo de un mensaje, a menos que el socket estè en estado no bloqueado.

fx_Transact( )

La funciòn fx_Transact( ) envia un mensaje y recibe una respuesta sobre el pipe. Esta funciòn es provista usando sockets que invocan fx_write( ) seguido de fx_read( ).

fx_Close( )

La funciòn fx_Close cierra el pipe. En tèrminos de sockets, esto significa utilizar un comando soclose( ), èsta es la misma llamada que usamos en

fx_Disconnect. Los sockets cliente y servidor son simètricos cuando se desconectan.

ESTRUCTURAS DE DATOS DE TCP/IP.

```
typedef struct IN_addr
{
    ULONG s_addr;  } IN_ADDR;
typedef struct  {
    SHORT      sin_family;
    USHORT     sin_port;
    IN_ADDR    sin_addr;
    CHAR       sin_zero[8];
} SOCKADDR_IN;

typedef struct
{
    CHAR *   _Seg16  h_name;        /* Nombre oficial del host  */
    CHAR **  _Seg16  h_aliases;     /* Lista de alias */
    SHORT            h_addrtype;    /* tipo de dirección del host */
    SHORT            h_length;      /* longitud de la dirección  */
    CHAR **  _Seg16  h_addr_list;   /* lista de direcciones del    */
                                    /* nombre del servidor  */
    #define h_addr   h_addr_list[0]    /*dirección para compatibilidad*
```

} **HOSTENT;**

El listado del programa con las respectivas funciones y estructuras se encuentra en el apèndice B.

CONCLUSIONES Y RESULTADOS

1.- La plataforma Cliente-Servidor provee de un ambiente abierto y flexible donde la unión y la igualdad es la regla. La idea de Cliente-Servidor ha sido la de dividir una aplicación para crear varias formas de soluciones del software de redes de área local, las cuales se distinguen por la naturaleza del servicio que provee a sus clientes, entre ellas tenemos a los Servidores de archivos, Servidores de Bases de datos, Servidores de Transacciones y Servidores de aplicaciones.

2.- Con los Servidores de Archivos, el cliente debe solicitar los registros de archivos al servidor, es decir, que se basa en intercambios de mensajes sobre la red para encontrar el dato solicitado.

3.- Los Servidores de Bases de datos reciben las solicitudes SQL del cliente en forma de mensajes y los resultados son retornados sobre la red. El servidor usa su propio poder de procesamiento para encontrar el dato solicitado en lugar de regresar todos los registros al cliente y permitirle encontrar su propio dato.

4.- Con los Servidores de Transacciones el Cliente invoca a los procedimientos remotos que residen en el Servidor. El intercambio de red consiste de un solo mensaje del tipo solicitud/respuesta. Los Servidores de aplicaciones al igual que los servidores de transacciones no son necesariamente una base de datos centralizada, sino que suplen el código para el cliente y servidor.

5.- Las soluciones Cliente-Servidor dan como resultado bajos costos de mantenimento y de desarrollo, además brinda soluciones de alta productividad, debido a su estandar abierto y flexible.

6.- El desarrollo de la aplicación Cliente-Servidor requiere de habilidades que incluyen procesamiento de transacción, diseño de la base de datos, experiencia en comunicación, y, una amigable interfase gráfica con el usuario.

7.- Cliente-Servidor es la industria más sobresaliente debido a que provee de la oportunidad de crear nuestra propia definición. Aquí clientes y servidorés son entidades lógicas que trabajan juntas sobre la red para llevar a cabo una tarea. En el modelo Cliente-Servidor, un servidor puede ser un PC, un supermini o un mainframe, debido a que iguala el trabajo del PC al servidor sin tener que recurrir a las ideas de "islas de automatización". El servidor puede atender a muchos clientes al mismo tiempo y compartir sus recursos.

8.- Dentro de las técnicas de programación en ambientes Cliente-Servidor tenemos cuatro de ellas que se han tratado en esta documentación, que son las siguientes: NetBIOS, TCP/IP, Named Pipes y RPC, cada una de las cuales brinda facilidades según el ambiente que quiera aplicarse.

9.- Con NetBIOS el proceso usuario no tiene acceso a ningún otro servicio, su ventaja principal es que permite a varios adaptadores LAN comunicarse con cada uno de los demás por medio de un PC intermediario, o de una PS/2 que esté

corriendo en un programa de interconexión de la red Token-Ring. Dicha ventaja capacita a las estaciones de trabajo de la red LAN comunicarse con otras estaciones de trabajo basadas en una red Token-Ring. Las aplicaciones pueden comunicarse con otras aplicaciones usando datagramas o sesiones.

10.- Named Pipes es un mecanismo de IPC de comunicación bidireccional, que efectúa intercambio de información entre cliente y servidor, mientras un proceso escribe, el otro recibe y viceversa. Bajo el Sistema Operativo OS/2, Named Pipes es el único IPC que permite que los procesos de comunicación sean distribuidos en diferentes computadores de la red. Named Pipes requiere de dos componentes: El Servidor, el cual crea el Named Pipe y el Cliente que se conecta al servidor.

11.- El protocolo TCP/IP provee de un API de programación conocido como la "Interfase Socket" que permiten a un programa comunicarse con otros programas a través de la red internet usando semánticas parecidas a la usada en archivos. En la perspectiva del programador , un socket esconde los detalles de la red, y son diferenciados por el dominio en el cual operan y por el tipo del socket.

12.- El Método RPC proporciona las rutinas que capacitan a los programas locales a ejecutar procedimientos en computadores remotos por medio de solicitudes y respuestas entre clientes y servidores. El Método RPC provee de la semántica usada en llamadas a funciones para la comunicación entre procesos locales o remotos, libera los mensajes utilizando un transporte que es el que permite la comunicación entre mecanismos RPC que están en diferentes computadores.

13.- RPC usa el modelo Cliente-Servidor en las aplicaciones distribuídas, el servidor ofrece sus servicios a la red donde el cliente puede accesar . Una aplicación puede ser un cliente y un servidor al mismo tiempo. En este modelo los servidores proveen de recursos, mientras que los clientes las consumen.

14.- RPC simplifica el desarrollo de aplicaciones distribuídas proveyendo de un modelo de programación muy familiar, escondiéndo los detalles de programación de la red, enfocándose en resolver los problemas direccionados por la aplicación.

BIBLIOGRAFIA

1.- [STEV 90] W. Richard Stevens, UNIX NETWORKI PROGRAMMING, PTR Prentice - Hall, Inc. A Simon & Schuster Company, (Englewood Cliffs, New Yersey, 1990), pp 224,225,238-251.

2.- [TANN 91] Andrew S. Tanenbaum, REDES DE ORDENADORES, Prentice Hall Hispanoamericana, S.A, (México, Englewood Cliffs, 1991), pp

3.- [ORF92] Robert Orfali/ Dan Harkey, CLIENT/SERVER PROGRAMMING WITH OS/2, 2nd ED, Library of Congress Cataloging-in-Publication Data, Van Rostrand Reinhold, (New York, N.Y., 1992), pp 5 -16, 161-172, 199 - 214, 323 -336.

4.- [STALL 94] William Stallings, Ph.D., DATA AND COMPUTER COMMUNICATIONS, Fourth Edition, Macmillan Publishing Company, (New York, 1994), pp 451 - 467.

5.- [CORB 91] John R. Corbin, THE ART OF DISTRIBUTED APPLICATIONS, Programming Techniques for Remote Procedure Calls, Springer-Verlag, Sun Microsystems, Inc, (Mountain View, 1991), pp 1 - 9, 65 - 74.

6.- [MCG 87] Osborne Mc Graw-Hill, C: THE COMPLETE REFERENCE, Copyright Mc Graw-Hill , Berkeley, (California, 1987) , pp .

7.- [HOW 89] W. David Schwaderer c/o Howard W. Sams & Company, C PROGRAMER'S GUIDE TO NetBIOS, James S. Hill - Albright Communicatios, Incorporated, (United States, 1989), pp 19 -35.

8.-[TECH 91] Technical Library , CONTROL PROGRAM PROGRAMMING REFERENCE VERSION 2.00 , Copyright International Business Machines Corporation 1986, 1991.

9.-[RYAN 90] Ralph Ryan, MICROSOFT LAN MANAGER A Programmer's Guide VERSION 2, Microsoft Press A Division of Corporation , (Redmond, Washington, 1990), pp 338 - 367.

Escuela Superior Politécnica del Litoral

Facultad de Ingeniería en Electricidad

"Técnicas de Programación en Ambientes Cliente - Servidor"

TESIS DE GRADO

Previa a la obtención del Título de

INGENIERO EN COMPUTACION

Presentada por:

Alexandra del Rocío Colombo Peña

Guayaquil - Ecuador

-: 1 9 9 4 :-

Escuela Superior Politécnica del Litoral

FACULTAD DE INGENIERIA EN ELECTRICIDAD

"TECNICAS DE PROGRAMACION EN AMBIENTES CLIENTE-SERVIDOR"

TESIS DE GRADO

PREVIA A LA OBTENCION DEL TITULO DE

INGENIERO EN COMPUTACION

PRESENTADA POR:

ALEXANDRA DEL ROCIO COLOMBO PEÑA

Guayaquil - Ecuador

1994

ING. A. ALTAMIRANO

DECANO

FAC. ING. ELECTRICA

ING. RIGOBERTO PUNIN P.

DIRECTOR DE

TESIS

ING. JAIME PUENTE

MIEMBRO PRINCIPAL DEL

TRIBUNAL

ING. SIXTO GARCIA

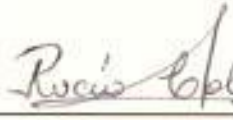
MIEMBRO PRINCIPAL DEL

TRIBUNAL

DECLARACION EXPRESA

"LA RESPONSABILIDAD POR LOS HECHOS,IDEAS Y DOCTRINAS EXPUESTOS EN ESTA TESIS, ME CORRESPONDEN EXCLUSIVAMENTE; Y EL PATRIMONIO DE LA MISMA, A LA ESCUELA SUPERIOR POLITECNICA DEL LITORAL".

(REGLAMENTO DE EXAMENES Y TITULOS PROFESIONALES DE LA ESPOL)



ALEXANDRA DEL ROCIO COLOMBO PEÑA

AGRADECIMIENTO

Quiero expresar mi agradecimiento primeramente al Señor Jesucristo por haber estado siempre a mi lado y por darme las fuerzas y el valor necesario para no rendirme en los momentos dificiles. Tambièn le doy las gracias a mis padres, tio y hermanas que a la distancia y de cerca siempre me alentaron y me brindaron la confianza de saber que ellos estaban conmigo, ademàs a la familia Flor que me abriò las puertas de su hogar desde el inicio de mi tesis y a mis amigos, entre ellos mi Director de tesis, tan especiales que con el apoyo incondicional me hicieron sentir que valia la pena el esfuerzo. A todos ellos gracias y que Dios los bendiga siempre.

DEDICATORIA

Dedico esta tesis a mis padres quienes merecen este titulo, pues es gracias a todo su esfuerzo que pude culminar mis estudios. Tambièn a la persona que me impulsò y ayudò a desarrollar el tema de mi tesis.

RESUMEN

El grado de conectividad e integración que las Redes de Computadoras ofrecen ha evolucionado lentamente comparado con la demanda de los servicios esperados. Por ejemplo: a) No hay una forma efectiva de explotar toda la capacidad de procesamiento de todos los componentes individuales de la red. b) Hay falta de compatibilidad de productos de red, como por ejemplo: EMAIL. Además, c) Hay Bases de datos que funcionan en ciertas redes mientras que en otras no.

Los pasos que se están dando para resolver este problema son los siguientes:

Crear un esquema de trabajo (Cliente-Servidor) abierto y flexible donde la unión e igualdad es la regla, es decir, procesamiento en forma cooperativa. Cliente - Servidor provee de la oportunidad de crear nuestra propia definición. Tanto **Clientes** como **Servidores** trabajan juntos para llevar a cabo una tarea, cada una de las cuales debe ir al computador adecuado.

La Comunicación **Cliente-Servidor** requiere mecanismos de transporte de requerimientos/Respuestas como lo son los siguientes:

- **NETBIOS**: Este mecanismo tiene un alcance limitado, se encuentra localizado a nivel de la capa de red del modelo OSI.
- **SOCKETS DE TCP/IP**: Este protocolo no está específicamente localizado en las capas física o de enlace. TCP/IP proporciona una gran cantidad de opciones de conectividad con dichas capas usando los mismos protocolos de ellas utilizan. La arquitectura del protocolo TCP/IP está basada en una forma de comunicación que envuelve 3 agentes: Procesos, Procesadores ("hosts") y Redes. Estos tres conceptos dan un principio fundamental del protocolo TCP/IP: La transferencia de información

hacia un proceso puede ser acoplada primeramente obteniendolo del host en el cual reside el proceso y entonces obtener el proceso dentro del "host".

- **NAMED-PIPES**: Este mecanismo se encuentra a nivel de la capa de presentación.
- **RPC**: se encuentra a nivel de la capa de aplicación.

Con los tres primeros mecanismos mencionados necesitamos conocer con qué estación nos queremos conectar, mientras que con el último que es RPC, se elimina esta limitación accesando a un servicio de directorio, el cual indica donde se encuentran los datos requeridos.

En este trabajo se va a hacer un estudio de todas las alternativas de solución al problema planteado.

Se desarrollarán ciertos prototipos, analizando las ventajas y desventajas de lo aquí propuesto.

INDICE GENERAL

Pag.

APENDICE A

FUNCIONES UTILIZADAS POR SOCKETS DE TCP/IP:

int sock_init():

Descripción: No tiene parámetros asociados con esta llamada. La llamada sock_init() inicializa las estructuras de datos del socket y chequear si INET.SYS está corriendo. Esta función debe ser llamada al inicio de cada programa que usa socket().

Valores Retornados: Retorna 0 si la llamada es exitosa, retorna 1 si ocurre un error.

****

int socket(domain, type, protocol)

Parámetro	**Descripción**
domain	La dirección domain solicitada. Debe ser AF_INET
type	El tipo de socket creado, sea SOCK_STREAM, SOCK_DGRAM o SOCK_RAW
protocol	El protocolo solicitado. Algunos valores posibles son 0, IPPROTO_UDP, o IPPROTO_TCP.
Descripción:	Esta llamada crea un punto final para la comunicación y retorna un descriptor del socket que representa el punto

final. Diferentes tipos de sockets proveen de diferentes sevicios de comunicación.

El parámetro domain especifica un dominio de comunicación dentro de cuya comunicación toma lugar. Este parámetro selecciona la dirección de la familia. La única familia soportada es AF_INET, la cual es el dominio de internet. Esta constante es definida en el archivo de cabecera <SYS\SOCKET.H>

****

ushort htons(a)

ushort a;

Parámetro	**Descripción**
a	El entero unsigned short puesto en el orden de byte de la red.
Descripción:	Esta llamada traslada un entero short desde el byte de orden del host hacia el orden del byte de la red.
Valores Retornados:	Retorna el entero corto trasladado.

****

int bind(s, name, namelen)

int s;

struct sockaddr *name;

int namelen;

Parámetros	**Descripción**
s	descriptor del socket retornado por la llamada previa socket()

name	puntero a la estructura sockaddr que contiene el nombre que está en el rango de s.
namelen	tamaño del nombre en bytes.

struct servent *getservbyname(name, proto)

char *name;

char *proto;

Parámetros	**Descripción**
name	puntero al nombre del servicio especificado
proto	puntero al protocolo especificado

****

int listen(s, backlog)

int s;

int backlog;

Parámetro	**Descripción**
s	El descriptor del socket.
backlog	Define la longitud máxima de la cola de conexiones pendientes.

****

int connect(s, name, namelen)

int s;

struct sockaddr *name;

int namelen;

Parámetros	**Descripción**
s	Descriptor del socket
name	Puntero a la dirección de la estructura del socket que contiene la dirección del socket.
namelen	tamaño de la dirección del socket apuntado por el nombre en bytes.

****

int accept(s, name, namelen)

int s;

struct sockaddr *name;

int *namelen;

Parámetros	**Descripción**
s	Descriptor del socket
name	Dirección del socket del cliente conectado que es llenadopor la llamada accept() antes de que retorne.
namelen	Indice al entero que contiene el tamaño en bytes de lo almacenado apuntado por el nombre.

****

int send(s, msg, len, flags)

int s;

char *msg;

int len;

int flags;

Parámetros	**Descripción**
s	descriptor del socket
msg	apunta al buffer que contiene el mensaje que se va a transmitir.
len	longitud del mensaje apuntado por el parámetro msg
flags	el parámetro flags es fijado por una ó más de las siguientes banderas:
	MSG_OOB Envía el dato out-of-band en el socket que soporta esta noción.
	MSG_DONTROUTE Esta opción se prende para la duración de una operación.

****

int ***recv(s, buf, len, flags)***

int s;

char *buff;

int len;

int flags;

Parámetros	**Descripción**
s	Descriptor del socket

buf	Puntero al buffer que recibe el dato
len	La longitud en bytes del buffer apuntado por el parámetro buf.
flags	El parámetro flag es fijado a una ó más de las siguientes banderas. Si es especificada más de una bandera., debe usarse el operador lógico OR.

****

int readv(s, iov, iovcnt)

int s;

struct iovec *iov;

int iovcnt;

Parámetros	**Descripción**
s	El descriptor del socket
iov	Puntero al arreglo de la estructura iovec
iovcnt	Número de buffers apuntado por el parámetro iov.

****

int writev(s, iov, iovcnt)

Parámetros	**Descripción**
s	El descriptor del socket
iov	Un puntero al arreglo de buffers iovec
iovcnt	El número de buffers apuntados por el parámetro iov.

Descripción: Esta llamada escribe el dato en el socket con descriptor s. El dato es obtenido desde los buffers especificados por iov[0]...iov[iovcnt-1]. La estructura iovec es definida en SYS\SOCKET.H.

****

int sendto(s, msg, len, flags, to, tolen)

int s;

char *msg;

int len;

int flags;

struct sockaddr *to;

int tolen;

Parámetros	**Descripción**
s	Descriptor del socket
msg	puntero al buffer que contiene el mensaje a transmitir
len	La longitud del mensaje en el buffer apuntado por el parámetro msg.
flags	Un parámetro que puede ser fijado a 0 ó MSG_DONTROUTE.
to	Dirección del target.
tolen	tamaño de la dirección apuntada por el parámetro.

Descripción: La llamada sendto() envía los paquetes al socket con descriptor s. Esta llamada es aplicada a cualquier socket de tipo datagrama, ya sea conectado ó desconectado.

****

int ***recvfrom(s, buf, len, flags, name, namelen)***

int s;

char *buf;

int len;

int flags;

struct sockaddr *name;

int *namelen;

Parámetro	**Descripción**
s	Descriptor del socket
buf	puntero al buffer que recibe el dato
len	longitud en bytes del buffer apuntado por el parámetro buf
flags	parámetro que puede ser fijado a 0 ó MSG_PEEK.

****

int ***select(s, noreads, nowrites, noexcepts, timeout)***

int *s;

int noreads;

int nowrites;

int noexcepts;

long timeout;

Parámetros	**Descripción**
s	Arreglo de número de sockets en el cual los números de sockets leídos están seguidos por los números de sockets escritos, seguidos por los números del socket.
noreads	número de sockets a ser chequeados para la lectura.
nowrites	número de sockets a ser chequeados para la lectura de lo escrito.
noexcepts	número de sockets a ser chequeados para las condiciones pendientes.
timeout	Máximo de intervalos, en milisegundos, para esperar por la selección a compeltar.

****

int ***ioctl(s, cmd, data, lendata)***

int s;

int cmd;

caddr_t data;

int lendata;

Parámetros	**Descripción**
s	Descriptor del socket
cmd	Comando a ejecutarse
data	Puntero al dato asociado con cmd
lendata	longitud de los datos en bytes.

****

int soclose(s)

Parámetro	**Descripción**
s	El descriptor del socket a descartar.

Descripción: Esta llamada cierra el socket asociado con el descriptor del socket s, y libera los recursos asignados al socket. Si s se refiere a una conexión TCP abierta, la conexión es cerrrada.

Valores Retornados: El valor 0 significa éxito, el valor -1 indica un error.

