UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

UNIDAD ACADEMICA DE LOS CICLOS PROFESIONAL Y DE POSGRADO
DEL COLEGIO DE CIENCIAS Y HUMANIDADES
PROYECTO MAESTRIA EN CIENCIAS DE LA COMPUTACION

"OPTIMIZACION DEL CANAL DE TRANSMISION DE UNA RED DE AREA LOCAL"

T E S I S

PARA OBTENER EL GRADO DE

MAESTRIA EN CIENCIAS DE LA COMPUTACION

P R E S E N T A:

ALBERTO PAZ GUTIERREZ

NOVIEMBRE, 1995

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

Dedicatoria

Para mi querida esposa Lupita, quién cedió más tiempo del que disponía para permitir realizar este trabajo, y que con su comprensión y cariño soportó hasta su culminación.

Para mi hija Karlita, símbolo de la esperanza que la vida nos dá. Sea esto un pequeño ejemplo

Al Dr. Angel Kuri Morales, por su dirección y consejos en la realización de este trabajo.

A todas las personas que han impulsado este trabajo animándome en las etapas difíciles.

Alberto Paz Gutiérrez
Otoño, 1995

"Optimización del Canal de Transmisión de una Red de Area local"

Resumen

Actualmente existe la tendencia a interconectar computadoras tanto para obtener información como para ofrecerla a quién la solicite. Los medios de comunicación han variado hasta tener transmisiones a través de modem hacia redes de área local confinadas conceptualmente a distancias cortas; así, hoy se establecen redes de redes propietarias y/o de diferentes organizaciones

Este documento describe la arquitectura y los componentes de la biblioteca Dynamic Data Exchange Management Library (DDEML); también muestra como establecer la conexión entre dos o más computadoras que participan de una red de área local en un dominio en el ambiente de Windows NT. El programa Servidor (programa que corre en Windows NT) ofrece la transmisión de archivos a través de la red comprimiéndolos antes de ser enviados hacia el (los) Cliente(s) (programas que están corriendo en Windows para Trabajo en Grupo) que está (estén) activo(s) en una (o varias) computadora(s). Se utiliza la interfaz de programación Network DDEML (NDDEML) para aplicaciones en 32 bits por lo cual podrá usarse en Windows para Trabajo en Grupo, Windows NT, y Windows 95. Esta interfaz está orientada a funciones basándose en mensajes DDE (Dynamic Data Exchange). La biblioteca DDEML registra la aplicación Servidor y Cliente, si existe afinidad entre los servicios que se ofrecen y solicitan, entonces se establece el enlace entre las partes; dependiendo del tipo de enlace, cuando existe un cambio en los datos del Servidor, la biblioteca informa al Cliente y envía el dato. Si el Cliente solicita el dato sólo lo recibirá si existe un Servidor que haya registrado y ofrecido el tipo de dato requerido por el Cliente. Se utiliza el algoritmo Lempel-Ziv-Huffman para comprimir/descomprimir el archivo a transmitir/recibir, para lo cual se utilizó una rutina shareware para Windows para 16 bits. Se utilizó el compilador de Microsoft (R) 32-bit C/C++ Optimizing Compiler Version 9.00 para 80x86.

Se presenta el protocolo de transmisión DDEML para establecer una conexión entre nodos.
Normalmente éste tipo de comunicación entre procesos es local. Sin embargo, cuando se usa NDDEML (Network DDEML) pueden establecerse comunicaciones remotas o locales potenciando su uso para equipos en red. Como ejemplo de lo que puede hacer DDEML, considérese la interfaz Shell DDE. Cuando un programa Windows se autoinstala, solicita al usuario la confirmación para crear un nuevo grupo para el programa en el Program Manager. Si se acepta, el programa de instalación conversa con el Program Manager, a través de la biblioteca DDEML, y pasa la información para crear el nuevo grupo.

1.- Introducción:

Hoy en día es común encontrar computadoras en una empresa formando una red de área local, que transmiten entre si información, reemplazando la vieja usanza de intercambiar información entre computadoras mediante diskettes en forma de respaldos. Esta práctica de compartir información es una función del sistema operativo de red, de tal manera que se han ofrecido éste tipo de funciones en diferentes formas (o protocolos), de las cuales una de las más estandarizadas en redes de área local ha sido *Netbios*[1], desarrollado desde 1984 para ambientes *DOS*, prácticamente desde que aparecieron las computadoras personales en el mercado de cómputo.

Las reglas para la transmisión de información de datos en red dependen del sistema operativo que se utiliza y de los protocolos que se implemente en él, por ejemplo: *Netbios*[2] IPX/SPX, Sockets[3], WNet, NDDEML, Named Pipes[4], etcétera. Sin embargo la tendencia en el mercado de software es la integración de sistemas heterogéneos en velocidad, protocolos, y medios de comunicación lo que hace muy compleja la selección de un protocolo de comunicación estándar para acceder a ambientes de comunicación internacional. Uno de los protocolos de red más aceptado mundialmente es TCP/IP (Transport Control Protocol / Internet Protocol), debido a que éste proporciona la base de la red de redes denominada Internet sobre la cual es posible acceder a cualquier parte del mundo y mantener una comunicación en línea o correo electrónico con cualquier usuario que esté conectado a esta red. La popularidad de Internet ha ido en aumento en los medios académicos y comerciales, de tal forma que actualmente es una herramienta indispensable para acceder a la información más actualizada sobre cualquier campo de la ciencia.

Otro estándar en la industria de las computadoras personales ha sido el sistema operativo *MS-DOS* y el ambiente de trabajo Windows[5] 3.x debido a que mediante presentaciones gráficas permite administrar los recursos de hardware y software de usuarios no especializados en el manejo de una computadora y que la utilizan con propósitos específicos como procesadores de palabras, hojas de cálculo, manejadores de bases de datos, presentaciones por computadora, así como programas de aplicación específica para una organización, tales como manejo de inventarios, cálculo de nóminas, control de la contabilidad, cuentas por cobrar, etcétera. La compañía Microsoft Corp. desarrolló Windows 3.1 desde 1989, y actualmente está desarrollando versiones actualizadas de éste sistema

---

[1] [Naughtobl 88]. Netbios: Network Basic Input Output System, desarrollado por Sytek Inc. para IBM en 1984.
[2] [Baker 93]
[3] [Allard 93]
[4] [Asche 93]
[5] Windows 3.X se considera como un sistema operativo monousuario, debido a que proporciona todos los elementos requeridos de un sistema operativo multiusuario

operativo tales como *Windows para Trabajo en Grupos* (WFW), para agrupar computadoras en red en pequeños grupos de trabajo (alrededor de 10 equipos), *Windows NT* para una organización con 20 o más equipos en red agrupados en dominios, *Windows NT Advanced Server* para controlar grupos de dominios.

El 24 de agosto de éste año Microsoft liberó un sistema operativo personal (*Windows 95*) que no depende de *MS-DOS* para instalarse, es multitarea y con capacidad para establecer comunicación con cualquier otro sistema que tenga los protocolos NDDE, *Netbios*, o TCP/IP. Debido a la gran popularidad de *Windows 3.1* se estima que *Windows 95* se establezca como estándar durante esta década. El software de aplicación actual basado en Windows se caracteriza por ocupar un espacio considerable tanto en memoria principal como en memoria secundaria. Esto es consecuencia de poder usar tecnologías de video de alta resolución (VGA, SVGA), procesadores más rápidos (386, 486 de Intel), tecnologías de almacenamiento más rápido (SCSI, IDE, etc.), ambientes gráficos, y que son preferidos por cualquier usuario u organización debido a que facilita la administración y comprensión de las herramientas de trabajo que se utilizan normalmente.

El panorama anterior ofrece un escenario en redes de área local para desarrollar trabajo bajo la premisa siguiente: debido a que en el medio nacional, la tecnología señalada es utilizada por la mayoría de las organizaciones que poseen una red de computadoras, y que la aceptación y demanda de éste tipo de productos irá en aumento en los próximos años, es deseable optimizar el canal de transmisión que utilizan estas redes; si al efectuar una transmisión de datos (imagen, texto, binarios, etcétera), antes de ser transmitidos se comprimen por una rutina de propósito general, se ocupará, por ende, por menos tiempo (equivalente al porcentaje de compresión del archivo) el medio de transmisión.