APENDICE B

MODULO SERVIDOR UTILIZADO PARA TRANSFERENCIA DE ARCHIVOS

PARA LOS PROGRAMAS EJEMPLO DE NETBIOS Y DE TCP/IP

FXSVR.C

```
/*************************************************************************/
/* NOMBRE DEL PROGRAMA: FXSVR - Un ejemplo del archivo de transferencia, modulo servidor/

/* DESCRIPCION DEL PROGRAMA: Este modulo es la porcion del servidor de un programa de  */
/*                           transferencia de archivos. Este modulo usa funciones de */
/*                           comunicacion modeladas despues de que las funciones del */
/*                           named pipe ejecuten la transferencia de archivos        */

/* INVOCACION DEL PROGRAMA: FXSVR                                                    */
/*************************************************************************/

#include  <process.h>
#include  <stdio.h>
#include  <stdlib.h>
#include  <string.h>
#include  <io.h>
#include  <fcntl.h>
#include  <sys\types.h>
#include  <sys\timeb.h>
#include  <sys\stat.h>


#define   INCL_DOSPROCESS
#include  <os2.h>

#include  "fxfer.h"
#include  "fxsvr.h"

/*-------------------------------------------------------------------------*/
/*  Rutina principal. Ejecuta la transferencia de archivo    */
/*-------------------------------------------------------------------------*/

VOID main(VOID)
{
  initialize_server();

        /*-------------------------------- ------------------------*/
        /* lazo que continuamente sirve a las solicitudes */
        /*-------------------------------------------------------------*/

        do  {
          if (get_service_request() == GOOD)
```

```
        {
          switch (packet_ptr->request.service)
          {
                case  FILE_SEND_TO_SERVER:    receive_file();
                                                    break;
                case  FILE_RECEIVE_FROM_SERVER:      send_file();
                                                          break;
                default:   printf("Solicitud para un servicio recibido desconocido.\n");
               } /* end switch  */
          terminate_service_request();
         } // end if
       } while (TRUE);  // enddo
} // end main
/*-------------------------------------------------------------------------------*/
/*  Inicializa el servidor desde el archivo de configuracion de datos  */
/*-------------------------------------------------------------------------------*/

VOID  initialize_server(VOID)
{
  printf("Inicializando el servidor.....\n");

  /*=============================================================*/
  /* Rutina de salida para cerrar el pipe si el  servidor termina   */
  /*=============================================================*/

  if (rc = DosExitList(1, (PFNEXITLIST) fxsvr_exit))
  {
    print_error(rc,"estableciendo DosExitList",_FILE_,_LINE_);
    exit(FAILED);
  }
  /*=============================================================*/
  /* Ejecuta la inicializacion  -get configuration data          */
  /*=============================================================*/
  if (rc = fx_Initialize(&packet_ptr, &filedata_packet_size))
  {
     print_error(rc,"during fx_Initialize",_FILE_,_LINE_);
     exit(FAILED);
  }

  /*=============================================================================*/
  /* Calcule la cabecera y tama¤ode datos del  paquete usado para la transferencia de archivos  */
  /*=============================================================================*/

  filedata_header_size = (sizeof(packet_ptr->filedata) -
                          sizeof(packet_ptr->filedata.Data));
  filedata_data_size   = filedata_packet_size - filedata_header_size;

  /*=============================================================*/
  /* Establecer el servidor                */
  /*=============================================================*/

  if (rc = fx_MakePipe())  {
     print_error(rc,"durando fx_MakePipe",_FILE_,_LINE_);
```

```
    exit(FAILED);
  }

  printf("Completa la Inicialización del servidor.\n\n");
} /* end initialize_server */



/*=============================================================*/
/*        recibe la solicitud del cliente                      */
/*=============================================================*/

USHORT get_service_request(VOID)
{
   /*-------------------------------------------------------------*/
   /* Esperar por que el solicitante abre el pipe */
   /*-------------------------------------------------------------*/

   printf("\nEsperando por la solicitud de servicio.....\n");
   if (fx_ConnectPipe() )
         return(FAILED);

   /*-------------------------------------------------------------------------*/
   /* Recibe los parametros del archivo de transferencia */
   /*-------------------------------------------------------------------------*/
   if (fx_Read(packet_ptr,
                  sizeof(packet_ptr -> request),
                  &PipeBytesRead))
         return(FAILED);
    } /* end get_service_request */


/*-----------------------------------------------------------------------------------------*/
/* Funcion del programa servidor la cual recibe un archivo usando      */
/* los siguientes pasos:  1) crea el archivo, 2) responde a la         */
/* solicitud, 3) recibe el archivo de datos                            */
/*-----------------------------------------------------------------------------------------*/

VOID receive_file(VOID)
{
   printf("Archivo envia la solicitud recibida.\n");

   /*=============================================================*/
   /*                        Crea el archivo                      */
   /*=============================================================*/

   FileHandle = open(packet_ptr->request.file_name,
                     O_BINARY | O_WRONLY| O_CREAT,
                     S_IREAD | S_IWRITE);
   if (FileHandle == -1) {
     print_error(errno,"abriendo un archivo",_FILE_,_LINE_);
```

```
  packet_ptr->response.rc = FAILED;
}
else packet_ptr->response.rc = GOOD;
packet_ptr ->filedata.packet_type = RESPONSE_PACKET;

/*=============================================================*/
/*                    Responde a la solicitud                  */
/*=============================================================*/
if (fx_Write(packet_ptr->response),
                &PipeBytesWritten) == 0)
       && (packet_ptr -> response.rc == 0))




/*=============================================================*/
/*   Recibe el archivo de datos del solicitante  */
/*=============================================================*/
 do {
       /*=======================================================*/
       /*   Recibe un bloque de datos              */
       /*=======================================================*/
        if (fx_Read(packet_ptr,filedata_packet_size,
                  &PipeBytesRead))
           break;

        /*=====================================================*/
        /*   Busca la localidad del archivo para el bloque */
        /*=====================================================*/
        if (-1L == lseek(FileHandle, packet_ptr->filedata.offset,SEEK_SET))
        {
         print_error(errno,"ejecutando LSEEK",_FILE_,_LINE_);
         break;
        }

        /*=====================================================*/
        /*   Escribe el bloque en el archivo          */
        /*=====================================================*/

         if (-1 == write(FileHandle, packet_ptr ->filedata.Data,
                          packet_ptr->filedata.length))
         {
           print_error(errno,"writing to  a file",_FILE_,_LINE_);
           break;
         }

 } while ((packet_ptr ->filedata.rc == 0)
          && (packet_ptr->filedata.length != 0)); /*enddo*/

/*=============================================================*/
/*   Cierra el archivo                  */
/*=============================================================*/
```

```
   close(FileHandle);
} /* end receive_file  */


/*-------------------------------------------------------------------------------*/
/*  Funcion del programa servidor la cual envia un archivo usando los
    siguientes pasos:  1)Abre el archivo, 2) responde a la solicitud,
    3) envia el archivo de datos                                                 */
/*-------------------------------------------------------------------------------*/

VOID send_file(VOID)
{
 printf("Archivo  recibe la solicitud.\n");

 /*-----------------------------------------------*/
 /*    abre un archivo existente                  */
 /*-----------------------------------------------*/
 FileHandle = open(packet_ptr ->request.file_name,
                   O_BINARY | O_RDONLY );
 if (FileHandle == -1)  {
   print_error(errno,"abriendo un archivo",_FILE_,_LINE_);
   packet_ptr->response.rc =FAILED;
 }
 else packet_ptr->response.rc =GOOD;
 packet_ptr ->filedata.packet_type = RESPONSE_PACKET;

 /*===============================================*/
 /*    Responde al solicitante                    */
 /*===============================================*/
 if ( (fx_Write(packet_ptr, sizeof(packet_ptr->response),
                &PipeBytesWritten) == 0)
        && (packet_ptr->response.rc == 0))
  {
    packet_ptr->filedata.offset = 0L;


    /*===================================================*/
    /*   Envia el archivo de datos al solicitante   */
    /*===================================================*/
    do {
          packet_ptr->filedata.packet_type =FILEDATA_PACKET;
          packet_ptr->filedata.length =0L;
          /*=================================================*/
          /*   Lee el bloque de datos desde el archivo    */
          /*=================================================*/
          do {
            packet_ptr-> filedata.packet_type =FILEDATA_PACKET;
            packet_pt-> filedata.length =0L;

            /*=================================================*/
            /*    lee el bloque de datos desde el archivo    */
            /*=================================================*/
```

```
        packet_ptr -> filedata.length = read(FileHandle,
                                        packet_ptr->filedata.Data,
                                        filedata_data_size);
        if (packet_ptr -> filedata.length == -1)
        {
          print_error(errno,"leyendo el archivo de datos",_FILE_,_LINE_);
          packet_ptr -> filedata.rc =-1;
          packet_ptr -> filedata.length =0;
        }

        /*=================================================*/
        /*   Crea el archivo                               */
        /*=================================================*/
        if ( fx_Write(packet_ptr, filedata_header_size +
                        packet_ptr->filedata.length,
                        &PipeBytesWritten)
              | packet_ptr->filedata.rc)
              break;
         packet_ptr->filedata.offset =  packet_ptr->filedata.offset +
                        packet_ptr->filedata.length;
         } while ((packet_ptr ->filedata.rc == 0) &&
                  (packet_ptr->filedata.length != 0 ) ); /*enddo */
      } // end if

      /*=================================================*/
      /*   Cierra el archivo                             */
      /*=================================================*/
      close(FileHandle);
} // end send_file

/*=================================================*/
/*  Funcion para terminar una solicitud de servicio  */
/*=================================================*/

USHORT terminate_service_request(VOID)
{
   /*=================================================*/
   /*   Recibe la solicitud de desconexi¢n            */
   /*=================================================*/
    if (fx_Read(packet_ptr, filedata_packet_size,
                  &PipeBytesRead))
    {
        print_error(rc,"recibiendo la solicitud de desconexi¢n",
                    _FILE_,_LINE_);
        return(FAILED);
    }

    if (!( ( packet_ptr->request.packet_type == REQUEST_PACKET) &&
        (packet_ptr -> request.service   == DISCONNECT)))
        printf("Protocolo incorrecto. Revise la solicitud de desconexi¢n.\n");

    /*=================================================*/
    /*   Termina la sesi¢n del pipe                    */
```

```
    /*=========================================================*/
    if (fx_DisconnectPipe())
    {
        print_error(rc,"durante fx_DisconnectPipe",_FILE_,_LINE_);
        return(FAILED);
    }
    printf("Solicitud de servicio completada.\n");
} /* end server_send_receive_file */

/*=========================================================*/
/*  Error Display Routine                  */
/*=========================================================*/

VOID  print_error(ULONG Error, PCHAR msg, PCHAR file, USHORT line)
{
   printf("Error %lu  (Ox%x) detectado mientras %s \ en la linea %lu en
          el archivo %s.\n", Error, Error, msg, line, file);
} // end print_error

/*=========================================================*/
/*   Rutina de salida                  */
/*=========================================================*/
VOID  APIENTRY fxsvr_exit(ULONG exit_type)
{
  if (fx_Close())
    printf("Error encontrado mientras terminaba el archivo de
          transferencia del servidor.");
  else
    printf("El archivo de transferencia del servidor ha terminado normalmente\n");

  DosExitList(3, (void far *) 0);
} // fxsvr_exit
```

```
/*************************************************************************/
/*  INCLUDE NAME:  FXSVR -declarations  for FXSVR program        */
/*************************************************************************/

/*-----------------------------------------------------------*/
/*  Forward declarations of functions in this module          */
/*-----------------------------------------------------------*/

VOID   initialize_server(VOID);
USHORT get_service_request(VOID);
USHORT terminate_service_request(VOID);
VOID   send_file(VOID);
VOID   receive_fie(VOID);
VOID   print_error(ULONG, PCHAR, PCHAR, USHORT);
VOID  APIENTRY  xfsvr_exit(ULONG);
```

```
/*-------------------------------------------------------------------*/
/* Variables globales para este modulo                               */
/*-------------------------------------------------------------------*/

        /*------------------------------------- */
        /* Variables del archivo de transferencia */
        /*------------------------------------- */

PACKET  *packet_ptr;
USHORT   filedata_packet_size;
USHORT   filedata_data_size;
USHORT   filedata_header_size;

USHORT   PipeBytesRead;
USHORT   PipebytesWritten;
ULONG    FileHandle;


        /*------------------------------------- */
        /* Variables de miscelanea              */
        /*------------------------------------- */
ULONG    rc;



/*********************************************************************/
/* INCLUDE NAME: FXSVR -declarations for FXSVR program               */
/*********************************************************************/

/*-------------------------------------------------------------------*/
/* Forward declarations of functions in this module                  */
/*-------------------------------------------------------------------*/

VOID   initialize_server(VOID);
USHORT get_service_request(VOID);
USHORT terminate_service_request(VOID);
VOID   send_file(VOID);
VOID   receive_fie(VOID);
VOID   print_error(ULONG, PCHAR, PCHAR, USHORT);
VOID  APIENTRY  xfsvr_exit(ULONG);

/*-------------------------------------------------------------------*/
/* Variables globales para este modulo                               */
/*-------------------------------------------------------------------*/

        /*------------------------------------- */
        /* Variables del archivo de transferencia */
        /*------------------------------------- */

PACKET  *packet_ptr;
USHORT   filedata_packet_size;
USHORT   filedata_data_size;
USHORT   filedata_header_size;
```

```
USHORT   PipeBytesRead;
USHORT   PipebytesWritten;
ULONG    FileHandle;

         /*------------------------------------ */
         /* Variables de miscelanea           */
         /*------------------------------------ */
ULONG    rc;


/*********************************************************************/
/* INCLUDE NAME: FXSVR -declarations for FXSVR program        */
/*********************************************************************/

/*-------------------------------------------------------------*/
/* Forward declarations of functions in this module          */
/*-------------------------------------------------------------*/

VOID   initialize_server(VOID);
USHORT get_service_request(VOID);
USHORT terminate_service_request(VOID);
VOID   send_file(VOID);
VOID   receive_fie(VOID);
VOID   print_error(ULONG, PCHAR, PCHAR, USHORT);
VOID  APIENTRY  xfsvr_exit(ULONG);

/*-------------------------------------------------------------*/
/* Variables globales para este modulo                       */
/*-------------------------------------------------------------*/

         /*------------------------------------ */
         /* Variables del archivo de transferencia */
         /*------------------------------------ */

PACKET   *packet_ptr;
USHORT   filedata_packet_size;
USHORT   filedata_data_size;
USHORT   filedata_header_size;

USHORT   PipeBytesRead;
USHORT   PipebytesWritten;
ULONG    FileHandle;


         /*------------------------------------ */
         /* Variables de miscelanea           */
         /*------------------------------------ */
ULONG    rc;
```

```
/*********************************************************************/
/* INCLUDE NAME: FXSVR -declarations for FXSVR program        */
/*********************************************************************/

/*-------------------------------------------------------------------*/
/* Forward declarations of functions in this module    */
/*-------------------------------------------------------------------*/

VOID   initialize_server(VOID);
USHORT get_service_request(VOID);
USHORT terminate_service_request(VOID);
VOID   send_file(VOID);
VOID   receive_fie(VOID);
VOID   print_error(ULONG, PCHAR, PCHAR, USHORT);
VOID  APIENTRY  xfsvr_exit(ULONG);

/*-------------------------------------------------------------------*/
/* Variables globales para este modulo                */
/*-------------------------------------------------------------------*/

            /*------------------------------------------------ */
            /* Variables del archivo de transferencia */
            /*------------------------------------------------ */

PACKET  *packet_ptr;
USHORT   filedata_packet_size;
USHORT   filedata_data_size;
USHORT   filedata_header_size;

USHORT   PipeBytesRead;
USHORT   PipebytesWritten;
ULONG    FileHandle;


            /*---------------------------------------- */
            /* Variables de miscelanea       */
            /*---------------------------------------- */
ULONG    rc;
```

MODULO LLAMADO POR EL PROGRAMA FXSVR.C

FXSVR.H

```
/*********************************************************************/
/* INCLUDE NAME: FXSVR -declarations for FXSVR program        */
/*********************************************************************/

/*-------------------------------------------------------------------*/
/* Forward declarations of functions in this module    */
/*-------------------------------------------------------------------*/

VOID   initialize_server(VOID);
USHORT get_service_request(VOID);
USHORT terminate_service_request(VOID);
VOID   send_file(VOID);
VOID   receive_fie(VOID);
VOID   print_error(ULONG, PCHAR, PCHAR, USHORT);
VOID  APIENTRY  xfsvr_exit(ULONG);

/*-------------------------------------------------------------------*/
/* Variables globales para este modulo                */
/*-------------------------------------------------------------------*/

            /*-------------------------------------------------- */
            /* Variables del archivo de transferencia */
            /*-------------------------------------------------- */

PACKET  *packet_ptr;
USHORT   filedata_packet_size;
USHORT   filedata_data_size;
USHORT   filedata_header_size;

USHORT   PipeBytesRead;
USHORT   PipebytesWritten;
ULONG    FileHandle;


            /*-------------------------------------- */
            /* Variables de miscelanea       */
            /*-------------------------------------- */
ULONG    rc;
```

PROGRAMA CLIENTE UTILIZADO PARA SOLICITAR EL SERVICIO DE TRANSFERENCIA DE ARCHIVOS.

// FXREQ.C LISTADO

```
/*********************************************************************/
/* NOMBRE DEL PROGRAMA:  FXREQ  - Mòdulo del Cliente                 */
/* DESCRIPCION:   Este mòdulo es la porciòn del solicitante          */
/* de un programa del archivo de transferencia.  Este mòdulo         */
/* usa funciones modeladas de comunicaciòn despuès de que las        */
/* funciones del named pipe ejecuten el archivo de transferencia     */
/* Estos servicios son implementados en mòdulos separados usando     */
/* NETBIOS, APPC, o comunicaciòn entre named pipes para comunicaciòn  */
/* de PC - a - PC                                                    */

/* INVOCACION DEL PROGRAMA:

   FXREQ direcci¢n del nombre del archivo.
   donde:
        la direcci¢n es SEND o RECV.

        nombre del archivo (file-name) es el nombre del archivo a
        transferir.                                                  */

/*********************************************************************/

#include <process.h>      /* For exit()   */
#include <stdio.h>        /* Para printf() & NULL   */
#include <stdlib.h>
#include <string.h>
#include <sys\types.h>
#include <sys\timeb.h>
#include <sys\stat.h>
#include <io.h>
#include <fcntl.h>
#include <sys\types.h>

#include <os2.h>
#include <fxfer.h>
#include <fxreq.h>

/* ------------------------------------------------------------------------*/
/* Rutina Principal, ejecuta la transferencia de un archivo y calcula */
/* estadisticas.                                                          */
/* ------------------------------------------------------------------------*/

VOID main( SHORT argc, PCHAR argv[], PCHAR envp[])
{
```

```
/* ------------------------------------------------------------ */
/*  Obtiene los argumentos de la linea de comandos */
/* ------------------------------------------------------------ */

if (get_args(argc, argv) ) {
   print_help();
   exit (FAILED);
}

/* ------------------------------------------------ */
/*  Inicializa al solicitante                */
/* ------------------------------------------------ */

initialize_requester();

/* ------------------------------------------------ */
/*  Servicio del solicitante                 */
/* ------------------------------------------------ */

switch (service)
{
  case FILE_SEND_TO_SERVER:      open_file();
                                 request_file_transfer();
                                 send_file();
                                 break;
  case FILE_RECEIVE_FROM_SERVER: open_file();
                                 request_file_transfer();
                                 receive_file();
                                 break;
 } /* End Switch */

 /* ------------------------------------------------ */
 /*  Termina la solicitud de servicio            */
 /* ------------------------------------------------ */
 terminate_service_request();

 /* ------------------------------------------------ */
 /*  Imprime las estadisticas                 */
 /* ------------------------------------------------ */
 printf("Archivo de transferencia completo. \n");
 file_size = get_file_size( file_name );
 print_statistics();

 exit(GOOD);

} /* End main */

 /* ------------------------------------------------------------------------------*/
 /*  Inicializa las variables del programa desde la linea de comandos */
 /* ------------------------------------------------------------------------------ */

 USHORT get_args(SHORT ARGC, PCHAR argv[]);
 {
```

```
    /* ------------------------------------------------ */
    /*  Existe el n£mero exacto de argumentos?        */
    /* ------------------------------------------------ */

    if ( argc != 3)
          return (FAILED);

    /* ------------------------------------------------ */
    /*  Envia o Recibe                                */
    /* ------------------------------------------------ */
    if (strcmp("SEND", strupr(argv[1]) ) == 0)
          service = FILE_SEND_TO_SERVER;
    else
          if ( strcmp ("RECV", strupr(argv[1] )) == 0)
             service = FILE_RECEIVE_FROM_SERVER;
          else
          return (FAILED);

    /* ------------------------------------------------ */
    /*  Obtiene el nombre del archivo                 */
    /* ------------------------------------------------ */
    strcpy( file_name, argv[2] );

    /* ------------------------------------------------ */
    /*  Todo esta bien!                               */
    /* ------------------------------------------------ */
    return (GOOD);
} /* end_args */

/* ------------------------------------------------------------------------------------------------ */
/*  Inicializa al solicitante desde el archivo de configuraciòn  de los  datos  */
/* ------------------------------------------------------------------------------------------------ */

VOID  initializate_requester(VOID)
{
  /* ------------------------------------------------------------------------------------------------ */
  /* ejecuta la inicializaciòn   obtiene la configuraciòn de datos
     ubica un buffer para la comunicaciòn el puntero al buffer y el tamaño son retornados        */
  /* ------------------------------------------------------------------------------------------------*/

  printf("Inicializando .....\n");
  open_bgn_time = save_time();
  if (fx_Initializate( &packet_ptr, &filedata_packet_size )) {
    printf ("Error en Inicializando.\n");
    exit(FAILED);
  }

  /* ------------------------------------------------------ */
  /*  Calcula la cabecera y tama□o de datos del paquete usado
      para el archivo de transferencia                    */
  /* ------------------------------------------------------ */
  filedata_header_size = (sizeof (packet_ptr -> filedata) -
                          sizeof (packet_ptr -> filedata.Data) );
  filedata_data_size = filedata_packet_size - filedata_header_size;
```

```
 /* ------------------------------------------------ */
 /*  Abre el pipe                                    */
 /* ------------------------------------------------ */
 if (fx_Open() )
 {
   printf ("Error al abrir el Pipe de comunicaci¢n. \n");
   exit (FAILED);
 }

 printf("Inicializaci¢n completa. \n");
} /* end initialize_requester */
```

```
/* ------------------------------------------------------------------------------------*/
/* Funcion para abrir un archivo existente o crea una nueva linea */
/* ------------------------------------------------------------------------------------*/
VOID  open_file( VOID);
{
  /* ------------------------------- */
  /*  Abre el archivo                */
  /* ------------------------------- */
  printf("Empezando la transferencia del archivo .....\n");
  if (service == FILE_SEND_TO_SERVER )  {
        if ( -1 == (FileHandle = open(file_name, O_BINARY | O_RDONLY ) ) ) {
          print_error(errno, "opening a file", _FILE_,_LINE_);
          exit (FAILED);
        }
   }
   else  {
        if ( -1 == (FileHandle = open(filename, O_BINARY | O_WRONLY | O_CREAT, S_IREAD |
S_IWRITE)) ) {
          print_error(errno, "opening a file",_FILE_,_LINE_);
          exit(FAILED);
        }
   }
} /* end open_file */

/* ------------------------------------------------------------------------------------ */
/*  Funciòn para solicitar el servicio de transferencia de archivo */
/* ------------------------------------------------------------------------------------ */
VOID request_file_transfer(VOID)
{
  /* ------------------------------------------------ */
  /* Envia los parametros del archivo de transferencia  */
  /* y obtiene una respuesta                          */
  /* ------------------------------------------------ */
  xfer_bgn_time = save_time();
  packet_ptr -> request.packet_type = REQUEST_PACKET;
  packet_ptr -> request.service     = service;
  strcpy (packet_ptr -> request.file_name, file_name);
```

```
  if (fx_Transact(packet_ptr,
                  sizeof(packet_ptr -> request),
                  packet_ptr,
                  sizeof(packet_ptr ->response),
                  &PipeBytesRead ) )
  {  printf("Error al inicio de la transferencia de archivos.\n");
     exit(FAILED);
  }

  if (packet_ptr -> response.rc !=0 ) {
     printf("Server Error cuando inicia la transferencia de archivos\n");
     exit(FAILED);
  }
} /* end request_file_transfer  */
```

```
/* ------------------------------------------------------------------------*/
/*  Funcion que recibe un archivo desde el servidor        */
/* ------------------------------------------------------------------------*/

VOID receive_file(VOID)
{
  /* ------------------------------------------------ */
  /*  Recibe el archivo de datos del solicitante     */
  /* ------------------------------------------------ */

  do {
    /* ------------------------------------------------ */
    /*  Recibe un bloque  del archivo de datos        */
    /* ------------------------------------------------ */
    if (fx_Read(packet_ptr,
                  filedata_packet_size,
                  &PipeBytesRead) )
          break;

    /* ------------------------------------------------ */
    /*  Seek to file location for the block           */
    /* ------------------------------------------------ */
    if (-1L == lseek(FileHandle,
                     packet_ptr -> filedata.offset,
                     SEEK_SET ))
    {
          print_error(errno,"perfoming a LSEEK", _FILE_,_LINE_);
          break;
    }
```

```
    /* ------------------------------------------------ */
    /*  Escribe el bloque en el archivo                 */
    /* ------------------------------------------------ */
    if (-1 == write(FileHandle,
                    packet_ptr -> filedata.offset,
                    SEEK_SET ))
    {
        print_error (errno,"writing to a file",_FILE_,_LINE_);
        break;
    }
} while ( ( packet_ptr -> filedata.rc == 0)
          && (packet_ptr -> filedata.length !=0 )); // enddo

/* ------------------------------------------------ */
/*  Termina la transferencia del archivo            */
/* ------------------------------------------------ */

close (FileHandle);

} /* end receive_file */

/* ------------------------------------------------------------ */
/*  Funciòn para enviar un archivo al servidor                  */
/* ------------------------------------------------------------ */
VOID send_file (VOID)
{
  /* ------------------------------------ */
  /*  Envia el archivo de datos al servidor */
  /* ------------------------------------ */
  packet_ptr ->filedata.offset = 0;
  do {
    /* ------------------------------------ */
    /*  Lee el bloque del archivo de datos  */
    /* ------------------------------------ */
    packet_ptr -> filedata.length = read ( FileHandle,
                                           packet_ptr-> filedata.Data,
                                           filedata_data_size);
    if (packet_ptr -> filedata.length == -1)  {
        print_error( errno, "reading file data",_FILE_,_LINE_);
        packet_ptr -> filedata.rc = FAILED;
        packet_ptr -> filedata.lenght = 0;
    }

    /* ------------------------------------------------ */
    /*  Envia el bloque del archivo de datos, lo borra  */
    /*  del lazo si envia un error al archivo           */
    /* ------------------------------------------------ */
    packet_ptr -> filedata.packet_type = FILEDATA_PACKET;
    if (fx_Write(packet_ptr, filedata_header_size +
                    packet_ptr ->filedata.length,
                    &PipeBytesWritten) |  packet_ptr -> filedata.rc)
        break;
    packet_ptr -> filedata.offset = packet_ptr -> filedata.offset +
                                    packet_ptr -> filedata.length;
```

```
  } while (( packet_ptr -> filedata.rc == 0) &&
           ( packet_ptr -> filedata.length != 0) );

  /* ------------------------------------------------*/
  /*  Termina la transferencia de archivos */
  /* ------------------------------------------------*/
  close(FileHandle);
} /*end send_file */

/* ------------------------------------------------------------------*/
/* Cierra el archivo, termina la solicitud de servicio  */
/* cierra el comm. pipe                                              */
/* ------------------------------------------------------------------*/

VOID terminate_service_request(VOID)
{
   /* ------------------------------- */
   /* Envia la solicitud de desconexión    */
   /* ------------------------------- */
   packet_ptr->request.packet_type = REQUEST_PACKET;
   packet_ptr->request.service     = DISCONNECT;
   if (fx_Write(packet_ptr, sizeof(packet_ptr->request),
                      &PipeBytesRead ))
   {
     printf ("Error al termino del servicio.",_FILE_,_LINE_);
     exit(FAILED);
   }



   /* ------------------------------- */
   /*      Termina la sesion del pipe      */
   /* ------------------------------- */
   close_bgn_time = save_time();
   if (fx_Close() ) {
          printf("Error al cerrar la comunicacion del pipe.");
          exit(FAILED);
   }

   end_time = save_time();
} /* end terminate_service_request */


/* ------------------------------------------------------- */
/*  Start timing an event.                                 */
/* ------------------------------------------------------- */

double save_time(VOID)
{
 ftime(&timebuff);
 return( (double)timebuff.time+((double)timebuff.millitm)/(double)1000);
} /*end save_time */
```

```
/* ------------------------------------------------------------ */
/* Print Statistics                                             */
/* ------------------------------------------------------------ */
VOID print_statistics(VOID)
{
  if (service == FILE_SEND_TO_SERVER)
    printf("\n%s recibido desde el servidor.\n",file_name);

  printf("El tiempo de apertura del enlace de comunicaci¢n fue %f segundos.\n",
          xfer_bgn_time-open_bgn_time);
  printf("%ld bytes transferidos en %f segundos.\n",
          file_size,close_bgn_time-xfer_bgn_time);
  printf("El rango promedio de transferencia fue %.2f kilobytes/segundos.\n",
          file_size/((close_bgn_time-xfer_bgn_time)*1000));
  printf("El tiempo de cerrar el enlace de comunicaci¢n fue %f segundos.\n",
          end_time-close_bgn_time);
} /* end print_statistics */

/* ------------------------------------------------------------ */
/*        Imprime la ayuda                                      */
/* ------------------------------------------------------------ */
VOID print_help(VOID)
{
 printf("\nComando de invocaci¢n incorrecto. Ingrese como sigue:\n");
 printf("\n");
 printf("\nFXR direccion del file-name                    \n");
 printf("                                          \n");
 printf("  donde:                                    \n");
 printf("     Direccion es SEND o RECV                    \n");
 printf("                                          \n");
 printf("     file-name es el nombre del archivo a transferir. \n");
 printf("                                          \n");
 printf("     NOTE: DDL para el protocolo que desea probar debe\n");
 printf("          ser copiado al PROTOCOL.DLL en un       \n");
 printf("          subdirectorio en su LIBPATH             \n");
 printf("\n\n");
} /* end print_help */

/* ------------------------------------------------------------ */
/* Obtiene el tama¤o del archivo                                */
/* ------------------------------------------------------------ */
ULONG get_file_size(PCHAR name)
{
  SHORT  fhandle;
  ULONG  size;

  if (-1 == (fhandle = open(name, O_BINARY | O_RDONLY ))) {
    print_error(errno, "abriendo un archivo", _FILE_, _LINE_);
    return (FAILED);
  }
  size = filelength(fhandle);
  close(fhandle);
  return(size);
} /* end get_file_size */
```

```
/* ------------------------------------------------------------ */
/* Error Display Routine                                        */
/* ------------------------------------------------------------ */
VOID  print_error(USHORT Error, PCHAR msg, PCHAR file, USHORT line)
{
  printf("Error %u (Ox%x) detectado mientras %s  hacia la linea %u en el archivo %s.\n",
          Error, Error, msg, line, file);
} /* end print_error  */
```