De esta forma podrá optimizarse el canal, independientemente de la velocidad del mismo. El costo que se paga por optimizar el canal de transmisión corresponderá al aumento en el tiempo utilizado para efectuar la compresión del archivo a transmitir. En ciertos casos es posible que no exista entropía[6] y se consuma más tiempo para efectuar el análisis del archivo, retrasando considerablemente la respuesta del sistema al usuario que requiere la información; o que el archivo a transmitir pueda ser demasiado pequeño que la rutina de compresión no mejorará significativamente su tamaño final, obteniendo solamente un mayor tiempo de respuesta en la transmisión. Otra de las desventajas que pudiera resultar de la práctica de estas rutinas es que el programa que recibe los datos no recupere la información original, debido a excesivo ruido en el canal de transmisión produciéndose errores críticos, y provocándose retransmisiones de la información, congestionando nuevamente al medio de transmisión. La respuesta de la red en general mejorará significativamente si

[6]Entropía: En Teoría de la información se define como una medida de la libertad de elección; representa la incertidumbre promedio para determinar en donde debiera estar un sistema entre muchos. [*Leinver 87*], pp 206.

se utiliza frecuentemente la transmisión de archivos, disminuyendo el número de accesos que cada usuario hace al medio de comunicación.

Para desarrollar esta aplicación se ha decidido usar la interfaz de programación para red NDDE[7] para ambientes de Windows de 32 bits. Esta aplicación podrá funcionar en WFW 3.x, Windows NT, y el nuevo ambiente *Windows 95*. La restricción de que sea únicamente a 32 bits y no en 16 bits es debido a que la interfaz API[8] *NDDEAPI.DLL*, utilizada para implementar la interfaz NDDE, sólo se consigue para 32 bits, y sólo bajo algunas consideraciones de hardware y medio ambiente, los sistemas de 16 bits podrán acceder al sistema de 32 bits para que puedan usar esta aplicación.

[7] NDDE: Network Dynamic Data Exchange
[8] API: Application Program Interface

2.- Protocolo Network DDE

DDE es un protocolo basado en mensajes para compartir información entre aplicaciones, y es un protocolo estándar de manera que cualquier programa DDE puede comunicarse con otro programa DDE, e inclusive enviarse órdenes entre ellos. Sin embargo, DDE puede ser difícil de administrar puesto que no provee una estructura suficiente para interactuar y un mecanismo de verificación de errores robusto. Network DDE es un protocolo importante de software de la compañia Microsoft para efectuar funciones de red, las cuales se llevan a cabo mediante rutinas de servicio para administrar la comunicación entre procesos. WFW y Windows NT proveen una biblioteca de ligado dinámico (DLL) denominada NETDDE.EXE para administrar las conversaciones de la red mediante el concepto DDE.

Para un programa local, Network DDE emula que la comunicación sucede en la máquina local. El servicio Network DDE procesa todas las comunicaciones DDE requeridas por las aplicaciones y lleva a cabo las operaciones de red necesarias para iniciar y mantener la conversaciòn. Pueden usarse las funciones del Network NDDE para administrar una conversaciòn sobre una red entre dos aplicaciones, cliente y servidor, que suministren funciones DDE. Estas incluyen herramientas para establecer y cerrar conexiones Network DDE, mantener seguridad, transar con recursos compartidos, y administrar recursos y sesiones Network DDE. Estas funciones trabajan sólo sobre el protocolo NetBEUI suministrado por WFW y no puede ser usado por otro protocolo, limitando su uso en ambientes heterogéneos. La implantación sobre Windows NT es más flexible y puede correr sobre cualquiera de los tres protocolos suministrados.

La última versión 3.11 de WFW suministra las mismas interfaces API de red, sin embargo los manejadores son ahora de 32 bits. Cuando se opera en modo extendido WFW puede procesar todas las peticiones de red en 32 bits sin tener que conmutar las tareas a manejadores DOS. Windows NT también utiliza manejadores de 32 bits para red, sin embargo lo hace en ambiente multitarea y conmutable. A diferencia de WFW, pueden operarse apilamientos de diferentes protocolos de red sin distinción entre ellos como primario o secundario, Microsoft suministra varios protocolos apilados con Windows NT cliente -NetBEUI, data link control (DLC), TCP/IP, y la implementación de Microsoft de IPX/SPX (NWLink, el servidor de Windows NT también incluye el protocolo de Macintosh AppleTalk).

2.1- Componentes de Network DDE

Los componentes de Network DDE que suministra WFW son: Network DDE application, Network DDE driver y Network DDE API library, ClipBook Viewer y ClipBook Server. Network DDE esta

implementada como una aplicación basada en Windows (NETDDE.EXE) que corre en el ambiente de fondo[9] de la estación de WFW, y es cargada por el manejador de red WFWNET.DRV al iniciarse. Debido a que Network DDE es una aplicación que no es visible al usuario, aquella no aparece en la lista de tareas cuando está corriendo. Para utilizar *Netbios*, NETDDE.EXE usa un manejador de red NDDENB.DLL para comunicarse con estaciones de trabajo sobre un protocolo de transporte dado, por ejemplo con NetBEUI con objeto de establecer una conversación Network DDE. Esto se muestra en la figura 2.1

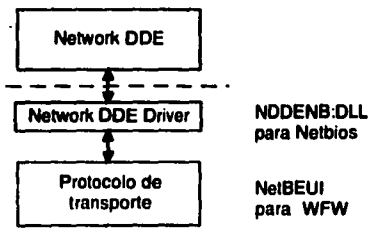

Figura 2.1 Relación entre NetworkDDE y el manejador NetworkDDE y un protocolo de transporte

La biblioteca NDDEAPI.DLL se utiliza para proveer una interfaz de programación para ser usada por desarrolladores de aplicaciones para suministrar la creación y manipulación de recursos compartidos DDE. La biblioteca Network DDE API permite a las aplicaciones que suministran DDE comunicarse directamente entre si.

2.2- Control de acceso a un recurso compartido

Toda comunicación Network DDE entre una aplicación en una estación local y otra en una estación remota con ambientes WFW, es dirigida a través de una aplicación Network DDE mediante un nombre reservado denominado NDDE$[10]. Una vez que la conversación se ha establecido, la aplicación puede requerir información sobre un elemento dado. Con objeto de establecer la conversación Network DDE, primero se establece una conexión entre una aplicación en la estación local y una aplicación identificada por el recurso DDE compartido en la estación remota. La conversación DDE se establece pasando el nombre de la aplicación y el tópico deseado con la cual la aplicación local requiere comunicarse; un ejemplo de la sintaxis utilizada para conectar un recurso compartido DDE se define como a continuación:

---

[9] Cuando un sistema operativo personal ejecuta más de una aplicación concurrentemente, sólo la que está en primer plano está ejecutándose con la mayor prioridad, las demás se ejecutan con menor prioridad, en ambiente de fondo (background)

[10] Para establecer una conversación entre un recurso compartido en una estación WFW remota, se efectua en particular entre una aplicación y un tópico.

aplicación : *NombreComputadora*\NDDE$ tópico: *RecursoCompartido*

en éste ejemplo "*NombreComputadora*" representa el nombre definido de la estación remota WFW y "*RecursoCompartido*" representa el nombre del recurso compartido.

Las siguientes operaciones se llevan a cabo en una comunicación Network DDE:

a) DDE es usado por la aplicación en la estación local para establecer una conversación con NETDDE.EXE en la estación local.
b) NETDDE.EXE en la estación local establece una conexión con NETDDE.EXE en la estación remota usando *Netbios*.
c) NETDDE.EXE en la estación remota verifica si el recurso compartido requerido es válido.