MODULO LLAMADO POR EL PROGRAMA FXREQ.C

FXREQ.H

```
/*********************************************************************/
/*    INCLUDE NAME:  FXREQ - declaraciones del programa FXREQ    */
/*********************************************************************/
/* ------------------------------------------------------------------ */
/* Forward declarations of functions in this module          */
/* ------------------------------------------------------------------ */

USHORT   get_args(SHORT, PCHAR *);
VOID     initialize_requester(VOID);
VOID     open_file(VOID);
VOID     request_file_transfer(VOID);
VOID     receive_file(VOID);
VOID     send_file(VOID);
VOID     terminate_service_request(VOID);
double   save_time(VOID);
VOID     print_statistics(VOID);
VOID     print_help(VOID);
ULONG    get_file_size(PCHAR);
VOID     print_error(USHORT, PCHAR, PCHAR, USHORT);

/* ------------------------------------------------------------------ */
/*         Variables globales para este modulo                */
/* ------------------------------------------------------------------ */
            /*-------------------------------------------------*/
            /* command line arguments variables   */
            /*-------------------------------------------------*/

USHORT      service;
CHAR        file_name[128];

            /*-------------------------------------*/
            /*       timing variables          */
            /*-------------------------------------*/

double  open_bgn_time    = 0.0;   // Empieza fx_Open
double  xfer_bgn_time    = 0.0;    // Empieza el inicio de la trasnsferencia de datos
double  close_bgn_time   = 0.0;   // Empieza fx_Close
double  end_time         = 0.0;    // Completa fx_Close
struct  timeb  timebuff;

            /*-------------------------------------*/
            /*       file size variable        */
            /*-------------------------------------*/
ULONG    file_size;

            /*-------------------------------------*/
            /*    file transfer  variables       */
            /*-------------------------------------*/

PACKET   *packet_ptr;            // puntero al paquete
```

```
USHORT    filedata_packet_size;     // tamano del paquete para el archivo de transferencia
USHORT    filedata_data_size;       // tama¤o del area de datos para el archivo de transf.
USHORT    filedata_header_size;     // tama¤o  de la cabecera del archivo xfer.


USHORT    PipeBytesRead;            // numero de bytes leidos desde el pipe
USHORT    PipeBytesWritten;         // numero de bytes escritos al pipe
ULONG     FileHandle;               // handle para el archivo

          /*-------------------------------------*/
          /* misc                                */
          /*-------------------------------------*/
ULONG      rc;       // variable que recibe los codigos retornados
```

PROGRAMA CLIENTE - SERVIDOR UTILIZANDO LA TECNICA DE PROGRAMACION NETBIOS

```
/*************************************************************************/
/* NOMBRE DEL MODULO:        FXNETB - Comunicacion NetBIOS        */
/* DESCRIPCION DEL PROGRAMA: Este modulo es una parte de los
   programas del archivo de transferencia FX. Hace referencia al programa modulo FXREQ.C.
   Este modulo implementa procedimientos del archivo de transferencia
   fx_* usando NETBIOS OS/2.                                              */
/*************************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include <os2.h>
#include "fxfer.h"

/*-----------------------------------------------------------------*/
/* defines                                                         */
/*-----------------------------------------------------------------*/

        /*-------------------------------------------------*/
        /* defines para Comandos del codigo NCB */
        /*-------------------------------------------------*/
#define NB_CALL_WAIT            0x0010
#define NB_LISTEN_WAIT          0x0011
#define NB_HANG_UP_WAIT         0x0012
#define NB_SEND_WAIT            0x0014
#define NB_RECEIVE_WAIT         0x0015
#define NB_ADD_NAME_WAIT        0x0030
#define NB_RESET_WAIT           0x0032
        /*-------------------------------------------------*/
        /* defines para Recursos de la sesion */
        /*-------------------------------------------------*/
#define LANA            0
#define NET_LSN         2
#define NET_NCB         4
#define NET_NAMES       2
#define REQUEST_RESOURCES  0
#define RELEASE_RESOURCES  1

        /*-------------------------------------------------*/
        /* defines para timeout de NETBIOS    */
        /*-------------------------------------------------*/

#define RECV_TIMEOUT    20
#define SEND_TIMEOUT    20

/*-----------------------------------------------------------------*/
/* typedef para estructura NCB                                     */
/*-----------------------------------------------------------------*/
```

```
        /*-------------------------------------------------------*/
        /*  estructura general para comandos NCB */
        /*-------------------------------------------------------*/
typedef struct
{
   BYTE   command;
   BYTE   retcode;
   BYTE   lsn;
   BYTE   num;
   PBYTE  _Seg16 buffer_address;
   USHORT lenght;
   BYTE   callname[16];
   BYTE   name[16];
   BYTE   rto;
   BYTE   sto;
   PBYTE  _Seg16 post_address;
   BYTE   lana_num;
   BYTE   cmd_cplt;
   BYTE   reserve[14];
} CMD_NCB;

typedef struct
{
   BYTE   command;
   BYTE   retcode;
   BYTE   lsn;
   BYTE   num;
   PBYTE  _Seg16 add_name_address;
   USHORT length;
   BYTE   req_sessions;
   BYTE   req_commands;
   BYTE   req_names;
   BYTE   req_name_one;
   BYTE   not_used2[12];
   BYTE   act_sessions;

   BYTE   act_commands;
   BYTE   act_names;
   BYTE   act_name_one;
   BYTE   not_used3[4];
   BYTE   load_sesion;
   BYTE   load_commands;
   BYTE   load_names;
   BYTE   load_stations;
   BYTE   not_used4[2];
   BYTE   load_remote_names;
   BYTE   not_used5[5];
   USHORT dd_id;
   BYTE   lana_num;
   BYTE   not_used6;
   BYTE   reserve[14];
} RESET_NCB;

tyepedef union
```

```
{
  CMD_NCB   cmd_ncb;
  RESET_NCB reset_ncb;
} NCB;

/*-------------------------------------------------------------------*/
/*  forward declaration for functions in this module   */
/*-------------------------------------------------------------------*/

          /*-----------------------------------------*/
          /*  netbios helper functions          */
          /*-----------------------------------------*/

BYTE   netbios_reset(BYTE);
BYTE   netbios_reset(BYTE);
BYTE   netbios_add_name(PCHAR);
BYTE   netbios_call(PCHAR, PCHAR, PBYTE);
BYTE   netbios_listen(PCHAR, PCHAR, PBYTE);
BYTE   netbios_hang_up(BYTE);
BYTE   netbios_send(PVOID, USHORT, BYTE);
BYTE   netbios_recv(PVOID, USHORT, BYTE, PUSHORT);

          /*-------------------------------------*/
          /*    error display function        */
          /*-------------------------------------*/

VOID   print_error(USHORT, PCHAR, PCHAR, USHORT);

          /*---------------------------------------------*/
          /*  netbios dinamic link function       */
          /*---------------------------------------------*/

extern  unsigned NETBIOS (NCB *);
#pragma  linkage(NETBIOS, far16 pascal)

/*-------------------------------------------------------------------------*/
/* Configuration variables, initialized after fx_Initialize()        */
/* function call                                                        */
/*-------------------------------------------------------------------------*/

CHAR    PipeName[128];
USHORT  FileBufferSize;

/*-------------------------------------------------------------------*/
/*  global  variables for this module                    */
/*-------------------------------------------------------------------*/

BYTE    lsn;
USHORT  rc;
CHAR    server_name[17];
CHAR    requester_name[17];
USHORT  server;
NCB     ncb;
```

```
/*-------------------------------------------------------------------*/
/*  Error Display Routine                                            */
/*-------------------------------------------------------------------*/

VOID  print_error(USHORT  Error, PCHAR msg, PCHAR file, USHORT line)
{
   printf("NETBIOS Error %u (0x%x) detectado mientras %s en la linea %u en el archivo %s.\n",
           Error, Error, msg, line, file);
} /* end print_error  */

/*-------------------------------------------------------------------------------------*/
/*  Esta funcion lee los parametros especificos de la comunicacion   */
/*  y ubica un buffer  para la comunicacion.  Un puntero al buffer    */
/*  y el tamaño del buffer es retornado                                */
/*-------------------------------------------------------------------------------------*/

ULONG  fx_Initialize(PPACKET *buffer_p, PUSHORT size)
{
 FILE    *config_file;
 CHAR    line[80];
 PPACKET  _Seg16 bufferp;

 printf("Leyendo los parametros de configuración.\n");

 /*-----------------------------------------------*/
 /* abre el archivo de configuaracion */
 /*-----------------------------------------------*/

 if (NULL == (config_file = fopen("fxnetb.cfg","r") ) )  {
    print_error(rc,"abriendo el archivo de configuracion",_FILE_,_LINE_);
    return(FAILED);
  }
  else  {
  /*-------------------------------------------------*/
  /* Lee los paramteros de configuracion.
     Los parametros se esperan que sean:
          - en la forma <name = value>
          - si el valor es un string encerrado
            en parentesis.
          - en el orden leido abajo          */
  /*-------------------------------------------------*/
     fgets(line, sizeof(line), config_file);
     if (sscanf(line,"PIPE_NAME =%s", PipeName) != 1) {
           print_error(rc,"Leyendo el PipeName desde la configuracion",
                       _FILE_,_LINE_);
           return(FAILED);
          }
          printf("PIPE_NAME = %s \n", PipeName);

          fgets(line, sizeof(line), config_file);
          if (sscanf(line, "FILE_BUFFER_SIZE = %u", &FileBufferSize) != 1)
          {
             print_error(rc,"leyendo FileBufferSize de la configuracion",
                         _FILE_, _LINE_);
```

```
            return(FAILED);
          }
          printf("FILE_BUFFER_SIZE = %u \n",FileBufferSize);
    } // end if

fclose(config_file);

/*-----------------------------------------------*/
/* Inicializa server_name y la variable */
/* requester_name para usarla en los
   comandos ncb                                  */
/*-----------------------------------------------*/

strcat(PipeName, "               "); // make pipe name >= 16 char long

strcpy(server_name, PipeName, 16);
strcpy(&server_name[12], ".SVR", 4);

strcpy(requester_name, PipeName,16);
strcpy(&requester_name[12], ".REQ",4);

/*--------------------------------------*/
/* Ubica el buffer, retorna el puntero
   y el tamaño                          */
/*--------------------------------------*/
   bufferp = (malloc)(FileBufferSize);
   if (bufferp == NULL)
   {
     print_error(rc,"ubicando el archivo buffer",_FILE_,_LINE_);
     return(FAILED);
   }
   *buffer_p = buffer_p;
   *size     = FileBufferSize;
    return(GOOD);
} // end fx_Initialize


/*-------------------------------------------------------------------*/
/* Esta funcion establece el servidor en la comunicacion bajo pipes */
/*-------------------------------------------------------------------*/

ULONG fx_MakePipe(VOID)
{
   /*--------------------------------------*/
   /* recordar ques este es el servidor    */
   /*--------------------------------------*/

   server = TRUE;

   /*--------------------------------------*/
   /* ejecuta el reset de NETBIOS          */
   /*--------------------------------------*/

    rc = netbios_reset(REQUEST_RESOURCES);
```

```
  if (rc)
  {
    print_error(rc,"reseting netbios",_FILE_,_LINE_);
    return(FAILED);
  }

  /*---------------------------------------*/
  /* ejecuta add_name de netbios           */
  /*---------------------------------------*/
  rc = netbios_add_name(server_name);
  if(rc) {
    print_error(rc,"adding netbios name",_FILE_,_LINE_);
    return(FAILED);
  }

  return(GOOD);
} // end fx_MakePipe


/*-----------------------------------------------------------------*/
/*  netbios reset helper routine                                   */
/*-----------------------------------------------------------------*/

BYTE  netbios_reset(BYTE request_release)
{
  memset(&ncb.reset_ncb, 0, sizeof(ncb.reset_ncb) );
  ncb.reset_ncb.command  = NB_RESET_WAIT;
  ncb.reset_ncb.lana_num = LANA;
  ncb.reset_ncb.lsn      = request_release;
  ncb.reset_ncb.req_sessions  = NET_LSN;
  ncb.reset_ncb.req_commands  = NET_NCB;
  ncb.reset_ncb.req_names     = NET_NAMES;

  NETBIOS(&ncb);

  return(ncb.reset_ncb.retcode);
}

/*-----------------------------------------------------------------*/
/*  netbios add name helper routine                                */
/*-----------------------------------------------------------------*/
BYTE  netbios_add_name(PCHAR name)
{
  memset(&ncb.cmd_ncb, 0 , sizeof(ncb.cmd_ncb));
  ncb.cmd_ncb.command  = NB_ADD_NAME_WAIT;
  ncb.cmd_ncb.lana_num = LANA;
  strncpy(ncb.cmb.name,name, 16);

  NETBIOS(&ncb);

  return(ncb.cmd_ncb.retcode);

}
```

```
/*-------------------------------------------------------------*/
/*  Funcion para habilitar el servidor de la comunicacion      */
/*-------------------------------------------------------------*/

ULONG   fx_ConnectPipe(VOID)
{
   /*-------------------------------------*/
   /*  perform netbios listen             */
   /*-------------------------------------*/
   rc = netbios_listen(requester_name, server_name, &lsn);

   if (rc)  {
     print_error(rc, "perfoming a netbios listen",_FILE_,_LINE_);
     return(FAILED);
   }

   return(GOOD);
} // end fx_ConnectPipe

/*-------------------------------------------------------------*/
/*  netbios listen name helper routine                         */
/*-------------------------------------------------------------*/

BYTE  netbios_listen(PCHAR caller_name, PCHAR station_name, PBYTE lsn)
{
  memset(&ncb.cmd_ncb, 0, sizeof(ncb.cmd_ncb));
  ncb.cmd_ncb.command  =  NB_LISTEN_WAIT;
  ncb.cmd_ncb.lana_num = LANA;
  ncb.cmd_ncb.rto      = RECV_TIMEOUT<<1;
  ncb.cmd_ncb.sto      = SEND_TIMEOUT<<1;
  strncpy(ncb.cmb.name, station_name, 16);
  strncpy(ncb.cmd.callname, caller_name, 16);

  NETBIOS(&ncb);

  *lsn = ncb.cmd_ncb.lsn;
  return(ncb.cmd_ncb.retcode);
}

/*-------------------------------------------------------------*/
/* funcion para deshabilitar el servidor del pipe de comunicaccion*/
/*-------------------------------------------------------------*/

ULONG fx_DisconnectPipe(VOID)
{
  /*-------------------------------------*/
  /* perform netbios hangup              */
  /*-------------------------------------*/
  rc = netbios_hang_up(lsn);
  if (rc)  {
     print_error(rc,"perfoming a netbios hang up",_FILE_,_LINE_);
     return(FAILED);
   }
   return(GOOD);
```

```
} // end fx_DisconnectPipe


/*-------------------------------------------------------------*/
/* netbios hang up helper routine                              */
/*-------------------------------------------------------------*/
BYTE netbios_hang_up(BYTE lsn)
{
  memset(&ncb.cmd_ncb, 0, sizeof(ncb.cmd_ncb) );
  ncb.cmd_ncb.command  = NB_HANG_UP_WAIT;
  ncb.cmd_ncb.lana_num = LANA;
  ncb.cmd_ncb.lsn      = lsn;

  NETBIOS(&ncb);

  return(ncb.cmd_ncb.retcode);
}

/*-------------------------------------------------------------------------------*/
/* function a requester uses to prepare a pipe for communication  */
/*-------------------------------------------------------------------------------*/

ULONG  fx_Open(VOID)
{
   /*-----------------------------------------*/
   /* remember that this is not a server  */
   /*-----------------------------------------*/
    server  = FALSE;

   /*-----------------------------------------*/
   /* perform  netbios reset            */
   /*-----------------------------------------*/
     rc = netbios_reset(REQUEST_RESOURCES);
     if (rc) {
           print_error(rc, "resetting netbios",_FILE_,_LINE_);
           return(FAILED);
     }

    /*-----------------------------------------*/
    /* perform netbios add name          */
    /*-----------------------------------------*/

     rc = netbios_add_name(requester_name);
     if (rc) {
           print_error(rc, "adding a netbios name",_FILE_,_LINE_);
           return(FAILED);
      }

    /*-------------------------------------*/
    /* perform netbios call              */
    /*-------------------------------------*/
    rc = netbios_call(server_name, requester_name, &lsn);
    if (rc) {
           print_error(rc,"performing a netbios call",_FILE_,_LINE_);
```

```
        return(FAILED);
    }

    return(GOOD);

} // end fx_Open

/*-----------------------------------------------------------------*/
/* netbios  call name helper routine                               */
/*-----------------------------------------------------------------*/
BYTE  netbios_call(PCHAR callname, PCHAR  station_name, PBYTE lsn)
{
  memset(&ncb.cmd_ncb, 0, sizeof(ncb.cmd_ncb));
  ncb.cmd_ncb.command  = NB_CALL_WAIT;
  ncb.cmd_ncb.lana_num = LANA;
  ncb.cmd_ncb.rto      = RECV_TIMEOUT<<1;
  ncb.cmd_ncb.sto      = SEND_TIMEOUT<<1;
  strncpy(ncb.cmd_ncb.name, station_name, 16);
  strncpy(ncb.cmd_ncb.callname, callname, 16);

  NETBIOS(&ncb);

  *lsn = ncb.cmd_ncb.lsn;
  return(ncb.cmd_ncb.retcode);
}

/*-----------------------------------------------------------------*/
/* function both requester and server use to write to a pipe       */
/*-----------------------------------------------------------------*/
ULONG  fx_Write(PVOID BuffArea, USHORT BufferLength, PUSHORT BytesWritten)
{
    /*-------------------------------------*/
    /* perform netbios send                */
    /*-------------------------------------*/

    rc = netbios_send(BufferArea, BufferLength, lsn);

    if (rc) {
        print_error(rc,"performing a netbios send",_FILE_,_LINE_);
        return(FAILED);
    }

    *BytesWritten  = BufferLength;
    return(GOOD);
} // end fx_Write

/*-----------------------------------------------------------------*/
/* netbios send helper routine                                     */
/*-----------------------------------------------------------------*/
BYTE  netbios_recv(PVOID buffer, USHORT  length, BYTE lsn, PUSHORT bytes_received)
{
   memset(&ncb.cmd_ncb, 0, sizeof(ncb.cmd_ncb));
   ncb.cmd_ncb.command  = NB_RECEIVE_WAIT;
   ncb.cmd_ncb.lana_num = LANA;
```

```
  ncb.cmd_ncb.lsn      = lsn;
  ncb.cmd_ncb.buffer_address = buffer;
  ncb.cmd_ncb.length   = length;

  NETBIOS(&ncb);

  *bytes_received = ncb.cmd_ncb.length;
  return(ncb.cmd_ncb.retcode);
}


/*-------------------------------------------------------------------------*/
/* function both requester and server use read from a  pipe     */
/*-------------------------------------------------------------------------*/
ULONG fx_Read(PVOID BufferArea, USHORT BufferLength, PUSHORT BytesRead)
{

   /*-----------------------------------------*/
   /* perform netbios receive          */
   /*-----------------------------------------*/
   rc = netbios_recv(BufferArea, BufferLength, lsn, BytesRead);
   if (rc) {
         print_error(rc,"performing a netbios receive",_FILE_,_LINE_);
         return(FAILED);
   }
   return(GOOD);
} // end fx_Read


/*-----------------------------------------------------------------*/
/* netbios  receive helper routine                    */
/*-----------------------------------------------------------------*/

BYTE  netbios_recv(PVOID buffer, USHORT length, BYTE lsn, PUSHORT bytes_received)
{
  memset(&ncb.cmd_ncb, 0, sizeof(ncb.cmd_ncb));
  ncb.cmd_ncb.command  = NB_RECEIVE_WAIT;
  ncb.cmd_ncb.lana_num = LANA;
  ncb.cmd_ncb.lsn      = lsn;
  ncb.cmd_ncb.buffer_address = buffer;
  ncb.cmd_ncb.length   = length;

  NETBIOS(&ncb);

  *bytes_received = ncb.cmd_ncb.length;
  return(ncb.cmd_ncb.retcode);
}

/*-------------------------------------------------------------------*/
/* function     requester uses to request a service          */
/*-------------------------------------------------------------------*/
ULONG  fx_Transact(PVOID  InBufferArea, USHORT InBufferLength,
                   PVOID  OutBufferArea, USHORT OutBufferLength,
                   PUSHORT BytesRead)
```

```
{
  USHORT  BytesWritten;

  rc = fx_Write(InBufferLength, &BytesWritten);

   if (rc) {
         print_error(rc,"writing during a transaction",_FILE_,_LINE_);
         return(FAILED);
   }

   rc = fx_Read(OutBufferArea, OutBufferLength, BytesRead);
   if (rc) {
         print_error(rc,"reading during a transaction",_FILE_,_LINE_);
         return(FAILED);
   }
   return(GOOD);
} // end fx_Transact


/*-------------------------------------------------------------------------------*/
/* function  requester uses to end communication with a server   */
/*-------------------------------------------------------------------------------*/
ULONG  fx_Close(VOID)
{
  if(!server)  {
     /*-----------------------------------*/
     /* perform netbios reset             */
     /*-----------------------------------*/
     rc  = netbios_reset(RELEASE_RESOURCES);
     if (rc) {
          print_error(rc,"resetting netbios ",_FILE_,_LINE_);
          return(FAILED);
     }
   }
    return(GOOD);
} // end fx_Close
```