Si el recurso compartido es válido y se suministró la clave (password), si fuera requerida, entonces se establece una conversación Network DDE entre las estaciones local y remota. Si el recurso compartido no existe, o no se suministra la clave correcta, entonces no se efectúa la conversación Network DDE.

Debido a los mecanismos de seguridad construidos en Network DDE, cualquier conversación de éste tipo primero pasa a través de la aplicación de nombre reservado NDDE$ para validar la conversación con un recurso compartido. Una forma diferente de establecer una comunicación es creando una comunicación Network DDE llamando a las funciones apropiadas DDE de la biblioteca NDDEAPI.DLL.

2.3- Interacciones DDE[11]

Aisladas en espacios de direccionamiento privado, las aplicaciones para WFW y Windows NT no pueden compartir datos directamente, sino a través de un protocolo establecido tal como Dynamic Data Exchange (DDE), el visor del portafolio (clipboard), y Object Linked Embebed (OLE).

Cualquier intercambio de información entre dos aplicaciones usando DDE se denomina una *conversación*[12]. Cada conversación está constituida por dos partes. La aplicación que requiere y recibe la información se le denomina el *cliente*, la aplicación que responde a los requerimientos y provee la información se le denomina el *servidor*, un *enlace* (link) en términos de DDE es una serie

[11] [Myers 93] pp 778.
[12] [Myers 93] pp 789.

de mensajes que conllevan una pieza particular de información. Una simple conversación puede establecer varios enlaces de diferentes clases. Un enlace puede ser frío (cold), caliente (hot) o tibio (warm); es el cliente quien determina qué tipo de enlace usar para cada dato requerido.

Todos los enlaces principian cuando un cliente inicia una conversación solicitando un dato, en un *enlace frío* (cold link), el servidor regresa inmediatamente el dato y el intercambio termina; en un *enlace caliente* (hot link), el servidor se mantiene enviando la misma pieza de información cada vez que esta cambie; en un *enlace tibio* (warm link), el servidor no envía el nuevo dato en cada cambio, unicamente notifica al cliente que esto ha ocurrido y el cliente puede optar por ver el nuevo valor del dato.

2.4- Inicio de una conversación

Antes de que un programa intente establecer contacto con otro programa, éste deberá registrarse asimismo con la biblioteca DDEML; la biblioteca mantiene una tabla de todos los participantes DDE. Después de terminar el contacto, la aplicación deberá notificar a la biblioteca de que no participará más en las conversaciones DDE. Registrarse a la biblioteca DDEML se le denomina *iniciarse*, esto se logra llamando a la rutina *DdeInitialize*. En la llamada, deberá decirse a la biblioteca a donde enviar sus mensajes y qué tipo de mensajes no se recibirán. La función pasa un identificador de instancia, un valor que será necesario para identificar al programa cuando se llamen otras funciones DDEML.

```
UINT DdeInitialize(
LPDWORD     lpdwInstID      // dirección del identificador de la instancia
PFNCALLBACK pfnCallback,    // dirección de la función callback DDEML
DWORD       dwFlags         // Ordenes y banderas de filtrado
DWORD       dwRes);         // reservado
```

Después de iniciarse, las aplicaciones del servidor deberá de registrarse los nombres de los servicios que proveen. La biblioteca DDEML envía un mensaje XTYP_REGISTER a otros procedimientos *callback* para avisar del registro de cada nuevo servicio; los clientes deberán de mantener una lista de servicios disponibles DDEML para el usuario. Para registrar los servicios, un programa servidor llama *DdeNameService* con un manejador (handle), a la cadena que nombra al servicio y un identificador de instancia del servicio. De éste último, la biblioteca sabe que procedimiento suministra qué servicio.

```
HDDEDATA DdeNameService(
    DWORD       wInstID,            // identificador de la instancia
    HSZ         hsz1,       // manejador al servicio de nombre-cadena
    HSZ         hszRes,     // reservado
    UINT        uFlags );           // banderas del nombre-servicio
```