PROGRAMA CLIENTE - SERVIDOR UTILIZANDO LA TECNICA DE PROGRAMACION TCP/IP

FXTCPIP.C

```
/*********************************************************************/
/* MODULE NAME: FXTCPIP - TCP/IP Communication                       */
/* DESCRIPCION: Este modulo es una parte del archivo de transfe-     */
/* rencia FX. Usa OS/2 TCP/IP 1.2                                    */
/*********************************************************************/
#include  <stdio.h>
#include  <malloc.h>
#include  <string.h>

#include  <os2.h>
#include  "fxfer.h"

/*********************************************************************/
/* Estructuras TCP/IP                                                */
/*********************************************************************/
typedef struct IN_ADDR
{
  ULONG s_add;
} IN_ADDR;


typedef struct
{
  SHORT   sin_family;
  USHORT  sin_port;
  IN_ADDR sin_addr;
  CHAR    sin_zero[8];
} SOCKADDR_IN;

typedef struct
{
  CHAR *  _Seg16 h_name;
  CHAR ** _Seg16 h_aliases;
  SHORT          h_addrtype;
  SHORT          h_length;
  CHAR ** _Seg16 h_addr_list;
#define    h_addr h_addr_list[0]
} HOSTENT;

/*********************************************************************/
/* Forward Declarations                                              */
/*********************************************************************/

SHORT accept(SHORT socket, SOCKADDR_IN * _Seg16 name, SHORT * _Seg16 namelen);
SHORT bind(SHORT socket, SOCKADDR_IN * _Seg16 name, SHORT * _Seg16 namelen);
SHORT connect(SHORT socket, SOCKADDR_IN * _Seg16 name, SHORT * _Seg16 namelen);
SHORT listen(SHORT socket, SHORT backlog);
```

```
SHORT  recv(SHORT socket, CHAR * _Seg16 buffer, SHORT  length, SHORT flags);
SHORT  sock_init(VOID);
SHORT  socket(SHORT domain, SHORT type, SHORT protocol);
SHORT  soclose(SHORT socket);
USHORT bswap(USHORT addr);
ULONG  inet_addr(CHAR * _Seg16 addr);
SHORT  tcperrno(VOID);

#pragma  linkage(accept, far16)
#pragma  linkage(bind, far16)
#pragma  linkage(connect, far16)
#pragma  linkage(listen, far16)
#pragma  linkage(recv, far16)
#pragma  linkage(send, far16)
#pragma  linkage(sock_init, far16)
#pragma  linkage(socket, far16)
#pragma  linkage(soclose, far16)
#pragma  linkage(gethostbyname, far16)
#pragma  linkage(bswap, far16)
#pragma  linkage(inet_addr, far16)
#pragma  linkage(tcperrno, far16)

#pragma  stack16(8192)

VOID print_error(SHORT, PCHAR, PCHAR, USHORT);

/*********************************************************************/
/*  Variables de Configuracion, inicializadas despues de fx_Initialize */
/*********************************************************************/
CHAR   HostAddressStr[128];   // TCP/IP Host Address string
SHORT  ServerPort;            // Server Port number
USHORT FileBufferSize;        // file and message buffer

/*********************************************************************/
/*  Variables globales para este modulo                              */
/*********************************************************************/

SHORT   server_socket;
SHORT   connection_socket;
SOCKADDR_IN   server_addr;
SOCKADDR_IN   client_addr;

SHORT  namelen;
SHORT  rc;
BOOL   server;

typedef struct  TCPIP_PACKET
{
  ULONG  length;
  PACKET fxpacket;
} TCPIP_PACKET, *PTCPIP_PACKET;

PTCPIP_PACKET  bufferp;
```

```
/*********************************************************************/
/*  Error Display Routine                                            */
/*********************************************************************/

VOID  print_error(SHORT Error, PCHAR msg, PCHAR file, USHORT line)
{
  printf("TCPIP Error %hd (0x%hx) detected while %s at line %lu in file %s.\n",
          Error, Error, msg, line, file);
  rc = tcperrno();
  printf("errno = %hd (0x%hx). \n", rc, rc);
} // end print_error

/*********************************************************************/
/*  Funcion para leer los parametros especificos de la comunicacion  */
/*  y ubicar un buffer para la comunicacion. Se retornara un puntero*/
/*  al buffer y al tama¤o del buffer.                                */
/*********************************************************************/

ULONG  fx_Initialize(PPACKET *buffer_p, PUSHORT size)
{
  FILE   *config_file;
  CHAR    line[80];
  INT     scancount;

  printf("Leyendo los parametros de configuracion.\n");

  /* ------------------------------------- */
  /*  Abriendo el archivo de configuracion  */
  /* ------------------------------------- */
  if (NULL == (config_file = fopen("fxtcpip.cfg","r")))
  {
    print_error(rc,"abriendo el archivo de configuracion",_FILE_,_LINE_);
    return(FAILED);
  }

  else {
    /* ------------------------------------- */
    /* Lee los parametros de configuracion    */
    /* ------------------------------------- */
    fgets(line,sizeof(line), config_file);
    if (sscanf(line,"SERVER_PORT = %d", &ServerPort) != 1)  {
          print_error(rc,"Leyendo ServerPort de la configuracion",_FILE_,_LINE_);
          return(FAILED);
    }

    printf("SERVER_PORT = %d \n", ServerPort);

    fgets(line,sizeof(line),config_file);
    if (sscanf(line,"HOST_ADDRESS" = %s", HostAddressStr) != 1)  {
          print_error(rc,"Leyendo Host Address de la configuracion",_FILE_,_LINE_);
          return(FAILED);
    }

    printf("HOST_ADDRESS = %s \n", HostAddressStr);
```

```
    fgets(line,sizeof(line),config_file);
    if (sscanf(line,"FILE_BUFFER_SIZE" = %u", &FileBufferSize) != 1)  {
          print_error(rc,"Leyendo File Buffer Size de la configuracion",_FILE_,_LINE_);
          return(FAILED);
    }

    printf("FILE_BUFFER_SIZE = %u \n", FileBufferSize);

  } // end if

  fclose (config_file);



  /*---------------------------------------*/
  /* ubica el buffer y retorna un puntero  */
  /* y el tama□o                           */
  /*---------------------------------------*/

  bufferp = malloc(FileBufferSize+4);
  if (bufferp == NULL) {
    print_error(rc,"allocating file buffer",_FILE_,_LINE_);
    return(FAILED);
  }
  *buffer_p = &bufferp->fxpacket;
  *size     = FileBufferSize;

  /*---------------------------------------*/
  /* Inicializa sockets de TCP/IP          */
  /*---------------------------------------*/
  rc = sock_init();
  if (rc != 0) {
    print_error(rc,"Inicializando el socket",_FILE_,_LINE_);
    return(FAILED);
  }
  return(GOOD);

} // end fx_Initialize


/*-------------------------------------------------------------------------------------*/
/*  funcion para establecer al servidor de la comunicacion del pipe    */
/*-------------------------------------------------------------------------------------*/

ULONG  fx_MakePipe(VOID)
{
  #define  SOCK_STREAM   1
  #define  SOCK_DGRAM    2
  #define  AF_INET       2
  #define  IN_ADDR_ANY   0

  /*---------------------------------------*/
```

```
  /*  Recuerde que este es un servidor     */
  /*-------------------------------------*/
  server = TRUE;

  server_socket  = socket(AF_INET, SOCK_STREAM,0);
  if (server_socket <0) {
    print_error(server_socket,"creando el server del socket", _FILE_, _LINE_);
    return(FAILED);
  }

  server_addr.sin_family   = AF_INET;
  server_addr.sin_port     = bswap(ServerPort);
  server_addr.sin_addr.s_addr  = INADDR_ANY;

  rc = bind(server_socket, &server_addr, sizeof(server_addr));
  if (rc < 0) {
    print_error(rc,"binding", _FILE_, _LINE_);
    return(FAILED);
  }

  rc = listen(server_socket, 5);
  if (rc < 0) {
    print_error(rc,"listening", _FILE_, _LINE_);
    return(FAILED);
  }

  return(GOOD);
} // end fx_MakePipe


/*-------------------------------------------------------------------*/
/*  funcion para habilitar  al servidor en la comunicacion del pipe   */
/*-------------------------------------------------------------------*/

ULONG  fx_ConnectPipe(VOID)
{
  namelen  = sizeof(client_addr);
  connection_socket  =  accept(server_socket, &client_addr, &namelen);
  if (connection_socket < 0) {
     print_error(connection_socket, "accepting a connection",_FILE_,_LINE_);
     return(FAILED);
  }

  return(GOOD);
} // end fx_ConnectionPipe

/*-------------------------------------------------------------------*/
/* funcion para deshabilitar al servidor de la comunicacion del pipe  */
/*-------------------------------------------------------------------*/
 ULONG   fx_DisconnectPipe(VOID)
 {
   rc = soclose(connection_socket);
   if (rc < 0) {
     print_error(rc, "closing a socket",_FILE_,_LINE_);
```

```
    return(FAILED);
  }

  return(GOOD);
} // end fx_DisconnectPipe


/*-------------------------------------------------------------------------------------*/
/* funcion que el requester usa para preparar el pipe para la comunic. */
/*-------------------------------------------------------------------------------------*/
 ULONG   fx_Open(VOID)

    /*---------------------------------*/
    /* Recuerde que este es un cliente  */
    /*---------------------------------*/
   server = FALSE;

    /*-----------------------------------------------*/
    /* Establece la comunicacion usando   */
    /* el nombre del servidor y el puerto    */
    /*-----------------------------------------------*/
    connection_socket = socket(AF_INET, SOCK_STREAM, 0);
    if (connection_socket < 0) {
          print_error(connection_socket, "creando el socket",_FILE_,_LINE_);
          return(FAILED);
     }

    /*--------------------------------------*/
    /* Set up the server address structure */
    /*--------------------------------------*/
    server_addr.sin_family      = AF_INET;
    server_addr.sin_port        = bswap(ServerPort);
    server_addr.sin_addr.s_addr  = inet_addr(HostAddressStr);

    rc = connect(connection_socket, &server_addr, sizeof(server_addr));
    if (rc <0)  {
          print_error(rc, "connecting",_FILE_,_LINE_);
          return(FAILED);
    }
    return(GOOD);

} // end fx_Open

/*-------------------------------------------------------------------------------------*/
/* funcion que usan el requester y server para escribir al pipe.      */
/*-------------------------------------------------------------------------------------*/
 ULONG   fx_Write(VOID)
 {
   bufferp->length = BufferLength;
   rc = send(connection_socket, (CHAR * _Seg16)bufferp, BufferLength+4, 0);

   if (rc <0)  {
          print_error(rc, "sending",_FILE_,_LINE_);
```

```
        return(FAILED);
    }
    *BytesWritten = BufferLength;
     return(GOOD);
} // end fx_Write

/*-------------------------------------------------------------------------*/
/* funcion que usan el requester y server para leer del pipe.       */
/*-------------------------------------------------------------------------*/

ULONG fx_Read(PVOID BufferArea, USHORT BufferLength, PUSHORT BytesRead)
{
 ULONG bytes;
 ULONG total_bytes_rcvd;
 CHAR  *Seg16 buffer;

 /*-------------------------------------*/
 /*  obtiene la longitud del mensaje    */
 /*-------------------------------------*/

 total_bytes_rcvd = 0;
 do {
    buffer = (CHAR *)bufferp + total_bytes_rcvd;
    bytes  = recv(connection_socket, buffer,
                  sizeof(bufferp->length) - total_bytes_rcvd, 0);
    if (bytes <= 0) {
        print_error(bytes, "receiving",_FILE_,_LINE_);
        return(FAILED);
    }
    total_bytes_rcvd = total_bytes_rcvd + bytes;
 } while (total_bytes_rcvd < sizeof(bufferp->length));

 /*-------------------------------------*/
 /*  obtiene el mensaje                 */
 /*-------------------------------------*/
 total_bytes_rcvd = 0;
 do {
    buffer = (CHAR *)bufferp + sizeof(bufferp->length) + total_bytes_rcvd;
    bytes  = recv(connection_socket, buffer,
                  bufferp->length - total_bytes_rcvd, 0);
    if (bytes <= 0) {
        print_error(bytes, "receiving",_FILE_,_LINE_);
        return(FAILED);
    }
    total_bytes_rcvd = total_bytes_rcvd +bytes;
 } while (total_bytes_rcvd < bufferp->length);

  *BytesRead = total_bytes_rcvd;
  return(GOOD);
} // end fx_Read

/*-------------------------------------------------------------------------*/
/* funcion que usa el requester para solicitar un servicio.         */
/*-------------------------------------------------------------------------*/
```

```
ULONG  fx_Transact(PVOID InBufferArea, USHORT InBufferLength,
                   PVOID OutBufferArea, USHORT OutBufferLength,
                   PUSHORT  BytesRead)
{
    USHORT  BytesWritten;

    rc = fx_Write(InBufferArea, InBufferLength, &BytesWritten);
    if (rc) {
          print_error(rc,"Escribiendo durante la transaccion",_FILE_,_LINE_);
          return(FAILED);
         }

         rc = fx_Read(OutBufferArea, OutBufferLength, BytesRead);
         if (rc) {
          print_error(rc,"Leyendo durante la transaccion",_FILE_,_LINE_);
          return(FAILED);
         }
         return(GOOD);
} // end fx_Transact



/*-----------------------------------------------------------------*/
/* funcion que usa el requester para terminar la comunicacion con  */
/* el servidor                                                     */
/*-----------------------------------------------------------------*/
ULONG  fx_Close(VOID)
{
  soclose(connection_socket);

  if (server) {
    rc = soclose(server_socket);
    if (rc) {
          print_error(rc,"Cerrando el socket del servidor",_FILE_,_LINE_);
          return(FAILED);
    }
  } // end if

  return(GOOD);
} // end fx_Close
```

MANUAL DEL USUARIO

1.- INSTALACION Y EJECUCION:

El programa tiene que ser instalado tanto en el lado del servidor como en el lado del cliente. Primero se debe crear un directorio en el disco donde van a residir los programas ejecutables y los archivos de datos utilizados por ellos, el cual puede ser:

C:\MD CONSULTA ↵ (ENTER)

Luego de haber creado el directorio seguimos los siguientes pasos:

1.1. Instalar en el Servidor los siguientes programas:

SERVIDOR.EXE

DATOS.TXT

1.2. Instalar en el Cliente los siguientes programas:

CLIENTE.EXE

DATOS.TXT

1.3. Ejecución de los programas:

Ejecutar el Programa Servidor:

C:\CONSULTA\SERVIDOR.EXE ↵

Ejecutar el Programa Cliente:

C:\CONSULTA\CLIENTE.EXE ↵

2.- MODO DE EMPLEO:

Al ejecutar el programa Cliente aparecerá el Meñú Principal con las siguientes opciones:

==========

MENU PRINCIPAL

1: CONSULTA DE SALDOS.

2: PAGO DE CHEQUES.

3: DEPOSITOS.

4: SALIR.

==========

Seleccione una opción y presione enter para ejecutarla: [ ]

Al escoger la **opción 1**: Consulta de Saldos aparecerá una pantalla que pedirá los siguientes datos, los cuales deben ser digitados:

CONSULTA DE SALDOS:

Ingrese su número de cuenta: [---------]

El Servidor leerá el número de la cuenta y una vez que haya comprobado que ésta existe retornará al cliente la respectiva consulta del saldo de su cuenta con los siguientes datos:

==========

DATOS PERSONALES:

NOMBRE: ----------

DIRECCION: ----------

TELEFONO: ------

CEDULA: ----------

FECHA DEL ULTIMO MOVIMIENTO: MM/DD/AA

SALDO ACTUAL: ---------- **SALDO ANTERIOR:** ----------

ESTADO DE CUENTA:

FECHA	**TIPO**	**VALOR**	**NUM_DOC**
mm/dd/aa	-	----------	--------

==========

Al escoger la **opción 2**: Pago de Cheques, aparecerá una pantalla que pedirá los siguientes datos:

Ingrese el número de su cuenta: [--------]

Digite el número del cheque: [--------]

Valor en sucres S/.: [----------]

Fecha del Movimiento: [mm/dd/aa]

Una vez ingresados todos los datos el servidor enviará un código (TRAN_FINAL) que indicará que la transacción ha sido efectuada exitosamente:

CONSULTA: Transacción OK. Fin de Datos.

Al escoger la **opción 3**: Depósitos aparecerá una pantalla que pedirá los siguientes datos:

Ingrese el número de su cuenta: [--------]

Digite el número del depósito: [--------]

Valor en sucres S/.: [----------]

Fecha del Movimiento: [mm/dd/aa]

Una vez ingresados todos los datos el servidor enviará un código (TRAN_FINAL) que indicará que la transacción ha sido efectuada exitosamente:

CONSULTA: Transacción OK. Fin de Datos.

Al escoger la **opción 4**: Salir, retornará al prompt del Sistema Operativo.