Una vez iniciado un programa puede participar en conversaciones que tipicamente siguen tres pasos: conectar, transferir datos, y desconectar. Sólo los programas cliente inician conversaciones. Como dijimos, una conversación es una secuencia de transacciones entre un cliente y un servidor sobre un tópico DDEML. En el caso más simple, un cliente conversa con sólo un servidor. Para iniciar, el cliente llama *DdeConnect*, incluyendo como parámetros dos manejadores que dan nombre al servicio y al tópico. La biblioteca consulta su tabla de servidores registrados y envía un mensaje de transacción XTYP_CONNECT a todos los que suministran el servicio requerido. El servidor recibe el manejador de la cadena de tópico en el parámetro *hsz1*, y el manejador de cadena de servicio en el parámetro *hsz2*. Para tópicos que son suministrados, los servidores regresan TRUE, y para tópicos no suministrados, regresan FALSE, si hay un servidor que regrese TRUE, *DdeConnect* regresa al cliente el manejador de la conversación. La biblioteca DDEML también envía al servidor mensajes de seguimiento, XTYP_CONNECT_CONFIRM, de manera que el servidor recibe también un manejador a la misma conversación. La figura 2.2 muestra la secuencia de las llamadas de funciones y mensajes que el cliente y el servidor inician así mismos estableciendo una conversación.

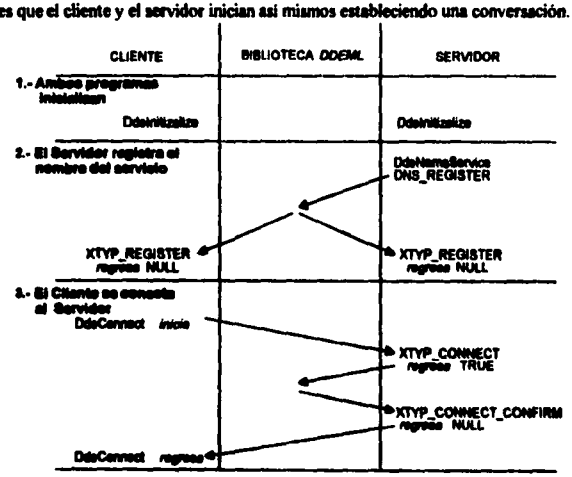

Figura 2.2 . El cliente y servidor se inician y comienzan una conversación DDEML

```
HCONV DdeConnect(
DWORD       dwInstID        // identificador de instancia DDEML
HSZ         hszService,     // manejador de la cadena del nombre del servicio
HSZ         hszTopic        // manejador de la cadena del nombre del tópico
PCONVCONTEXT pCC);          // estructura del contexto de los datos
```

2.5- Transacciones del cliente[13]

Una conversación DDEML consiste de transacciones en las cuales el cliente llama una función y el servidor responde. Hay tres tipos de transacciones: link, poke y execute. Una transacción en la cual el cliente requiere un dato del servidor es llamada una enlace (link), la orden que inicia cualquier tipo de transacción es la siguiente:

```
HDDEDATA DdeClientTransaction (
    LPBYTE      lpbData,        // dirección de los datos a pasar al servidor
    DWORD       dwDataSize,     // longitud de los datos, en bytes
    HCONV       hConv,          // manejador de la conversación
    HSZ         hszItem,        // manejador de la cadena nombre-elemento
    UINT        uFmt,           // formato del dato clipboard
    UNIT        uType,          // tipo de transacción
    DWORD       dwTimwout,          // duración del time-out
    LPDWORD     lpdwResult );   // apunta al resultado de la transacción.
```

Si la transacción termina tan pronto como el cliente requiere un elemento de dato (data item) del servidor, es decir enlace frío (cold link). Sin embargo, no todas las transacciones terminan inmediatamente, el cliente puede requerir la actualización del dato continuamente en cada cambio que haya sobre él. Un enlace tibio (*warm link*) es cuando el servidor sólo notifica que el dato ha cambiado, y se denomina un enlace caliente (*hot link*) cuando para cada cambio del dato, el servidor envía el nuevo dato.

Una transacción en donde el cliente envía un elemento de dato al servidor es denominado Poke; y si el cliente envía una orden al servidor, se denomina una transacción Execute.