// PROGRAMA CLIENTE- SERVIDOR BASADO EN LA TECNICA DE
// PROGRAMACION NAMED - PIPES

```
/*-----------------------------------------------------------------*
PROGRAMA : SERVI000.C
OBJETIVOS: Realizar las transacciones necesarias para la consulta y la
           actualizacion de datos desde las bases de consulta y captura.
           Es un programa servidor.

AUTOR   :
FECHA   :

NOTA    :** Este codigo debe ser compilado y ejecutado bajo OS/2. **
*-----------------------------------------------------------------*/


#define CONFIG
#include "servi000.h"


char *SEMPHR =  "\\SEM\\";
char *PIPE   =  "\\PIPE\\";

unsigned long RSem    = 0L;
int count = 0;

HSEM Procces=0L;
SEL selshr;

struct_var_sys var_sys;
extern struct_estacion var_estacion[];
extern FILE *fp_datos,*fp_saldo,*fp_movim;

/*-----------------------------------------------------------------*
          ** PROGRAMA PRINCIPAL **
*-----------------------------------------------------------------*/

int main (int argc,char *argv[])
{
   USHORT rc;
   char pnam[40];
   void station(void *);

   /* Var Apl */
   int i;
   USHORT   outsz = MAXBUFFER,   // outgoing buffer size
            insz  = MAXBUFFER;  // incoming buffer size

// ---- Lee configuracion ----
   Init_Aplic ("SERVIDOR.CFG");

// ---- Total de Hilos para Atender Requerimientos: PERMANENTES + TEMPORALES
   var_sys.max_hilos_total = var_sys.max_hilos_perm + var_sys.max_hilos_temp;
```

```
   DosSemSet (&RSem);

// ---- Inicializamos la bases de Consulta de Saldos en SQL Server (consulta y captura)
   rc = AbrirArchivos();

// ---- Forma nombre del Pipe:
//      Input : se toma este nombre tal como es
//      Output: se le concatena 01, 02, .... nn, hasta el max. de Out Pipes

   printf("va a concatenar el PIPE");
   strcpy (pnam, PIPE);
   printf("va a concatenar a pnam el name_pipe");
   strcat (pnam, var_sys.name_pipe);

   printf("nombre del pipe que se ha formado: %.40s",pnam);

// ---- Creacion de Input Pipes: Un solo Pipe pero con N INSTANCIAS
   Crea_In_Pipes(pnam, var_sys.max_in_pipe);

// ---- Creacion de Output Pipes: N PIPEs, con 1 INSTANCIA cada uno
   Crea_Out_Pipes(pnam, var_sys.max_hilos_total, var_sys.server);

// ---- Semaforos para Control de Ejecucion de HILOS PERMANENTES
   Set_Sem_Hilos_Perm();

// ---- Arranca Hilos de Proceso de Clientes: Uno para cada Hilo PERMANENTE
   for(i = 1; i <= var_sys.max_hilos_perm; i++)
     _beginthread (station, NULL, STACK_SIZE, (void *)i);

// ---- Arranca los Hilos para los Pipes de Entrada
   for (i = 1; i <= (var_sys.max_in_pipe); i++)

    _beginthread ((void *) Atiende_In_Pipe, NULL, STACK_SIZE, (void *) i);

   // Bloqueo la ejecuci¢n del proceso.
   rc = DosSemWait(&RSem, SEM_INDEFINITE_WAIT);

   fprintf (stderr,"End\n");

   // Desactiva los pipes
   for (i = 1; i <= var_sys.max_hilos_total; i++)
            if (rc = DosClose(var_estacion[i].pipe))
                     printf("\n SERVI000: Error DosClose Pipe[%i]", i);

   // Se desconecta la aplicacion con la Base de Datos
   CerrarArchivos();

   return 0;
}   /* fin de main */

//*************************************************************************
void Set_Sem_Hilos_Perm(void)
{
   int i, rc;
```

```
    for (i = 1; i <= var_sys.max_hilos_perm; i++)  {

          if ( (rc = DosSemSet(&SemHiloPerm[i])) )  {
            fprintf(stderr,"\n SERVI000: Error SemSet SemHiloPerm[%d] (rc = %d)", i, rc);
            exit (1);
          }
    }
} // End Set_Sem_Hilos_Perm()

// ***************************** ATIENDE_IN_PIPE ****************************
void Atiende_In_Pipe(int i)
{
 char       buff[BUFFERSZ], out_pipe[50];
 int        rc,  num_pipe, flag_P_T;
 char    msg[] = "\n SERVI000: Input Thread: Espera Req. de Estacion: Hilo [%d]\n";
 HPIPE   hp;

 hp = In_Pipe_Hand[i];
 printf(msg, i);

 // Corre Indefinidamente Esperando un Requerimiento de Alguna Estacion
 while ( 1 )  {

   // --- Espera por DosOpen de alguna estacion (Pipe General)
   if ((rc = DosConnectNmPipe(hp)) != 0)  {
             printf("\n SERVI000: Error en DosConnectNmPipe.  rc = %.2d", rc);
             _endthread();
             //continue;
   }
   // --- Lee Requerimiento Inicial de In_Pipe ----------------------
   memset(buff, 0, sizeof(buff));

   printf("Mensaje: Lee Requerimiento de In pipe");

   rc = Lee_Req_In_Pipe(buff, hp, i);

   if (rc == ERR_REQ_INICIAL) {
          // --- Retorna a la Estacion el respectivo Codigo de Error
             memcpy(buff, ERROR_REQ_INICIAL, 5);
             rc = Envia_Out_Pipe(hp, buff);
             if (rc != 0)   {
               printf("\n SERVIDOR: Error en Envio de OutPipe. rc = %.2i", rc);
               _endthread();
             }

             printf(msg, i);        // Print Mensaje de Nuevo Requerimiento
             continue;              // Regresa a Esperar por Nuevo Requerimiento
    }
    else if (rc != 0)          {
          printf("\n SERVIDOR: Error en Lectura de In Pipe. rc = %.2i", rc);
            _endthread();
}
```

```
printf("\n SERVIDOR:  RECIBI DE LA ESTACION:  -%s-", buff);

// --- Busca un Pipe Disponible (y su respectivo Hilo: TEMP o PERM)
memset(buff, 0, sizeof(buff));
if ( (rc = Busca_Pipe_Disponible(out_pipe, &num_pipe, &flag_P_T)) )  {

   printf("Nombre del out_pipe -%s-\n",out_pipe);

   strcpy(buff, PIPE_NO_DISPONIBLE);
}
else  {
            strcpy(buff, PIPE_DISPONIBLE);
            strcat(buff, out_pipe);
}

printf("\n SERVIDOR: Out_Pipe: -%s-", out_pipe);

if (flag_P_T == PERMANENTE)  {  // CLEAR de Sem que detiene a Hilo Perm.
            printf("\n SERVIDOR: Reinicia Hilo PERMANENTE.  Hilo (%d)", num_pipe);
            if ( (rc = DosSemClear(&SemHiloPerm[num_pipe])) )  {
               fprintf(stderr,"\n SERVIDOR: Error SemSet SemHiloPerm[%d] (rc = %d)", i, rc);
               _endthread();
            }
}
if (flag_P_T == TEMPORAL)  {   // Debe arrancar un NUEVO THREAD
            printf("\n SERVIDOR: Arranca Hilo TEMPORAL.  Hilo (%d)", num_pipe);
            _beginthread (station, NULL, STACK_SIZE, (void *) num_pipe);
}

// --- Envia a la Estacion el Nombre del Pipe de Proceso de Transaccion
// --- Luego se Desconecta y Regresa a Esperar por Nuevo Requerimiento
rc = Envia_Out_Pipe(hp, buff);
if (rc != 0)     {
            printf("\n SERVIDOR: Error en Envio de OutPipe. rc = %.2i", rc);
            _endthread();
}

 printf(msg, i);
} // Fin de while

} // End Atiende_In_Pipe()
```

```
// ************************* BUSCA_PIPE_DISPONIBLE **********************
// SERVIDOR  mantiene en memoria una lista de nombres de PIPE que pueden ser
//       usados para Envio de Resultados
// Esta funcion busca un PIPE que NO ESTE siendo usado
// Se busca hasta el max de HILOS PERMANENTES
// Si todos los PIPES de HILOS PERMANENTES estan ACTIVOS,
//    se busca un PIPE para un HILO TEMPORAL.
// En ambos casos se MARCA el PIPE como ACTIVO
// Retorna los siguientes valores:
//  - out_pipe :  Nombre del Pipe Obtenido
//  - num_pipe :  Indice en  TablaPipes del Pipe Obtenido
//  - flag_P_T :  Indica si Pipe corresponde a Hilo PERMANENTE o TEMPORAL
// ***********************************************************************

int Busca_Pipe_Disponible(char *out_pipe, int *num_pipe, int *flag_P_T)
{
  int  i;
  char estado;

  printf("\n SERVI000: Entra a Busca_Pipe_Disponible");
  DosSemRequest(&SemPipeDis, SEM_INDEFINITE_WAIT);

  printf("\n Busca_Pipe_Disponible: antes del lazo");
// BUSCA UN PIPE DE UN HILO PERMANENTE o TEMPORAL
  for(i = 1; i <= var_sys.max_hilos_total ; i++)        {
    estado = (TablaPipes[i])->status;

    if (estado == NO_ACTIVO)  {  // Encontro Pipe Disponible (TEMPORAL)
              (TablaPipes[i])->status         = ACTIVO;

              strcpy(out_pipe, (TablaPipes[i])->pipe_name);
       printf(" Busca_pipe_disponible: -%s-\n",out_pipe);
              *num_pipe = i;
              if (i > var_sys.max_hilos_perm)
                *flag_P_T = TEMPORAL;
              else
                *flag_P_T = PERMANENTE;

              break;
     } // Fin if
  }//fin for

  DosSemClear(&SemPipeDis);

  if (i > var_sys.max_hilos_total)  {
    printf("\n SERVIDOR: No encontro Pipe Disponible.");
    printf("\n       Estacion debe Reintentar.");
    return (1);       // Error: No encontro ningun Pipe Disponible
  }
  else
    return (0);

}//Fin de funci¢n Busca_Pipe_Disp
```

```
// ****************** LEE_REQ_IN_PIPE ***************************

int Lee_Req_In_Pipe(char *buff, HPIPE hp, int i)
{
 int rc, b_read;

 memset(buff, '\0', sizeof(buff));
 if ((rc = DosRead(hp, buff, 10, &b_read)) != 0)      {
   printf("\n ESTACION: Error en Lectura de In Pipe. Hilo (%.2i)", i);
   printf("\n Long: %d  Hilo (%.2i):", b_read, i);
   printf("\n ESTACION: Lectura Inicial: -");
   putbuffer(buff,b_read);
   printf("-");
   return (rc);
 }
 buff[b_read] = '\0';
//  printf("\n ESTACION: Lectura Inicial: -%s-  Long: %d  Hilo (%.2i):", buff, b_read, i);
 printf("\n Long: %d  Hilo (%.2i):", b_read, i);
 printf("\n ESTACION: Lectura Inicial: -");
 putbuffer(buff,b_read);
 printf("-");
 if (strncmp(buff, REQ_INICIAL, 5))  {
   printf("\n ESTACION: Error en Lectura de Req. Inicial.  Leido: -%s-", buff);
   return (ERR_REQ_INICIAL);
 }

 return (rc);
}

void putbuffer(char * buffer, int longitud)
{
  int i = 0;
  for ( i = 0 ; i < longitud; i++)
    putchar(buffer[i]);
}

// ****************** ENVIA_OUT_PIPE ***************************
int Envia_Out_Pipe(HPIPE hp, char *buff)
{
 int rc, b_written;

 printf("\nbuffer a retornar: (%s)", buff);
 // --- Envia a la Estacion el Nombre del Pipe de Proceso de Transaccion
 if ((rc = DosWrite(hp, buff, strlen(buff), &b_written))        != 0)  {
            printf("\n SERVIDOR: Error en DosWrite.  rc = %.2d", rc);
   return (rc);
 }
 printf("\nDoswrite OK rc: %d",rc);

 // --- Se Desconecta y Regresa a Esperar por Nuevo Requerimiento
 if ((rc = DosDisConnectNmPipe(hp)) != 0)  {
            printf("\n SERVIDOR: Error en DosDisConnectNmPipe.  rc = %.2d", rc);
   return (rc);
 }
```

```
 printf("\nDosDisConnect OK rc: %d",rc);

 return (rc);
} // End Envia_Out_Pipe()

/*-------------------------------------------------------------------------*/

int search_label (char *token)
{
int i;

for (i = 0; labels_id[i][0]; i++)
   if (strcmp(token,labels_id[i])==0)
      return i;
return -1;
}  // fin de search_label()


/*-------------------------------------------------------------------------*/

int Init_Aplic(char *fnam)
{
  FILE *fp;
  char line[MAXLINE],label[20],value[MAXLINE],*ptrline;
  int id;

  if ((fp = fopen(fnam, "r")) == NULL) {
    perror("fopen");
    printf ("Error Archivo de Configuraci¢n no Existe [%s]\n",fnam);
    exit(1);
  }  // endif

  while (fgets(line, MAXLINE, fp)) {
   ptrline=strchr(line,' ');

    if (ptrline) {
     line[strlen(line)-1]='\0';
     strnzcpy(label,line,ptrline-line);

     if ((id=search_label(label))!=-1) {
                strncpy (value,ptrline+1,line+strlen(line) - ptrline);
                ltrim (value);
                switch (id) {
                case MAX_IN_PIPE:
                    var_sys.max_in_pipe = atoi(value);
                    if (var_sys.max_in_pipe > MAXINPIPES) {
                              printf ("\nError en Num. de Hilos de Entrada (param. MAX_IN_PIPE)");
                              printf ("\nRevise Archivo de Configuracion (%.12s)", fnam);
                              exit (1);
                    } // endif
                    break;
                case MAX_HILOS_PERM:
                    var_sys.max_hilos_perm = atoi(value);
                    if (var_sys.max_hilos_perm > MAXSTATIONS) {
```

```
                        printf ("\nError en Num. de Hilos de Entrada (param.
MAX_HILOS_PERM)");
                        printf ("\nRevise Archivo de Configuracion (%.12s)", fnam);
                        exit (1);
                    } // endif
                    break;
                case MAX_HILOS_TEMP:
                    var_sys.max_hilos_temp = atoi(value);
                    if (var_sys.max_hilos_temp > MAXSTATIONS) {
                        printf ("\nError en Num. de Hilos de Entrada (param.
MAX_HILOS_TEMP)");
                        printf ("\nRevise Archivo de Configuracion (%.12s)", fnam);
                        exit (1);
                    } // endif
                    break;
                case TIME_OUT_PIPE:
                    var_sys.time_out_pipe = atoi(value);
                    if (var_sys.time_out_pipe > MAXTIMEOUT) {
                        printf ("\nError en Valor del Time Out (param. TIME_OUT_PIPE)");
                        printf ("\nRevise Archivo de Configuracion (%.12s)", fnam);
                        exit (1);
                    } // endif
                    break;
                case SERVER :
                    strcpy (var_sys.server, value);
                    break;
                case PIPENAME :
                    strcpy (var_sys.name_pipe, value);
                    break;
                case NAMAPLCC:
                    strcpy(var_sys.name_aplicc, value);
                    break;


            } // endswitch
        } else
            printf ("Identificador en configuracion Desconocido,%s\n",label);
        } else
            printf ("Identificador no tiene valor, %s\n",line);
    } // endwhile

    fclose (fp);
    return 0;
} // fin de Init_Aplic()
```

```
//***************************** CREA_IN_PIPES ***********************
// Crea los PIPES para Recepcion de Requerimientos.  Caracteristicas:
// -  mode : NP_WAIT --> BLOQUEADO: Espera hasta que haya Conexion.
// -  instances :  Varias Instancias.
// -  buff_size :  BUFFERSZ  --> Tanto para Input como para Output
// Los HANDLES de cada PIPE creado, quedan almacenados en:
//     In_Pipe_Hand[i]
// Tambien Inicializa la Memoria de Control con:
// -  Status PIPE: ACTIVO / NO_ACTIVO
// -  Nombre PIPE: Secuencial desde 1 hasta var_sys.max_hilos_total
// -  Handle PIPE: Retornado al Crear el PIPE
//*********************************************************************

void Crea_In_Pipes(char *nombre_pipe, int instances)
{
 int  i;

 printf("\n\n SERVIDOR: Creacion de Pipes de Entrada de Requerimientos");
 // --- Creamos la primera Instancia del Pipe, indicando NUMERO de instancias
 In_Pipe_Hand[1] = Crea_Pipe(nombre_pipe, NP_WAIT, instances, BUFFERSZ);

 if (In_Pipe_Hand[1] == (HPIPE) NULL_PIPE)  {
             exit(1);
 }
 // --- Creamos las (N - 1) Instancias restantes.
 for (i = 2; i <= instances; i++) {
    In_Pipe_Hand[i] = Crea_Pipe(nombre_pipe, NP_WAIT, 1, BUFFERSZ);

    if (In_Pipe_Hand[i] == (HPIPE) NULL_PIPE)  {
                  exit(1);
    }
 }  // End for()

} // End Crea_In_Pipes()


//***************************** CREA_OUT_PIPES ***************************
// Crea los PIPES para Envio de Resultados, con las sgtes caracteristicas:
// -  mode : NP_NOWAIT --> NO BLOQUEADO: no espera por Conexion.
// -  instances : UNA SOLA Instancia.
// -  buff_size : MAXBUFFER  --> Tanto para Input como para Output
// Los HANDLES de cada PIPE creado, quedan almacenados en:
//     (TablaPipes[i])->pipe_hand
// Tambien Inicializa la Memoria de Control con:
// -  Status PIPE: ACTIVO / NO_ACTIVO
// -  Nombre PIPE: Secuencial desde 1 hasta max_hilos_total
// -  Handle PIPE: Retornado al Crear el PIPE
//*********************************************************************

void Crea_Out_Pipes(char *nombre_pipe, int max_hilos_total, char *server)
{
 int  i;
 char out_pipe[25];
```

```
printf("\n\n --> Creacion de Pipes de Envio de Resultados");
// Asignamos memoria a TablaPipes, la Inicializamos, y creamos el Pipe
for (i = 1; i <= max_hilos_total; i++)  {
   TablaPipes[i] = (struct_pipes_disponibles *)
            malloc(sizeof(struct_pipes_disponibles));

   // Asigna Estado inicial del Pipe, su Nombre, y su Handle
   // El Nombre INCLUYE el Codigo de SERVER en el que se esta ejecutando
   (TablaPipes[i])->status = NO_ACTIVO;
   sprintf((TablaPipes[i])->pipe_name, "%s%s%02d", server, nombre_pipe, i);

   sprintf(out_pipe, "%s%02d", nombre_pipe, i);

   // Creamos los n Pipes, todo con UNA SOLA INSTANCIA
   (TablaPipes[i])->pipe_hand = Crea_Pipe(out_pipe, NP_NOWAIT, 1, MAXBUFFER);

   if ((TablaPipes[i])->pipe_hand == (HPIPE) NULL_PIPE)  {
        exit(1);
   }
 } //end for

} // End funcion Crea_Out_Pipes()


//******************************** CREA_PIPE ********************************
// Crea un PIPE.  Toma 3 argumentos como Input
//  - name_pipe:  Nombre del PIPE
//  - mode    :  Bloqueado    (para PIPEs de entrada)
//               No Bloqueado (para PIPEs de salida)
//  - instances:  Indica cuantas Instancias del PIPE seran Creadas
//               Input PIPEs --> n.    Out PIPEs --> siempre 1.
// Retorna  0  si no pudo Crear el Pipe
//***************************************************************************

HPIPE Crea_Pipe(char *name_pipe, USHORT mode, int instances, int buff_size)
{
 HPIPE   hp;
 USHORT  outsz,                    // outgoing buffer size
                 insz;             // incoming buffer size
 int  rc;

 insz = outsz = buff_size;

 printf("\n SERVIDOR: Nombre de Pipe a Crear: -%s-", name_pipe);
 if ((rc = DosMakeNmPipe(name_pipe, &hp,
                                 NP_ACCESS_DUPLEX | NP_NO_INHERIT | NP_WRITEBEHIND,
                                 mode | NP_READMODE_MESSAGE | NP_TYPE_MESSAGE |
instances,
                                 outsz, insz, (long) TIMEOUT)) != 0)  {
   printf("\n SERVIDOR: Error en Creacion de PIPE: -%s-", name_pipe);
   return ((HPIPE) NULL_PIPE);
 }

 return (hp);
```

```
}


/*-------------------------------------------------------------------------*/

void rdkbd(void *dummy)
{
   char buf[BUFSIZ];

   for(;;) {
     putchar('#');
     putchar(' ');
              gets(buf);
     if(!strcmp(buf,"quit"))
            break;
   } // endfor

   DosSemClear(&RSem);

  _endthread();
}  // fin de rdkbd


int AbrirArchivos()
{

  printf("\n Abro el archivo de datos requerido");

  fp_datos = fopen("datos.txt","r");

  fp_saldo = fopen("saldo.txt","r+");

  fp_movim = fopen("movims.txt","a+");


   if ( (fp_datos == NULL) )  {
     fprintf(stderr, "No se puede abrir archivo %s\n", "datos.txt");
     return(1);
   }

   if ( (fp_saldo == NULL) )  {
     fprintf(stderr, "No se puede abrir archivo %s\n", "saldo.txt");
     return(1);
   }


   if ( (fp_movim == NULL) )  {
     fprintf(stderr,"No se puede abrir archivo %s\n", "movim.txt");
     return(1);
   }


 }// End Abrir_archivos()
```

```
int  CerrarArchivos( )
{
//   FILE  *fp_datos, *fp_saldo, *fp_movim;


   fclose (fp_datos);

   fclose (fp_saldo);

   fclose (fp_movim);

   return (0);

} // End Cerrar_Archivos


/*fin de SERVIDOR.C */
```

MODULO INCLUIDO EN EL PROGRAMA SERVI000.C

```
/* inicio : servidor.h */

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//              DEFINICIONES PARA EL COMPILADOR
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
#define INCL_DOS
#define INCL_SUB
#define INCL_DOSSEMAPHORES
#define INCL_DOSNMPIPES


//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//              ARCHIVOS DE CABECERA
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
#include <os2.h>
#include <dos.h>
#include <time.h>
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stddef.h>
#include <memory.h>
#include <process.h>
#include <errno.h>
#include <malloc.h>
#include <direct.h>



//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//              DEFINICION DE CONSTANTES
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
#define REQ_INICIAL         "99999"
#define PIPE_DISPONIBLE     "99991"
#define PIPE_NO_DISPONIBLE  "99990"
#define ERROR_REQ_INICIAL   "99992"
#define ERR_REQ_INICIAL     9992
#define PIPE_NM_SZ          50
#define NP_NO_INHERIT       0x0080
#define RPLSZ               80
#define OFFSET              5
#define TRUE                1
#define FALSE               0
#define BUFFERSZ            256
#define SESLIMIT            5
#define NUMSESIONES         5
#define STACK_SIZE          20480
#define JNRSGSZ             30000
#define MAXTRANS            20
```

```
#define MAXLINE          80
#define TIMEOUT        30000     // 30 Segs.
#define MAXTIMEOUT       120      // 120 Segs.  Usado en Output Pipes
#define MAXINPIPES       10
#define MAXSTATIONS       32      // maximo #. de estaciones
#define MAXSQLHILOS       16      // numero de conexiones con SQL Server
#define MAXBUFFER       4096

#define ACTIVO          '1'
#define NO_ACTIVO        '0'
#define EXITO        0
#define PERMANENTE        1
#define TEMPORAL        2
#define NULL_PIPE       0
#define S_TRANOEXIST     -10  // transaccion no existe

// codigos de retorno de  transacciones (funcion RunTransaction)
#define TRAN_NO_EXISTE    0
#define TRAN_OK_NO_FIN    1
#define TRAN_OK_FIN      22
#define TRAN_ERROR       3



//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//                DEFINICION DE ESTRUCTURAS
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
#ifdef CONFIG
char *labels_id[]={
          "MAX_IN_PIPE",
          "MAX_HILOS_PERM",
          "MAX_HILOS_TEMP",
          "TIME_OUT_PIPE",
  "SERVER",
          "PIPENAME",
          "NAMAPLCC"
    };

enum labels{
          MAX_IN_PIPE,
          MAX_HILOS_PERM,
          MAX_HILOS_TEMP,
          TIME_OUT_PIPE,
  SERVER,
  PIPENAME,
          NAMAPLCC
    };
#endif

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//                DEFINICION DE ESTRUCTURAS
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
typedef struct {
        char trans_id[10];
```

```
        char turno;
} lib_trans;

typedef struct {
        int  max_in_pipe;
        int  max_hilos_perm;
        int  max_hilos_temp;
        int  max_hilos_total;
        int  time_out_pipe;      // Time Out expresado en segundos
 char server[25];
        char name_pipe[25];
        char name_aplicc[25];

} struct_var_sys;

typedef struct {
        char     transac[6];
        char     pipe_name[PIPE_NM_SZ];
        int      aplicc_ini;
        int      num_trans;
        lib_trans user_lib[MAXTRANS];
        HPIPE    pipe;
} struct_estacion;

// campos staticos a almacenar por hilo
typedef struct {
                        char tip_ima;
                        int  num_sel;
            } struct_campos_estaticos;


// Arreglo de Handlers para Output PIPEs
struct_estacion var_estacion[(MAXSTATIONS*2) + 1];



typedef struct {
        char  status;
        char  pipe_name[40];
 HPIPE  pipe_hand;
}struct_pipes_disponibles;

// Para Tabla de Output Pipes disponibles para Estaciones
struct_pipes_disponibles *TablaPipes[(MAXSTATIONS*2) + 1]; // Arreglo de ptrs

// Arreglo de Handlers para Input PIPEs
HPIPE  In_Pipe_Hand[MAXINPIPES + 1];

// Semaforo para Control de Acceso a Tabla de Output Pipes
HSEM SemPipeDis;

// Semaforos para Control de Ejecucion de Hilos Permanentes
HSEM SemHiloPerm[MAXSTATIONS + 1];
```

```
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//                    PROTOTIPOS DE FUNCIONES
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

// ------------ prototipos de SCAPIMA1.C ----------------------------------
void load_trans ( char *, int, int );
void SumaDias(char *,int, char * );

// ------------ prototipos de SERVIDOR.C ----------------------------------
void  Crea_In_Pipes(char *, int );
void  Crea_Out_Pipes(char *, int , char *);
HPIPE Crea_Pipe(char *, USHORT , int , int );
void  Atiende_In_Pipe(int );
int   Lee_Req_In_Pipe(char *, HPIPE , int );
int   Envia_Out_Pipe(HPIPE, char *);
int   Busca_Pipe_Disponible(char *, int *, int *);
int   Init_Aplic  (char *);
void  Set_Sem_Hilos_Perm(void);
void  putbuffer(char * buffer, int longitud);
char *ltrim(char *string);


// ------------ prototipos de SERVIOO2.C ----------------------------------
int GET_SALDO_CLIE (char *, int, HPIPE *, int );
int INSERTA_DATOS_MOVIMS (char *, int, HPIPE *, int );

// ------------ prototipos de SERVI001.C ---------------------------------
int Run_Transaction(char *,int, int);
char *strnzcpy (char *, char *, unsigned int);

int AbrirArchivos(void);
int CerrarArchivos(void);

void Liberar_Out_Pipe(int);
int Establece_Conexion(HPIPE);
int Lee_Cod_Transaccion(HPIPE, char *, int *);
int WaitRequerimiento(int);
void ShutDown(char *);

// Prototipos de SERVIDOR.C




 /* fin: SERVIDOR.H */
```

/* INICIO DE SERV001.C */

```
#include "servi000.h"

#include "funlib.h"

long num_secuencia = 0L;
unsigned long SecSem  = 0L;

extern trans_func funlib[];
extern struct_var_sys var_sys;
extern struct HANDLER handler[];

#define ES_HILO_TEMPORAL  (n > var_sys.max_hilos_perm)
#define ERROR_NO_DATA     232

/*-------------------------------------------------------------------------*/

void station(int n)
{
   USHORT rc, b_read, b_written;
   char buf[MAXBUFFER], reply[100];
   int  len = 21;       // Por favor comenta que es este 21

   printf("\n SERVI001: Arrancando Hilo de Proceso %d\n", n);


//  ---- Lazo Principal: Necesario para Hilos Permanentes
//                  En Hilos Temporales se rompe el Lazo y _endthread()
   while (1)  {

//    ---- Si es un Hilo PERMANENTE, ESPERA hasta que se Libere Semaforo
      if (n <= var_sys.max_hilos_perm)
                  if (rc = WaitRequerimiento(n) )
                    _endthread();

//    ---- Establece Conexion con Estacion
      if ((rc = Establece_Conexion((TablaPipes[n])->pipe_hand)) != EXITO)     {
            printf("\n SERVI001: No se pudo Establecer Conexion con Estacion");
                  printf("\n        Se procede a Liberar Output Pipe (%.2d)", n);

                  Liberar_Out_Pipe(n);
                  if (ES_HILO_TEMPORAL)
                     _endthread();    // Termina Ejecucion del Hilo
                  else
                     continue;       // Regresa a Esperar por Nuevo Requerimiento
      } // End Establece_Conexion


//    ----  Segundo Lazo: Para los sucesivos Reads de una misma Transaccion
      while (2) {
                  memset (buf, ' ', sizeof(buf));
        printf("\nAntes de Lee_Cod_T");
                  if((rc = Lee_Cod_Transaccion((TablaPipes[n])->pipe_hand, buf, &b_read)) != 0)  {
                     printf("\n SERVI001: Error DosRead.  Hilo (%d)  rc(%d)", n, rc);
                     break;
                   }
         buf[b_read] = '\0';
```

```
                printf("\nRecibo Hilo [%d]  bytes [%d]\n [%.78s]", n, b_read, buf);

                strnzcpy (reply, buf, OFFSET);

                if ((rc = Run_Transaction (buf, b_read, n)) == TRAN_NO_EXISTE) {
                   rc = S_TRANOEXIST;
                   sprintf(reply,"%.4d",rc);
                   printf ("reply -%s-\n",reply);
                   if((rc = DosWrite((TablaPipes[n])->pipe_hand, reply, strlen(reply), &b_written)) != 0)
{
                            fprintf (stderr,"Error Write %d\n",rc);
                            break;
                   }
                } // endif Run_Transaction()

                // Termina, si se completo la transaccion de varios envios
                if ( (rc == TRAN_OK_FIN) || (rc == TRAN_ERROR) )
                  break;
                // Caso Contrario, continua
    } // end while (2)

    // poner condicion para realizar el disconnect
    if ((rc = DosDisConnectNmPipe((TablaPipes[n])->pipe_hand))!=0)
              printf("\n SERVI001: Error DosDisconnect.  Hilo (%d)  rc(%d)", n, rc);

    Liberar_Out_Pipe(n);

//    ---- Termina Ejecucion, si es un Hilo TEMPORAL
    if (ES_HILO_TEMPORAL)   break;

  } // end while (1)

} // fin de station()


//*************************** LIBERAR_OUT_PIPE *****************************
// OBJETIVO: Asigna a un OUTPUT PIPE, el status de DISPONIBLE
//**************************************************************************

void Liberar_Out_Pipe(int num_pipe)
{

   DosSemRequest(&SemPipeDis, SEM_INDEFINITE_WAIT);
   (TablaPipes[num_pipe])->status = NO_ACTIVO;
   DosSemClear(&SemPipeDis);
}

//******** Lee_Cod_Transaccion ***********************************
int Lee_Cod_Transaccion(HPIPE hpipe, char *buf, int *b_read)
{
  int rc;
  time_t time_ini, time_fin;

  printf("\n entra a Lee_Cod_Transaccion");
  time(&time_ini);

  do {
    if ((rc = DosRead(hpipe, buf, MAXBUFFER, b_read)) == 0)
```

```
                return(rc);

    time(&time_fin);     // Si no, determina si TIMEOUT ha expirado

  } while(  (rc == ERROR_NO_DATA)   &&
            ( (time_fin - time_ini) < var_sys.time_out_pipe) );

  if (rc == ERROR_NO_DATA)
    printf("\n SERVI001: Expiro TIMEOUT de (%.3d) segs", var_sys.time_out_pipe);

  printf("\n SERVI001: Error DosRead.  rc(%d)", rc);
  return(rc);

} // End Lee_Cod_Transaccion()

//*************************** Establece_Conexion ************************
int Establece_Conexion(HPIPE pipe)
{
  int rc;
  time_t time_ini, time_fin;

  time(&time_ini);

  printf("\nSERVI001: Entra a realizar la conexion del pipe");
  // Se realiza la conexion al pipe de Envio de Resultados
  do {

    if ((rc = DosConnectNmPipe(pipe)) == 0)  // Retorna si logra Conexion
                        return (rc);

    time(&time_fin);     // Si no, determina si TIMEOUT ha expirado

  } while ( (time_fin - time_ini) < var_sys.time_out_pipe);

  printf("\n SERVI001: Expiro TIMEOUT de (%.3d) segs. rc(%d)", var_sys.time_out_pipe, rc);
  return (rc);

} // Fin de Establece_Conexion()


//*************************************************************************
int WaitRequerimiento(int n)
{
  int rc;

  // Espera hasta llegada de Requerimiento
  rc = DosSemWait(&SemHiloPerm[n], SEM_INDEFINITE_WAIT);
  if ( rc )
            fprintf(stderr,"\n SERVI001: Error SemWait SemHiloPerm[%d] (rc = %d)", n, rc);

  // Lo dejo listo para la proxima vez
  rc = DosSemSet(&SemHiloPerm[n]);
  if ( rc )
            fprintf(stderr,"\n SERVI001: Error SemSet SemHiloPerm[%d] (rc = %d)", n, rc);

  return(rc);
} // End funcion WaitRequerimiento()
```

```
//---------------------------------------------------------------------
void ShutDown(char *s)
{
   perror(s);
   //exit(3);
}



/*---------------------------------------------------------------------*/
int Run_Transaction (char *buf,int lenbuf, int n)
{
   int rc;
   int funlib_idx=0;
   int Fun_Lib (char *);

   funlib_idx = Fun_Lib(buf);
   if (funlib_idx != FUNLIBERROR) { //Existe la transacción
     rc = funlib[funlib_idx].ptrfun (&buf[OFFSET], (lenbuf-OFFSET), &(TablaPipes[n])->pipe_hand,n);
     return rc;
   }
   else
     return TRAN_NO_EXISTE;
} // fin de Run_Transaction()


/*---------------------------------------------------------------------*/


char *rtrim(char *string)
{
  char *ptr_string;

  ptr_string=string+strlen(string);
  for (ptr_string--;*ptr_string==' ';ptr_string--);
  *(++ptr_string)='\0';

return string;
}

/*---------------------------------------------------------------------*/
char *ltrim(char *string)
{
  char *ptr_string, *ptr_move, *edit;

  edit=(char*)malloc(strlen(string)+1);
  for (ptr_string=string;*ptr_string==' ';ptr_string++);
  for (ptr_move=edit;*ptr_string;*ptr_move=*ptr_string,ptr_string++,ptr_move++);
  *(ptr_move)='\0';
  strcpy(string,edit);
  free(edit);

return string;
}
```

```
/*-------------------------------------------------------------------*/
char *strnzcpy(char *dest,char *src,unsigned int len)
{
  strncpy(dest,src,len);
  *(dest+len)='\0';
  return dest;
}



/*-------------------------------------------------------------------*/
int unformat(char * string)
{
  char valor[30];
  int len,i,j;

  len=strlen(string);
  if(len)
  {
    for(i=0,j=0;i<len;i++)
          if((string[i] != '.')&&(string[i] != ','))
            valor[j++]=string[i];

    valor[j]='\0';
    strcpy (string,valor);
    return strlen(valor);
  }
  return 0;
}


/*-----------------------------------------------------*
Obtiene la la fecha y la hora del sistema y la retorna
AAAAMMDD HH:MM:SS
*-----------------------------------------------------*/
void GetFechayHora(char *fh)
{
  time_t ltime;
  struct tm *newtime;

  time( &ltime);                    /* get time as long integer */
  newtime = localtime( &ltime );    /* convert to local time */
  sprintf(fh, "%.4d%.2d%.2d %.2d:%.2d:%.2d",(newtime->tm_year+1900), (newtime->tm_mon+1),
                  newtime->tm_mday,newtime->tm_hour,newtime->tm_min,newtime->tm_sec);

} /* fin de GetFechayHora */

/*-------------------------------------------------------------------
copia N caracteres a T desde S[PI];
si BLANCOS != 0 rellenar con espacios en blanco hasta completar N caract.
si FLAG != 0 poner marca de fin de cadena '\0' a T
-------------------------------------------------------------------*/
void copystr(char *s, char *t, int pi, int n, int blancos, int fin)

{
  int i;

  for (i = 0; s[pi] != '\0'  &&  i < n; i++, pi++)
```

```
      t[i] = s[pi];
   for ( ; i < n  &&  blancos; i++)
            t[i] = ' ';
   if (fin)
     t[i] = '\0';
};  /* fin de copystr */


/* FIN DE SERVID00.C */
```

```
//*=============================================================================
//* PROGRAMA :  SERVI002.C
//* AUTOR    :  ROCIO COLOMBO
//* FUNCION  :  Funciones que ejecutan las TRANSACCIONES de
//*             GET_SALDOS_CLIE e INSERTA_DATOS_MOVIMS
//*=============================================================================


#include "SERVI002.H"
#include "funlib.h"

//***************** TRANSACCIONES UTILIZADAS POR SERVI000.C ***********


/*=============================================================================
// Funcion:  GET_SALDOS_CLIENTE
// Input:    data_in   ----> buffer de datos que envia el cliente
//                len_data_in     longitud de datos de data_in
//
// Return:   Los datos del cliente, con su saldo y sus respectivos
//           movimientos de la cuenta.
//=============================================================================*/

int GET_SALDOS_CLIE(char *data_in, int len_data_in, HPIPE *pipe_handle, int n)
{
 char       *data_out;
 char        cuenta[8];
 int  indice, find_cta,find_movs, ret, num_movs;
 int  bytes_escritos, bytes_sent, OFFSET;

 mov_cta     movs[100];
 dat_clie    data_clie;
 saldo_clie  data_saldo;

 Rewind_Archivos();

 printf("\n GET_SALDOS_CLIE  data_in:  -%s-  len: %d\n",data_in,len_data_in);

 OFFSET = BUFF_CLIE_SALDO+2;

 data_out = malloc(OFFSET+BUFF_MOVS);

 memcpy(cuenta,data_in,8);
 printf ("\n numero de la 'cuenta': %.8s",cuenta);

 printf("\n SERVI002: Obtiene los datos del cliente\n");

 find_cta = OBTIENE_DATOS_CLIENTE (cuenta, &data_clie, &data_saldo);

 switch (find_cta)
 {
    case NO_EXISTE_CTA:
                        strcpy(data_out,CLIENTE_NO_EXISTE);
              break;
```

```
    case ENCONTRO_CLIENTE:
            memcpy (data_out,&data_clie,REG_DAT_CLIE);
      memcpy (data_out + REG_DAT_CLIE, &data_saldo, REG_DAT_SALDO);
            break;
    default:
                    strcpy(data_out,TRAN_ERR);
            break;
  } // End Switch


  printf ("\n Antes de entrar a Obtiene_Movs");

  if (find_cta == ENCONTRO_CLIENTE) {
      find_movs = Obtiene_Movs(cuenta,movs,&num_movs);

      if (num_movs<0)
              num_movs =0;

            sprintf (data_out+BUFF_CLIE_SALDO,"%.2d",num_movs);

      switch (find_movs)
      {
        case TRAN_OK_FIN:
          for (indice=0; indice<num_movs; indice++)
                          memcpy(data_out+OFFSET+(REG_MOV_CTA*indice),
&movs[indice],REG_MOV_CTA);
                        *(data_out+OFFSET+(REG_MOV_CTA * num_movs) ) = '\0';
          break;
        case NO_HAY_MOVIMS:
          memcpy (data_out+REG_DAT_CLIE,NO_MOVIMS,LEN_COD_TRAN);
          break;
        default:
          memcpy(data_out+REG_DAT_CLIE,TRAN_ERR,LEN_COD_TRAN);
          break;
            } // End Switch find_movs

  } // End if ENCONTRO_CLIENTE

          bytes_sent= strlen(data_out);

  printf (" \nSERVI002:  Datos que son enviados: -%s-\n   bytes_sent: %d\n",data_out,bytes_sent);
  ret = DosWrite(*pipe_handle, data_out, bytes_sent, &bytes_escritos);


  if (ret){
            printf("\n Error DosWrite:ret = %i",ret);
            return(ret);
   }
   if (find_cta == NO_EXISTE_CTA)
      return (TRAN_ERROR);
   if (find_cta == ENCONTRO_CLIENTE)
      return (TRAN_OK_FIN);

} /* Fin de Get_Saldo_Clie*/
```

```
//=========================================================================
// ** FUNCION INSERTA_DATOS_MOVIMS:
// ** Esta funci¢n comprueba primero si existe la cuenta
// ** luego si el tipo de movimiento que se realiza es PAGO DE CHEQUES
// ** debe primero verificar el saldo, si ,ste es menor que el valor
// ** del cheque, regresa ERROR porque no hay suficientes fondos
// ** de lo contrario llama a ACTUALIZA_SALDO el cual resta del saldo la
// ** cantidad que se paga del cheque.
// ** Si el tipo de movimiento es DEPOSITO debe llamar a ACTUALIZA_SALDO
// ** el cual suma al saldo el valor del deposito.
// ** En ambos casos debe actualizar la fecha de la £ltima transacci¢n
//=========================================================================


int INSERTA_DATOS_MOVIMS(char *data_in,int len_data_in,HPIPE *pipe_handle,int n)
{
  char  data_out[7], sal_ant[13], sal_act[13], tipo_tran;
  char cuenta[9], var_cta[LEN_CTA];
  char fecha_mov[9], valor[13], num_doc[9],dat_saldo[REG_DAT_SALDO+1],saldo_cta[13];
  long value, saldo, saldo_ant, saldo_act;
  int      ret, band;
  int n_cod_tran, existe_cta, resp;
  int bytes_escritos, bytes_sent;

  Rewind_Archivos();

  memcpy(cuenta,data_in,8);
  printf("\n Numero de la cuenta ingresado: %.8s\n",cuenta);

  existe_cta = FALSE;


    // Obtiene los datos mandados por el cliente

    tipo_tran = *(data_in+8);
    printf ("\nTipo de la transaccion: %c\n",tipo_tran);
    sprintf (fecha_mov,"%.8s",data_in+9);
    printf ("\nfecha de movimiento:  %8s\n",fecha_mov);
    sprintf (valor,"%.12s",data_in+17);
    printf ("\nValor del cheque:  %.12s\n",valor);
    sprintf (num_doc,"%.8s",data_in+29);
    printf ("\nNumero del documento:  %8s\n",num_doc);
    band = getch();
    while (!feof(fp_saldo) )
    {
     if (existe_cta)
        break;

            if (fgets (dat_saldo, REG_DAT_SALDO + 1, fp_saldo ) == NULL) {
                    printf("\nLlego a fin de archivo saldo.txt\n");
                    memcpy (data_out, CLIENTE_NO_EXISTE, LEN_COD_TRAN);
                    break;
```

```
        }
        else            // 1
        {

                sprintf(var_cta,"%.8s",dat_saldo);
    printf ("\nNumero de la cuenta del archivo:  %.8s",var_cta);
                if ( strncmp(cuenta, var_cta, 8) == 0)  {
                        existe_cta = TRUE;
                        sprintf(saldo_cta,"%.12s",dat_saldo+8);
                        printf("\n Cuenta recuperada: %.8s\n",var_cta);
                        printf("\n Saldo recuperado: %.12s\n",saldo_cta);

                        value = atol(valor);
                        printf ("\nValor del cheque: %ld\n", value);
                        saldo = atol(saldo_cta);
                        printf ("\nSaldo de la cuenta: %ld", saldo);

                        if (tipo_tran == CHEQ)
                           n_cod_tran = N_PAGO_CHEQUES;
                        else
                           n_cod_tran = N_DEPOSITOS;

                        switch (n_cod_tran)  {
     // Si es Pago de Cheques entra a este lazo
                          case N_PAGO_CHEQUES:
                                printf ("\n Entra a N_PAGO_CHEQUES\n");
                                if (saldo >= value)  {
                                        saldo_ant = saldo;
                                        saldo_act = saldo_ant - value;
                        printf ("\nYa ha realizado las operaciones de saldo_ant y saldo_act\n");
                        sprintf (sal_ant, "%.12ld", saldo_ant);
                                        sprintf (sal_act, "%.12ld", saldo_act);
            printf  ("sal_ant:  %.12s    sal_act:  %.12s", sal_ant, sal_act);
                                        resp = INSERT_ACTUALIZA_DATOS
(cuenta, tipo_tran, fecha_mov, valor, num_doc, sal_act, sal_ant);

            printf ("\nSalio de INSERT_ACTUALIZA_SALDO");
                                }
                                else  {
                                        printf("\n No tiene suficientes fondos");
                                        memcpy (data_out, NO_HAY_FONDOS,
LEN_COD_TRAN);
                                }
                                break;

                          case N_DEPOSITOS:
                                printf("\n Entra a N_DEPOSITOS\n");
                                saldo_ant = saldo;
                                saldo_act = saldo + value;
            printf ("\nYa ha realizado las operaciones de saldo_ant y saldo_act\n");
sprintf (sal_ant, "%.12ld", saldo_ant);
                                        sprintf (sal_act, "%.12ld", saldo_act);
            printf  ("sal_ant:  %.12s    sal_act:  %.12s", sal_ant, sal_act);
```

```
                                        resp = INSERT_ACTUALIZA_DATOS (cuenta,
tipo_tran, fecha_mov, valor, num_doc, sal_act, sal_ant);
                                        break;

                                    default:
                                        break;
                                }// end case
                            } // end if existe cuenta
                        } // end else  1

            } // End while fp_saldo

    switch (resp)  {
            case ACTUALIZA_OK:
                            sprintf(data_out,"%.5s", TRAN_FINAL);
                            break;
            case NO_ENCONTRO_CTA:
                            sprintf(data_out,"%.5s", NO_ACTUALIZO_REG);
                            break;
      case TRAN_ERROR:
         sprintf (data_out,"%5s",NO_INSERTO_REG);
         break;
            default:
                            sprintf(data_out,"%.5s", TRAN_ERR);
                            break;
          } // End switch

    bytes_sent = strlen(data_out);

    ret = DosWrite(*pipe_handle, data_out, bytes_sent, &bytes_escritos);
    printf("\n Codigo retornado de DosWrite: (rc) %d",ret);

    if (ret){
            printf("\n Error DosWrite:ret = %i",ret);
            return(ret);
     }

     if (!existe_cta)
            return (TRAN_ERROR);
     if (resp == ACTUALIZA_OK)  {
       printf ("\nrespuesta retornada al Servidor:  %d",TRAN_OK_FIN);
       return (TRAN_OK_FIN);
     }

} // End Inserta_Datos_movims



//*********************************************************************
//************* FUNCIONES UTILIZADAS EN LAS TRANSACCIONES *************
//*********************************************************************

int INSERT_ACTUALIZA_DATOS(char *cta, char tipo, char *fecha_mv, char *valor, char *num_doc,
char *sal_act, char *sal_ant)
```

```
{
 //char data_out[7];
 int exito_inserta, exito_actualiza;


 printf ("\n Entro a INSERT_ACTUALIZA_DATOS");
 printf("\nNumero del documento ingresado:  %.12s", num_doc);
 printf ("\nAntes de ingresar a INSERTA_DATOS");
 exito_inserta = INSERTA_DATOS( cta, tipo, fecha_mv, valor, num_doc);

 printf ("\n Salio de INSERTA_DATOS Y DE INSERT_ACTUALIZA_DATOS\n");

 if (exito_inserta == TRAN_OK_FIN) {
          exito_actualiza = ACTUALIZA_SALDO (cta, sal_act, sal_ant, fecha_mv);
   printf ("\n Salio de ACTUALIZA_SALDO  respuesta: %d\n",exito_actualiza);
          return (exito_actualiza);
 }
 else
   return (exito_inserta);

}  // End INSERTA_ACTUALIZA_DATOS

//*************************************************************************
//************************* FUNCION INSERTA_DATOS  ***************************
//*************************************************************************

int INSERTA_DATOS (char *n_cuenta, char tip_mov, char *fecha, char *valor, char *doc)
{
   char nu_cta[9], n_fecha[9];
   int grabo_registro, retorno;

   printf("\nEntro a INSERTA_DATOS");
   grabo_registro =0;

   if ( fseek(fp_movim, 0L, SEEK_END) != 0 )
      return(TRAN_ERROR);
   else  {
          copystr(n_cuenta,nu_cta, 0,8,1,1);
              copystr(fecha,  n_fecha,0,8,1,1);
      printf ("\n Valores que van a ser ingresados al archivo de movimientos");
      printf ("\n cuenta:  %.12s   fecha:  %.8s, valor:  %.12s,  num_doc:  %.8s", nu_cta, n_fecha, valor,
doc);
                        retorno = fprintf(fp_movim, "%.8s%c%.8s%.12s%.8s\n", nu_cta, tip_mov,
n_fecha, valor, doc);
      if (retorno < 0)  {
        printf ("\nOcurrio un error: %d",retorno);
        return (retorno);
      }
      else  {
        printf ("\nRetorno de fprintf:  %d",retorno);
                grabo_registro = TRUE;
        fclose(fp_movim);
        fp_movim = fopen("movims.txt","a+");
        return (TRAN_OK_FIN);
```

```
      }
    }

    if (!grabo_registro )
       return (TRAN_ERROR);

} // End Inserta Datos



/* ************************************************************************
   ******************** FUNCION ACTUALIZA_SALDO
   ************************************************************************/

int ACTUALIZA_SALDO (char *n_cuenta, char *sal_act, char *saldo_ant, char *fecha)
{
  char  *dat_saldo;
  char    var_cta[9];
  saldo_clie *ptr_saldo;
  int rc;
  long tell, tell1;


  printf ("\n Entro a ACTUALIZA_SALDO\n");

  rewind (fp_saldo);
  dat_saldo = malloc(REG_DAT_SALDO+1);

  ptr_saldo = (saldo_clie *) dat_saldo;
  printf ("\nAntes de fread dat_saldo");

  fread (dat_saldo, REG_DAT_SALDO+1, 1, fp_saldo);


  while (!feof(fp_saldo)) {
    memcpy(var_cta, ptr_saldo->num_cta, 8);
    printf ("\nNumero de la cuenta a comparar: %.8s   cuenta recuperada: %.8s",n_cuenta, var_cta);
    if (strncmp(n_cuenta, var_cta, 8) == 0) {
       sprintf(ptr_saldo->sal_actual,   "%.12s", sal_act);
               sprintf(ptr_saldo->sal_ant,      "%.12s", saldo_ant);
               sprintf(ptr_saldo->fecha_ult_mov,"%.8s",  fecha);
               fseek(fp_saldo, -REG_DAT_SALDO-2 , SEEK_CUR);
       tell1 = ftell(fp_saldo);
       printf ("\n posicion2 del puntero de fp_saldo:  %ld",tell1);
       printf ("\n Va a actualizar los datos del saldo");
               rc = fwrite(dat_saldo, REG_DAT_SALDO, 1, fp_saldo);
       if (rc < 1 ) {
         printf ("\nOcurrio un error en fwrite:  %d", rc);
         return (TRAN_ERROR);
       }
       else {
         printf ("\n cantidad de datos escritos en saldo.txt: %d",rc);
         printf ("\nSalio de grabar en el archivo saldo.txt");
         fclose (fp_saldo);
```

```
        fp_saldo = fopen ("saldo.txt","r+");
        return (ACTUALIZA_OK);
      }

   }
   tell = ftell(fp_saldo);
   printf ("\n posicion1 del puntero de fp_saldo:  %ld",tell);
   fread (dat_saldo, REG_DAT_SALDO+1, 1, fp_saldo);

  } // End while fp_saldo

  return (NO_ENCONTRO_CTA);

} // End Actualiza_Saldo


// *****************************************************************
// FUNCION:  OBTIENE_DATOS_CLIE
// INPUT  :  numero de la cuenta
// OUTPUT :  Datos del cliente y Datos de su saldo
// SALIDA :  NO_EXISTE_CTA
//           ENCONTRO_CLIENTE
// *****************************************************************

int OBTIENE_DATOS_CLIENTE (char *num_cuenta, dat_clie *datos_clie, saldo_clie *saldo)

{
   char var_cta[9], var_ced[12], cedula[LEN_CED+1];
   char datos[REG_DAT_CLIE+1], dat_saldo[REG_DAT_SALDO+1];


   printf ("\nEntr¢ a OBTIENE_DATOS_CLIENTE, antes de entrar a fp_saldo\n");
          while (!feof(fp_saldo))
          {
            if (fgets (dat_saldo, REG_DAT_SALDO + 1, fp_saldo ) == NULL)  {
                printf("\nLlego a fin de archivo saldo.txt");
                return (NO_EXISTE_CTA);
              }
              else       // 1
              {
                 printf ("\nSERVI002:  datos del saldo   -%s-", dat_saldo);
                 memcpy(var_cta,dat_saldo,8);
                 if (strncmp(num_cuenta, var_cta, 8) == 0)
                 {
                       memcpy(saldo, dat_saldo, REG_DAT_SALDO);
                       memcpy(var_ced,dat_saldo+OFFSET_CED,11);
                       printf("\n # de la cedula: %.11s\n",var_ced);
                       while (!feof(fp_datos))
                       {

                    if (fgets(datos,REG_DAT_CLIE+1,fp_datos) == NULL)
                          {
                                printf("\n Llego a fin de archivo datos.txt\n") ;
                                return 0;
```

```
                        }
                        else { // 2

                    memcpy(cedula,datos,11);
                if (strncmp(cedula,var_ced,11) == 0)  {
                                memcpy(datos_clie, datos, REG_DAT_CLIE);
                   printf("\n Datos obtenidos del cliente  -%s-\n",datos_clie);
                                return (ENCONTRO_CLIENTE);
                }
                    } // end else 2

          } // End while (fp_datos)

        } // End If

      }  // end else  1

          } // End while  (fp_saldo)



} // End Obtiene_Datos_Cliente

// ---------------------------------------------------------------------

// FUNCION PARA OBTENER LOS MOVIMIENTOS DE LA CUENTA DEL CLIENTE

// ---------------------------------------------------------------------

int Obtiene_Movs( char *num_cuenta, mov_cta *data_movs, int *num_movs)
{
  int i, existe_movs;
  char var_cta[LEN_CTA+1], reg_movs[REG_MOV_CTA+1];

  i=0; existe_movs=0; *num_movs =0;

  do {

    if ( fgets (reg_movs, REG_MOV_CTA + 1, fp_movim) == NULL) {
       printf("\n Fin de archivo movs.txt \n");
       if (num_movs != 0)
          return (TRAN_OK_FIN);
       return (NO_HAY_MOVIMS);
    }
    else {
       memcpy (var_cta,reg_movs,8);

       if (strncmp(num_cuenta,var_cta,8) == 0)
       {
                memcpy (&data_movs[i], reg_movs, REG_MOV_CTA);
                (*num_movs) ++;
                existe_movs = TRUE;
                i = i+1;
```