2.6- Respuestas del servidor a las transacciones

Cada llamada a *DdeClientTransaction* causa que la biblioteca DDEML envíe al servidor un mensaje. Éste difiere de acuerdo al tipo de transacción que el cliente requiere. La función *Callback* del servidor trata a cada mensaje en un *case* separado.

[13] [Myers 93] pp 708

Mensaje	Transacción
XTYP_REQUEST	Cold Link : envía un elemento
XTYP_ADVISE	Warm or Hot Link : envía un elemento y lo actualiza
XTYP_POKE	Poke : recibe un dato
XTYP_EXECUTE	Execute : ejecuta una acción

Cuando el servidor recibe un mensaje XTYP_REQUEST, éste lee el nombre del tópico y el elemento dentro de los parámetros y verifica los datos requeridos. Si los reconoce como servicios que puede suministrar, crea un objeto para el valor actual del elemento solicitado y regresa el manejador HDDEDATA. Si no suministra el servicio, regresa NULL. La figura 2.3 muestra la interacción.

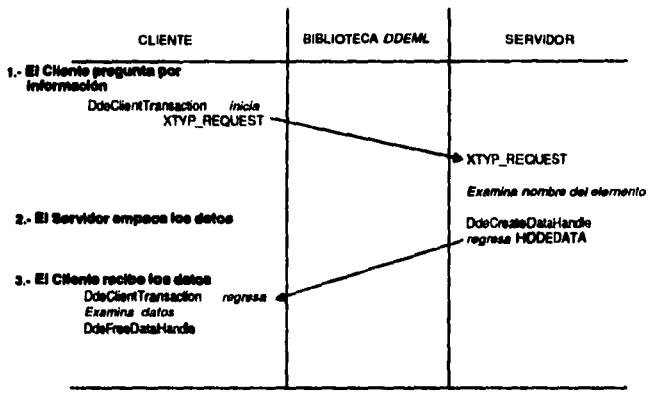

Figura 2. 3 Secuencia de eventos cuando un cliente requiere un dato al servidor

Cuando el cliente requiere actualizaciones en el enlace, el procedimiento del servidor *Callback* recibe el mensaje ADV_START. Si reconoce los nombres del tópico y el elemento, regresa TRUE para permitir el enlace o FALSE para prevenirlo. El servidor no regresa el dato sino hasta que ocurra un cambio. Cuando el dato cambia el servidor llama *DdePostAdvise* para notificar a la biblioteca DDEML que tiene un dato renovado para el cliente interesado. Ver figura 2.4.

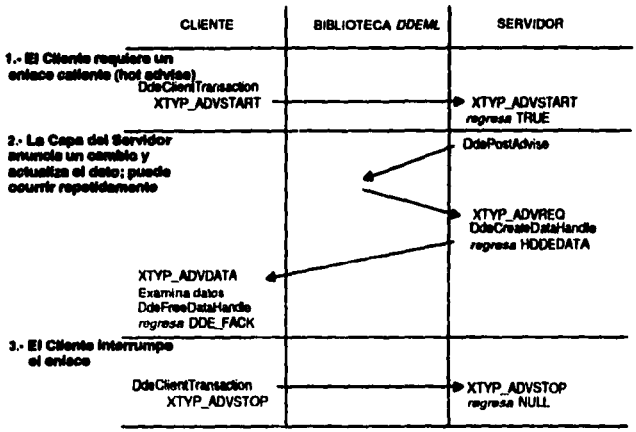

Figura 2.4. Inicio, parte media y final de un enlace "caliente" (hot loop)

2.7- Fin de una conversación y del servicio DDEML

Para terminar una conversación, ya sea el cliente o el servidor llaman a *DdeDisconnect* especificando el manejador de la conversación a terminar. Esto causa que la biblioteca DDEML envíe una transacción XTYP_DISCONNECT a la otra aplicación

```
BOOL DdeDisconnect (HCONV hConv);  // manejador de la conversación
```

Cuando la biblioteca DDEML recibe esta orden, abandona cualquier transacción que estuviera en progreso en una conversación dada. Por convención sólo los clientes interrumpen las conversaciones, aunque a veces deberán hacerlo los servidores -si, por ejemplo, el usuario cierra la aplicación del servidor.