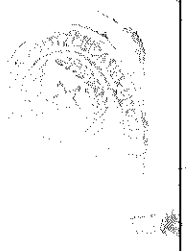
```
        }
      } // End Else

    } while (!feof(fp_movim)); // End While


 } // End  Obtiene_Movs

// ---------------------------------------------------------------------

void Rewind_Archivos()
{

    printf ("\n Inicializando todos los archivos de datos\n");
    rewind (fp_datos);

    rewind (fp_movim);
    rewind (fp_saldo);

}
```

MODULO LLAMADO POR EL PROGRAMA SERVI002.C

```
//*********************************************************************
// SERVI002.H
// header file para SERVI002.C
// Contiene prototipos de las funciones para invocar a las transacciones
// de Cliente-Servidor
// *********************************************************************

// ************** DEFINICIONES PARA EL COMPILADOR
 #define INCL_DOS
 #define INCL_DOSNMPIPES


// ***************  ARCHIVOS DE CABECERA *******************************

#include <os2.h>
#include <dos.h>
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stddef.h>

#include <memory.h>
#include <string.h>
#include <process.h>
#include <malloc.h>
#include <direct.h>
#include "stru_dat.h"

// *********************************************************************
//            CONSTANTES DEL PROGRAMA
// *********************************************************************

#define OR ||
#define AND &&



#define  CONSULTA_SALDOS   "01000"
#define  ACTUALIZA_CTA     "02000"

#define  CHEQ      '1'
#define  DEPO      '2'

#define  N_PAGO_CHEQUES      1
#define  N_DEPOSITOS         2


#define   REG_DAT_CLIE              77
#define   REG_MOV_CTA               37
#define   REG_DAT_SALDO     52
```

```
#define  BUFF_CLIE_SALDO  129
#define  BUFF_MOVS       (20*REG_MOV_CTA)+3
#define  OFFSET_CED       40


#define  TRUE                          1
#define  FALSE          0
#define  LEN_CTA         8

#define  LEN_CED         11

#define  EXISTE_CTA       1

#define  NO_EXISTE_CTA     0

#define  ENCONTRO_CLIENTE   1

#define  NO_ENCONTRO_CLIENTE 0

#define  LEN_COD_TRAN      5

// ******** CODIGOS DE RETORNO DE LAS TRANSACCIONES

#define  CLIENTE_NO_EXISTE    "01011"
#define  TRAN_ERR          "01021"
#define  TRAN_NO_EXISTE      "02011"
#define  NO_MOVIMS          "02020"
#define  NO_HAY_FONDOS       "03010"
#define  NO_INSERTO_REG      "04010"
#define  TRAN_FINAL         "05010"
#define  NO_ACTUALIZO_REG     "04020"


#define  TRAN_ERROR     3

#define  OK           1

#define  NO_HAY_SALDO    2

#define  ACTUALIZA_OK    4

#define  TRAN_OK_FIN     22

#define  NO_ENCONTRO_CTA  6

#define  NO_HAY_MOVIMS   7

// ************ VARIABLES GLOBALES ********************

FILE *fp_datos;

FILE *fp_saldo;

FILE *fp_movim;
```

```
// ******************************************************
// Definicion de funciones
// ******************************************************


int GET_SALDOS_CLIE(char *, int, HPIPE *, int);      //"01010"
int INSERTA_DATOS_MOVIMS(char *,int,HPIPE *,int); //"02010"

int INSERT_ACTUALIZA_DATOS(char *,char ,char *,char *,char *,char *,char * );
int  ACTUALIZA_SALDO(char *,char *,char *, char *);
int  INSERTA_DATOS(char *,char,char *, char *,char *);
int  OBTIENE_DATOS_CLIENTE(char *, dat_clie *, saldo_clie *);
int  Obtiene_Movs(char *,mov_cta *,int *);

void copystr (char *s,char *t,int pi, int n, int blancos, int fin);
void Rewind_Archivos( void );
```



// ==========

// CONFIGURACION EN EL SERVIDOR
// NOMBRE: SERVIDOR.CFG

// ==========

MAX_IN_PIPE 2
MAX_HILOS_PERM 3
MAX_HILOS_TEMP 2
TIME_OUT_PIPE 30
NAMAPLCC CONS
SERVER \\R0121503
PIPENAME TESIS

PROGRAMA CLIENTE (ESTACION) BASADO EN NAMED PIPES

```
//
//=============================================================================
// PROGRAMA   : ESTACION.C
// FUNCION    : Funciones para invocar a las TRANSACCIONES de Consulta de
//              Saldos, Pago de Cheques y Depositos.
// COMPILAR   :
// FECHA      :
//
//=============================================================================

#include "stdio.h"
#include "ctype.h"
#include "estac001.h"

// ****************************************************************
// **************   VARIABLES GLOBALES  ***************************
// ****************************************************************

char pipe_inicial[80];
struct_var_sys var_sys;

// ****************************************************************
// ***************  FUNCION PRINCIPAL  ****************************
// ****************************************************************

main()
{
   int   opcion, rc, d;

// ---- Lee configuracion ----
   Init_Aplic ("ESTACION.CFG");

// ---- Forma nombre del Pipe:
//      Input : se toma este nombre tal como es
//      Output: se le concatena 01, 02, .... nn, hasta el max. de Out Pipes
   strcpy (pipe_inicial, var_sys.server);
   strcat (pipe_inicial, var_sys.name_pipe);


   for (;;) {
      d = getch();
         opcion = Pide_Opcion();

               switch (opcion){
                           case 1:
            rc = Consulta_Saldos();
            if (rc)
             Analiza_Error(rc);
                                       break;
                           case 2:
            rc = Pago_Cheques();
            if (rc)
```

```
            Analiza_Error(rc);
                                    break;
                        case 3:
            rc =  Depositos();
            if (rc)
             Analiza_Error(rc);
            break;
                        case 4:  exit(0);
              }
   }
} /* End Main */

// ***************** FUNCIONES UTILIZADAS EN EL PROGRAMA PRINCIPAL
*******************

int Pide_Opcion(void)
{
  int  option, c;

  system ("CLS");

  printf ("============================\n\n");
  printf ("1: CONSULTA DE SALDOS. \n\n");
  printf ("2: PAGO DE CHEQUES.\n\n");
  printf ("3: DEPOSITOS.\n\n");
  printf ("4: SALIR \n\n");
  printf ("============================\n\n");

  printf ("\n Seleccione opcion:");


  do{
    c= getch();
    option = c -'0';
    printf("%c", c);
  } while (option < 1  OR  option > 4);

  return(option);

} /* END PIDE_OPCION */

// ==========================================================================
// FUNCION:  Consulta_Saldos
// OBJETIVO:  Muestra al cliente el saldo de su cuenta.
// ==========================================================================

int  Consulta_Saldos()
{
   int   rc, i, indice, c,  bytes_leidos, bytes_sent, offset, f;
   long int status;
   char  numero[9], out_pipe_name[80], num_cta[9], data_received[BUFF_SALDO+1];
   char  ptr_data_received [MAX_BUFF_IN];
   char  *inicio_data, data_sent[14], cod_tran[6];
   USHORT  pipe_handle;
```

```
char  S_error[ERR_MSG_SIZE], indice_aux[2];

system ("CLS");

printf("\n\nINGRESE SU NUMERO DE CUENTA:");
gets(numero);
printf("   %.8s\n",numero);

// *****INTENTA CONEXION INICIAL *********
memset(out_pipe_name, 0, 80);
if ((rc = Conexion_Inicial(out_pipe_name)) == OK)  {
  printf("ESTACION: Nombre del pipe: -%s-\n", out_pipe_name);

  if ((rc = Abrir_Sesion(out_pipe_name, &pipe_handle)))
            return (rc);
}
else
  return(rc);

// ***** PRIMER REQUERIMIENTO  ************
strncpy(cod_tran, CONSULTA_SALDO, 5);

// ***** FORMA BUFFER DE ENVIO AL PIPE  ************
copystr (numero, num_cta, 0, 8, 1, 1);  // Completa blancos al final

sprintf (data_sent,"%.5s%.8s", cod_tran, num_cta,'\0');


bytes_sent = strlen(data_sent);

// **** INICIA LAZO DE REQUERIMIENTOS (hasta obtener FIN_DE_DATOS ***

printf("\nEntra a Lazo de Requerimientos:\n data_sent:  -%s-\n ptr_data_received: -%s-", data_sent,
ptr_data_received);
printf("\n Bytes_sent de Lazo de Req:  %d", bytes_sent);

          if (rc = Envia_Requerimiento(data_sent, ptr_data_received, bytes_sent, &bytes_leidos,
pipe_handle)) {
            printf ("\n ESTAC001: No pudo establecer comunicacion con el server");
            return(rc);
          }

    printf("\nEntra a Check_Status\n");
    printf("\nDatos que ingresan a Check_Status:  -%s-, S_error -%s-",ptr_data_received,S_error);

    status = Check_Status(ptr_data_received, S_error);
   //  No hay Codigo ==> OK: Mas Datos    ==> Fin de Datos

     printf("\n Status: %d\n",status);

   // Copia data_received a Estructura de Resultado *****
    printf ("\n Va a copiar la Estructura de Resultado\n");
   if (status == OK) {
```

```
        memcpy (clientes.ced_cli,   ptr_data_received,   11);
        memcpy (clientes.nombre,    ptr_data_received+11,30);
        memcpy (clientes.direccion, ptr_data_received+41,30);
        memcpy (clientes.telefono,  ptr_data_received+71, 6);

        printf("\n Ya copio los datos del cliente en la estructura clientes\n");
        memcpy (saldos.num_cta,      ptr_data_received+LEN_REG_DAT,8);
        memcpy (saldos.sal_actual,   ptr_data_received+85,       12);
        memcpy (saldos.sal_ant,      ptr_data_received+97,       12);
        memcpy (saldos.fecha_ult_mov, ptr_data_received+109,     8);
        memcpy (saldos.num_ced,      ptr_data_received+117,      11);
        saldos.status_cta = (char) *(ptr_data_received+128);

        printf("\n Ya copio los datos del saldo en la estructura saldos\n");

        memcpy (indice_aux,       ptr_data_received+LEN_REG_DAT_SAL,2);

        indice= atoi(indice_aux);

        printf ("\n Ya copio el indice de los movimientos  %d\n",indice);
//    Inicializamos las variables utilizadas en movimientos *******

    inicio_data = ptr_data_received + CABECERA;

    offset = 0;

    if( indice ==0)
            printf ("\n No se han registrado Movimientos");
    else
            for (c = 0; c < indice; c++)
            {
        memcpy (&movi_cta[c], inicio_data+offset, LEN_MOV);
        /*
              memcpy (movi_cta[c].num_cta, inicio_data+offset, 8);
              memcpy (movi_cta[c].tip_mov, inicio_data + 8 + offset, 1);
        printf ("\n tipo del movimiento: %c\n",movi_cta[c].tip_mov);
              memcpy (movi_cta[c].fec_mov, inicio_data + 9 + offset, 8);
              memcpy (movi_cta[c].monto,   inicio_data +17 + offset,12);
              memcpy (movi_cta[c].num_doc, inicio_data +26 + offset, 8);
        */
        offset += LEN_MOV;
            } // End For

    printf ("\n Antes de Mostrar los Datos del Cliente\n");
    // f = getch();

    Mostrar_Datos_Clie (indice);

 } // end if
    printf ("\n Presione cualquier tecla para continuar\n");
    f = getch();
    Cerrar_Sesion(pipe_handle);
    if (status == OK)
       return(TRAN_OK_FIN);
```

```
    else
                return(status);

} // End Consulta_Saldos

// ============================================================================
// FUNCION:  Pago_Cheques
// OBJETIVO: Ingresar el valor y n£mero del cheque
//         Comprobar si puede ser pagado,
// ============================================================================

int Pago_Cheques()
{
          int    rc, bytes_leidos, bytes_sent, ret, status;
          char   data_sent[43], cod_tran[6];
          USHORT  pipe_handle;
 char   ptr_data_received[MAX_BUFF_IN], S_error[ERR_MSG_SIZE];
          char   out_pipe_name[80], tip_mov;
          char   NUM_CTA[9], num_cheq[9], valor[13], fech[9];
 char       n_cta[9], n_cheq[9], n_valor[13], n_fech[9];
 long   aux;

 system ("CLS");

 //   Presentaci¢n de datos a ingresar

          printf("\n\nINGRESE EL NUMERO DE SU CUENTA: ");
          gets(NUM_CTA);
          printf("  %.8s\n",NUM_CTA);
          printf("DIGITE NUMERO DEL CHEQUE:");
          gets(num_cheq);
 aux = atol(num_cheq);
 sprintf(n_cheq,"%.8ld", aux);
 printf("  %.8s\n",n_cheq);
          printf("VALOR EN SUCRES: S/.");
          gets(valor);
 aux = atol(valor);
 sprintf(n_valor,"%.12ld",aux);
 printf("  %.12s\n",n_valor);

          printf("FECHA DEL MOVIMIENTO (mm/dd/aa):");
          gets(fech);
          printf("   %.8s\n",fech);

          tip_mov = CHEQ;

// ************ INTENTA CONEXION INICIAL ************************
 memset(out_pipe_name, 0, 80);
          if ((rc = Conexion_Inicial(out_pipe_name)) == OK)  {
             if ((rc = Abrir_Sesion(out_pipe_name, &pipe_handle)))
                    return (rc);  // Retorna si hay algun error
             }
          else
             return(rc);
```

```
// ***********************************************************

// ************** PRIMER REQUERIMIENTO *************************

  strncpy(cod_tran, ACTUALIZA_CTA, 5);

// ***********************************************************


//********** FORMA BUFFER DE ENVIO AL PIPE *****************
   copystr(NUM_CTA, n_cta,  0, 8,1,1);
   copystr(fech,    n_fech, 0, 8,1,1);


   sprintf(data_sent,"%.5s%.8s%c%.8s%.12s%.8s",cod_tran, n_cta, tip_mov, n_fech, n_valor,n_cheq);

   bytes_sent = strlen(data_sent);

// **** INICIA LAZO DE REQUERIMIENTOS (hasta obtener FIN_DE_DATOS ***

   printf("\nEntra a Lazo de Requerimientos:\n data_sent:  -%s-\n ptr_data_received: -%s-", data_sent,
ptr_data_received);
   printf("\n Bytes_sent de Lazo de Req:  %d", bytes_sent);

   if ( ret = Envia_Requerimiento(data_sent,ptr_data_received, bytes_sent,&bytes_leidos,pipe_handle) )
   {
            printf("\nERROR:  No se pudo realizar Transmision");
            Cerrar_Sesion(pipe_handle);
            return(ret);
    }

    printf("\nEntra a Check_Status\n");
    printf("\nDatos que ingresan a Check_Status:  -%s-, S_error -%s-",ptr_data_received,S_error);

    status = Check_Status(ptr_data_received, S_error);
    //  No hay Codigo ==> OK: Mas Datos    ==> Fin de Datos

    printf("\n Status: %d\n",status);
    if (status == TRAN_FINAL)
            return (TRAN_OK_FIN);
    else
            return (status);

} //  Fin Pago_Cheques

// ===========================================================================
// FUNCION:  Depositos
// OBJETIVO:
// ===========================================================================

int Depositos()
{
        int rc, ret, bytes_leidos, bytes_sent, status;
```

```
        char data_sent[43], cod_tran[6];
        char ptr_data_received[MAX_BUFF_IN], S_error[ERR_MSG_SIZE];
        USHORT  pipe_handle;
        char  out_pipe_name[80],tip_mov;
        char  NUM_CTA[9], num_papel[9], valor[13], fech[9];
        char  n_cta[9], n_papel[9], n_valor[13], n_fech[9];
long  aux;

 system ("CLS");

        printf("\n=====================================\n");
        printf("INGRESE EL NUMERO DE SU CUENTA: ");
        gets( NUM_CTA);
        printf(" %.8s\n",NUM_CTA);
        printf("DIGITE NUMERO DEL DEPOSITO:");
        gets( num_papel);
 aux = atol(num_papel);
 sprintf(n_papel,"%.8ld", aux);
        printf(" %.8s\n",n_papel);

        printf("VALOR EN SUCRES: S/.");
        gets(valor);
 aux = atol(valor);
 sprintf(n_valor,"%.12ld", aux);
        printf(" %.12s\n",n_valor);

        printf("FECHA DEL MOVIMIENTO (mm/dd/aa):");
        gets(fech);
        printf(" %8s\n",fech);
        printf("=====================================");

        tip_mov = DEPO;

// ************ INTENTA CONEXION INICIAL ************************
memset(out_pipe_name, 0, 80);
        if ((rc = Conexion_Inicial(out_pipe_name)) == OK) {
          if ((rc = Abrir_Sesion(out_pipe_name, &pipe_handle)))
                return (rc);  // Retorna si hay algun error
        }
        else
          return(rc);

// ************************************************************

// ************** PRIMER REQUERIMIENTO  *************************

  strncpy(cod_tran, ACTUALIZA_CTA, 5);

// ************************************************************


// ************** FORMA BUFFER DE ENVIO AL PIPE *****************

   copystr(NUM_CTA,  n_cta,  0, 8,1,1);
```

```
    copystr(fech,    n_fech, 0, 8,1,1);

    sprintf(data_sent,"%.5s%.8s%c%.8s%.12s%.8s",cod_tran, n_cta, tip_mov, n_fech, n_valor, n_papel);

    bytes_sent = strlen(data_sent);

    // **** INICIA LAZO DE REQUERIMIENTOS (hasta obtener FIN_DE_DATOS ***

    printf("\nEntra a Lazo de Requerimientos:\n data_sent:  -%s-\n ptr_data_received: -%s-", data_sent,
ptr_data_received);
    printf("\n Bytes_sent de Lazo de Req:  %d", bytes_sent);

    if ( ret = Envia_Requerimiento(data_sent, ptr_data_received, bytes_sent, &bytes_leidos, pipe_handle)
)
    {
            printf("\nERROR:  No se pudo realizar Transmision");
            Cerrar_Sesion(pipe_handle);
            return(ret);
    }

    printf("\nEntra a Check_Status\n");
    printf("\nDatos que ingresan a Check_Status:  -%s-, S_error -%s-",ptr_data_received,S_error);

    status = Check_Status(ptr_data_received, S_error);
    //  No hay Codigo ==> OK: Mas Datos    ==> Fin de Datos

    printf("\n Status: %d\n",status);

    if (status == TRAN_FINAL)
            return (TRAN_OK_FIN);
    else
            return (status);


} //  Fin Depositos



// ************************************************************************
// ******************** FUNCIONES UTILIZADAS ******************************
// ************************************************************************


// ========================================================================
//  FUNCION:  Abrir Sesion
// ========================================================================

int Abrir_Sesion(char *out_pipe_name, USHORT *pipe_handle)
{
        int  ret, i;
         unsigned int accion;
```

```
        // ABRO EL PIPE
          ret = DosOpen(out_pipe_name, pipe_handle, &accion, 2200L,
              FILE_NORMAL, FILE_OPEN, OPEN_ACCESS_READWRITE|
OPEN_SHARE_DENYREADWRITE,(ULONG)0L);
  printf("Abrir_Sesion (1): error DosOpen: -%d-\n", ret);
          if (ret == 0)
            return (ret);  // Retorna si tiene exito al Abrir el pipe

        // Si falla Intento Inicial, Realiza 3 intentos adicionales( cada 5 seg)
          for (i=0; i<3; i++) {
            if(( ret= DosWaitNmPipe(out_pipe_name,5000L)) == 0) {

              ret = DosOpen(out_pipe_name, pipe_handle, &accion, 2200L,FILE_NORMAL,
                        FILE_OPEN, OPEN_ACCESS_READWRITE |
OPEN_SHARE_DENYREADWRITE, (ULONG)0L);

      printf("Abrir_Sesion (2): error DosOpen  %d \n", ret);
              if (ret ==0)
                        return(ret);  // Retorna si tiene exito al abrir el pipe
            }
            else  {
              printf("\n Error: DosWaitNmPipe -%s- rc(%d)", out_pipe_name,ret);
              return(ret);
         }
          }// End for()

        // Si fallan todos los intentos, Retorna Error
        if (ret) {
            printf ("\n Abrir_Sesion (3) Error: DosOpen de PIPE  -%s- rc(%d)",out_pipe_name,ret);
            printf ("\n       Error de Comunicacion con el Servidor :");
           return(ret);
  }


} // End Abrir_Sesion

// ******************************************
// FUNCION:  Conexion_Inicial()

// ******************************************

int Conexion_Inicial (char *out_pipe_name)
{
  int  ret, bytes_sent, bytes_leidos;
  char  data_sent[40], data_received[80];

  bytes_sent =5;
  strcpy(data_sent, REQ_INICIAL);


 printf("\nRealiza Conexion Inicial: In_Pipe -%s-\n", pipe_inicial);
 ret = DosCallNmPipe(pipe_inicial, data_sent, bytes_sent, data_received,
                                    MAX_BUFF_IN, &bytes_leidos,TIMEOUT);
 if (ret) {
```

CONFIGURACION PARA LA ESTACION : ESTACION.CFG

SERVER \\R0121503\PIPE\
PIPENAME TESIS

// FUNLIB.C

```
/* Esta funciòn retorna el ìndice del elemento del vector funlib[] en el caso
   de que el còdigo de la transacciòn se encuentre definido; de lo contrario,
   retornar -1 (ERROR)
*/

#include "servi000.h"
#include "funlib.h"

extern int GET_SALDOS_CLIE       (char *, int, HPIPE *,int );
extern int INSERTA_DATOS_MOVIMS  (char *, int, HPIPE *,int );


trans_func funlib[] = {
      {"01000", GET_SALDOS_CLIE },
      {"02000", INSERTA_DATOS_MOVIMS }

};

#define MAX_TRANS      (sizeof(funlib) / sizeof(trans_func))


int Fun_Lib (char *transid)
{
  int i, fnresult = FUNLIBERROR;

  for (i=0; i<MAX_TRANS; ++i)
    if (strncmp (transid, funlib[i].trans_id, COD_TRANS) == 0) {
      fnresult = i;
      break;
    }

  return (fnresult);
}
```

```
// ******************************************
// FUNLIB.H
// ******************************************
/* Definiciones para la libreria de funciones FUNLIB.C */

#define  COD_TRANS      5  /* Tama□o del c¢digo de transacci¢n */
#define  FUNLIBERROR    -1  /* El c¢digo de transacci¢n no existe  */


/*
#include <os2.h>
#include <dos.h>
#include <time.h>
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
*/
#include <stdio.h>



typedef struct  trans_func  {
        char trans_id[COD_TRANS+1];
        int (*ptrfun)  (char *, int, HPIPE *,int);
} trans_func;

int Fun_Lib (char *);
```

```
            // printf("\nError en DosCallNmPipe rc = %d", ret);
        return (ret);
    }
    data_received[bytes_leidos] = '\0';
    printf("\nOutPipe Recibido: -%s- len %d", data_received, bytes_leidos);

            if (strncmp(data_received,PIPE_DISPONIBLE, 5) == 0) {
              // Forma y retorna OUTPIPE = SERVERID + OutPipe Recibido
              //  strcpy(out_pipe_name, serverid);
              // Descarta  5 bytes iniciales del Codigo de Retorno
                strcat(out_pipe_name, data_received + 5);
                return (OK);
             }

            return (ERROR_CONEXION_INI);

} // End Conexion_Inicial()

// ****************************************************************

// FUNCION:  Envia_Requerimiento()

// ****************************************************************

int Envia_Requerimiento (char *data_sent, char *data_received, int bytes_sent,
                                        int *bytes_leidos, USHORT pipe_handle)
{
    int n, i, ret;

  // ** printf() de los datos a ser enviados
            putchar('\n');
               n= (bytes_sent > 200) ? 200 : bytes_sent;
               for (i=0; i < n; i++)
                   putchar(data_sent[i]);

             // Envia Buffer a Server
             ret = DosTransactNmPipe(pipe_handle, data_sent, bytes_sent,
                      data_received, MAX_BUFF_IN, bytes_leidos);
             printf("\nenvie datos\n");
             data_received[*bytes_leidos] = '\0';
             printf("\n Recibido de Envia_Req: -%s- len %d", data_received, *bytes_leidos);
             if (ret)  {
                              printf("\nError en DosTransactNmPipe rc = %d", ret);
                              return(ret);
              }
              return (0);

} // End Envia_Requerimiento()


// ****************************************************************
// FUNCION:  CerrarSesion
// ****************************************************************
```

```
void Cerrar_Sesion(USHORT pipe_handle)
{
   DosClose(pipe_handle);
} // end Cerrar_Sesion


// ************************************************************
// FUNCION:  Check_Status()
// ************************************************************

int Check_Status(char *data_received, char *S_error)
{
        int retorno;
         char aretorno[10];

         memset(aretorno, 0, 10);
  printf ("\nEntro a Check_Status\n");
  printf ("\n Datos recibidos en Check_Status: -%s-", data_received);
         strncpy( aretorno, data_received, LEN_COD_TRAN); //Copia COD. de Retorno
  printf ("\n aretorno: -%s-",aretorno);
         retorno = atoi(aretorno);


  switch (retorno) {
            case TRAN_FINAL:
                     printf ("\n CONSULTA: Transaccion OK. Fin de Datos");
                     break;
            case CLIENTE_NO_EXISTE:
                    // printf("\n  CONSULTA: ERROR: Clave No Existe.");
                     memcpy(S_error, data_received + LEN_COD_TRAN, ERR_MSG_SIZE);
                     break;
            case TRAN_ERR:
                    // printf("\n CONSULTA: ERROR: Clave Duplicada (ya existe).");
                     memcpy(S_error, data_received + LEN_COD_TRAN, ERR_MSG_SIZE);
                     break;
            case NO_HAY_FONDOS:
                    // printf("\n CONSULTA: ERROR: No se pudo procesar Transaccion.");
                    // printf("\n          Algunos de los Datos dados no son validos");
                     memcpy(S_error, data_received + LEN_COD_TRAN, ERR_MSG_SIZE);
                     break;
            case NO_INSERTO_REG:
                     memcpy(S_error, data_received + LEN_COD_TRAN, ERR_MSG_SIZE);
                     break;
            case TRAN_NO_EXISTE:
                     memcpy(S_error, data_received + LEN_COD_TRAN, ERR_MSG_SIZE);
                     break;

            case FIN_DE_DATOS:
                    // printf("\n CONSULTA: Fin de Transaccion. Fin de Datos.");
                     break;
            case TRAN_PENDIENTE:
                    // printf("\n CONSULTA: ERROR: No se pudo procesar Transaccion.");
                    // printf("\n                Existe una Transaccion pendiente.");
                     break;
```

```
                default:
                                printf("\nCONSULTA DE SALDO: No Retorna un codigo de error: Codigo de
Retorno No Esperado.\n");
                                retorno = OK;
                                break;
                } // End Switch


        return (retorno);

} // End Check_Status



void Mostrar_Datos_Clie (int num_movs)
{
 int indice;

    // Mostrar datos en pantalla al cliente
    printf("   DATOS PERSONALES:\n\n");
    printf("NOMBRE   :  %.30s\n",clientes.nombre);
    printf("DIRECCION:  %.30s\n",clientes.direccion);
    printf("TELEFONO :  %.6s\n", clientes.telefono);
    printf("# CEDULA :  %.11s\n",clientes.ced_cli);

    printf("\nFECHA DEL ULTIMO MOVIMIENTO:   %.8s",saldos.fecha_ult_mov);
    printf("\nSALDO ACTUAL: %.12s   SALDO ANTERIOR:
%.12s\n",saldos.sal_actual,saldos.sal_ant);
    printf("\nESTADO DE CUENTA       \n");
    printf("\nFECHA     TIPO      VALOR       NUM_DOC   ");

    if (num_movs == 0)
      printf("\n\n Actualmente no hay movimientos en su cuenta\n");
    else
      for  (indice = 0; indice < num_movs ; indice++){
          printf("\n %.8s    ",movi_cta[indice].fec_mov);
          printf(" %c     " , movi_cta[indice].tip_mov);
          printf(" %.12s       ", movi_cta[indice].monto);
          printf(" %.8s \n", movi_cta[indice].num_doc);
       }


 }  // Fin de Mostrar_Datos_Clie

/*-------------------------------------------------------------------------*/

int search_label (char *token)
{
int i;

for (i = 0; labels_id[i][0]; i++)
   if (strcmp(token,labels_id[i])==0)
      return i;
return -1;
```

```
} // fin de search_label()


//-------------------------------------------------------------------

int Init_Aplic(char *fnam)
{
  FILE *fp;
  char line[MAXLINE],label[20],value[MAXLINE],*ptrline;
  int id;

  if ((fp = fopen(fnam, "r")) == NULL) {
    perror("fopen");
    printf ("Error Archivo de Configuraci¢n no Existe [%s]\n",fnam);
    exit(1);
  } // endif

  while (fgets(line, MAXLINE, fp)) {
   ptrline=strchr(line,' ');

   if (ptrline) {
    line[strlen(line)-1]='\0';
    strnzcpy(label, line, ptrline-line);

    if ((id=search_label(label))!=-1) {
                strncpy (value, ptrline+1, line+strlen(line) - ptrline);
                ltrim (value);
                switch (id) {
                case SERVER:
                    strcpy(var_sys.server, value);
                    break;
                case PIPENAME :
                    strcpy (var_sys.name_pipe, value);
                    break;

               } // endswitch
    } else
                printf ("Identificador en configuracion Desconocido,%s\n",label);
   } else
                printf ("Identificador no tiene valor, %s\n",line);
  } // endwhile

  fclose (fp);
  return 0;
} // fin de Init_Aplic()


//-------------------------------------------------------------------
char *strnzcpy(char *dest,char *src,unsigned int len)
{
  strncpy(dest,src,len);
  *(dest+len)='\0';
  return dest;
}
```

```
// ---------------------------------------------------------------------------
// FUNCION:   copystr()
// OBJETIVO:  Copia n caracteres a T desde S[PI];
//            si BLANCOS !=0 rellenar con espacios en blanco hasta completar N caract.
//            si FLAG != 0 poner marca de fin de cadena '\0' a T
// ---------------------------------------------------------------------------

void copystr(char *s, char *t, int pi, int n, int blancos, int fin)
{
  int  i;

  for (i =0; s[pi] != '\0' &&  i < n; i++, pi++)
      t[i] = s[pi];
  for ( ;i<n && blancos; i++)
      t[i] = ' ';
  if (fin)
      t[i]= '\0';

} // fin de copystr


/*---------------------------------------------------------------------------*/
char *ltrim(char *string)
{
  char *ptr_string, *ptr_move, *edit;

  edit=(char*)malloc(strlen(string)+1);
  for (ptr_string=string;*ptr_string==' ';ptr_string++);
  for (ptr_move=edit;*ptr_string;*ptr_move=*ptr_string,ptr_string++,ptr_move++);
  *(ptr_move)='\0';
  strcpy(string,edit);
  free(edit);

return string;
}

// ---------------------------------------------------------------------

void Analiza_Error (int error )
{

  switch (error)
  {
    case 0:
        printf("\n Estacion: Ocurrio un error en el Servidor.");
        printf("\n No se completo requerimiento.");
        getch();
        break;
    case 2:
        printf("\nEstacion: ERROR Datos no v lidos");
        printf("\nNo se pudo procesar transacci¢n.");
        getch();
```

```
    break;
case 3:
    printf("\nEstacion: ERROR_PATH_NOT_FOUND\n");
    getch();
    break;
case 4:
    printf("\nEstacion: ERROR Muchos Archivos Abiertos");
    getch();
    break;
case 5:
    printf("\nEstacion: ERROR_ACCESS_DENIED\n");
    getch();
    break;
case 8:

    printf("\nEstacion: ERROR_NO_HAY_SUFICIENTE_MEMORIA");
    getch();
    break;
case 11:
    printf("\nERROR MAL FORMATO");
    getch();
    break;
case 12:
    printf("\nERROR ACCESO INVALIDO");
    getch();
    break;
case 84:
    printf("\nERROR_OUT_OF_STRUCTURES");
    getch();
    break;
case 87:
    printf("\nERROR_INVALID_PARAMETER");
    getch();
    break;
case 95:
    printf("\nERROR_INTERRUPT");
    getch();
    break;
case 109:
    printf("\nERROR_BROKEN_PIPE");
    getch();
    break;
case 164:
    printf("\nERROR_MAX_THRDS_REACHED");
    getch();
    break;
case 230:
    printf("\nERROR_BAD_PIPE");
    break;
case 231:
    printf("\nERROR_PIPE_BUSY");
    break;
case 233:
    printf("\nERROR PIPE NO ESTA CONECTADO");
```

```
        break;
    //case 234:ÿ
    //    printf("\nERROR_MORE_DATA");
    //    break;
    case NO_HAY_FONDOS:
            printf("\nLa cuenta no tiene fondos para pagar su cheque\n");
                break;
    case CLIENTE_NO_EXISTE:
                printf("\nEl numero de su cuenta no esta registrado\n");
                printf("\nVuelva a ingresar el numero de su cuenta\n");
        getch();
                break;
    case NO_MOVIMS:
                printf("\nNo hay movimientos en su cuenta\n");
                break;
    case NO_INSERTO_REG:
                printf("\nNo se inserto el registro\n");
                break;
    case TRAN_ERR:
                printf("\nHubo un error en su transaccion\n");
                break;
    case NO_ACTUALIZO_REG:
         printf ("\nNo se actualizo el registro de saldos");
         break;
    case TRAN_FINAL:
        printf("\n Su Transaccion ha sido realizada.  Gracias.");
        getch();
        break;

    default:
          printf ("\n Presione cualquier tecla para volver al menu principal\n");
          getch();
          exit (1);
          break;
   }

} // End Analiza_Error
```

MODULO LLAMADO POR ESTAC001.C

```
//*************************************************************************
// ESTAC001.H
// Header file para ESTAC001.C
// Contiene  Prototipos de las Funciones para Invocar a las Transacciones de
// Consulta de Saldos, Depositos y Pago de Cheques
// *************************************************************************

#define INCL_BASE
#define INCL_DOSNMPIPES
#define INCL_DOS
#define INCL_DOSSIGNALS
#define INCL_DOSERRORS

#include <stdio.h>
#include <malloc.h>
#include <stdlib.h>
#include <string.h>
#include <process.h>
#include <os2.h>
#include <string.h>
#include <memory.h>
#include <conio.h>
#include <bsedos.h>

#include <stddef.h>
#include <fcntl.h>
#include <io.h>
#include <sys\types.h>
#include <sys\stat.h>
#include <dos.h>
#include "stru_dat.h"

#define  CONFIG

#define  OR        ||
#define  AND       &&
#define  NORMAL     0
#define  OPEN      0x1
#define  OPEN_MODE      0x4012
#define  PATH_LENGTH    61
#define  ERR_MSG_SIZE   81
#define  INPUT_SIZE    512
#define  RESULT_SIZE   512
#define  MAX_BUFF_IN   3000
#define  BUFF_SALDO    132+(20*37)
#define  TIMEOUT     10000
#define  MAXLINE          80

#define CABECERA          131
#define LEN_CTA           8
#define LEN_REG_DAT       77
#define LEN_REG_DAT_SAL   129
```

```
#define LEN_MOV          37

#define  LEN_COD_TRAN     5
#define  BASE_CAPTURA    '0'
#define  BASE_CONSULTA    '1'
#define  COPIA_NORMAL    '0'
#define  COPIA_FORZADA    '1'
#define  FIRST          'F'
#define  NEXT           'N'

// Codigos Retornados por el Servidor

#define  CLIENTE_NO_EXISTE     1011
#define  TRAN_ERR            1021
#define  NO_HAY_FONDOS        3010
#define  TRAN_FINAL          5010
#define  NO_ACTUALIZO_REG      4020

#define  TRAN_NO_EXISTE       2011
#define  NO_INSERTO_REG       4010
#define  NO_MOVIMS           2020

#define  TRAN_OK_FIN        22
#define  DATOS_NO_VALIDOS    3
#define  OK              1
#define  MAS_DATOS         -1
#define  FIN_DE_DATOS       -2
#define  ERROR_EN_FUNCION   -7
#define  NO_HAY_DATOS       -8
#define  TRAN_PENDIENTE     -9



#define  ERROR_CONEXION_INI    99

#define  CONSULTA_SALDO        "01000"
#define  ACTUALIZA_CTA        "02000"  // Para Depositos y Pago de Cheques

#define  CHEQ   '1'
#define  DEPO   '2'


#define  TRANSMISION_HOST       "03010"
#define  RETRANSMISION_HOST       "03012"

#define  REQ_INICIAL           "99999"
#define  PIPE_DISPONIBLE        "99991"
#define  PIPE_NO_DISPONIBLE       "99990"


//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//                DEFINICION DE ESTRUCTURAS
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
#ifdef CONFIG
char *labels_id[]={
  "SERVER",
          "PIPENAME"
   };

enum labels{
          SERVER,
          PIPENAME
  };
#endif

typedef struct {
 char server[25];
          char name_pipe[25];
} struct_var_sys;


// ********************************************************************
// Prototipos de funciones utilizadas en ESTACION.C
// ********************************************************************

int Init_Aplic  (char *);
int Consulta_Saldos(void);      // "01000"
int Pago_Cheques(void);         // "02000"
int Depositos(void);           // "03000"

int Pide_Opcion (void);
int Abrir_Sesion(char *,USHORT *);
int Conexion_Inicial(char *);
int Envia_Requerimiento(char *, char *,int,int *, USHORT);
void Cerrar_Sesion(USHORT );
int  Check_Status(char *, char *);
void Mostrar_Datos_Clie (int);
char *strnzcpy (char *, char *, unsigned int);
void copystr(char *s, char *t, int pi, int n, int blancos, int fin);
char *ltrim(char *string);
void Analiza_Error( int);
```

INDICE DE FIGURAS

Pag.