Cuando una aplicación se cierra, o cuando ya no se requiere de mayor conversación con otras aplicaciones, aquella deberá retirar todos sus procedimientos *Callback* DDEML. Llamando *DdeUninitialize* remueve un servicio de la tabla de DDEML. También termina cualquier conversación aún en progreso y envía un XTYP_UNREGISTER a todos los procedimientos *Callback* DDEML. La figura 2.5 muestra los pasos para interrumpir una conversación y retirarse de la biblioteca DDEML.

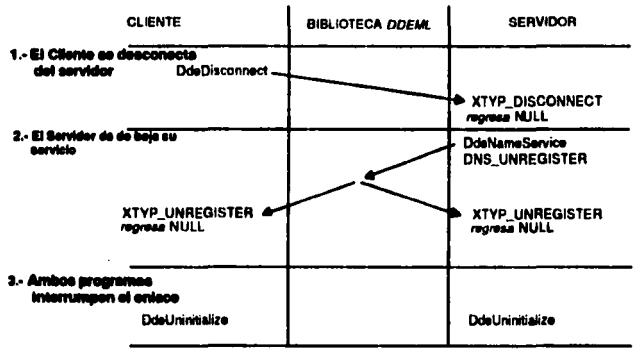

Figura 2.5. Etapas para dar de baja al cliente y al servidor

3.- Programa para transmisión de archivos

Para probar la hipótesis de optimizar el ancho de banda del canal de transmisión, se han programado dos rutinas o programas para efectuar la transmisión de archivos: el programa *cliente* y el programa *servidor*, los cuales, de manera complementaria, estarán conversando entre ellos cuando se activen en dos equipos en red.

El programa *Servidor*, que corre en Windows NT, es el que transmite el archivo, el cual es elegido de cualquiera de las unidades lógicas a las que tiene acceso, posteriormente es comprimido antes de efectuarse la transmisión.

El programa *Cliente*, el cual podrá correr local o remotamente en Windows NT, WFW o *Windows 95*, recibe el dato transmitido por el *Servidor* a través de la biblioteca DDEML, lo descomprime y lo almacena en las unidades lógicas a las que tiene acceso; pueden activarse más de una instancia, o ventana, de aplicaciones *Cliente* y recibirán simultáneamente la indicación de que se ha recibido un dato de la biblioteca DDEML.

3.1- Programa *Cliente*

El programa *Cliente* se activa haciendo doble clic en el Administrador de Archivos, se presenta con un Menú para registrarse a la biblioteca DDEML, preguntando mediante una caja de diálogo a que computadora deberá conectarse. Si existe una aplicación local o remota que ofrezca el servicio (Primero), queda establecido el enlace entre el *Cliente* y su *Servidor* esperando a que el *Servidor* transmita un dato.

3.1.1- Metodología de conexión

Para iniciarse ante DDEML se solicita el nombre del nodo (computadora) con el cual se establecerá la conversación, además de la dirección de la función callback (DdeCallback) especificando los filtros para la función callback :APPCLASS_STANDARD o APPCMD_CLIENTONLY

DdeInitialize (&idInst, &DdeCallback, APPCLASS_STANDARD | APPCMD_CLIENTONLY, 0L)

Una vez iniciada la aplicación *Cliente*, se conecta al *Servidor* estableciendo una conversación; para establecer la conversación NDDE con la aplicación *Servidor*, se utiliza el nombre reservado "Server$"; en la iniciación del *Cliente* con la biblioteca DDEML; se crean las cadenas para el servicio mediante:

```
hszService = DdeCreateStringHandle(idInst, szService,0);
hszShare = DdeCreateStringHandle(idInst, szShare ,0);
```

Una vez realizado se intenta la conexión mediante

```
DdeConnect(idInst, hszService, hszShare, NULL);
```

Finalmente se solicita el dato :

```
hszItem = DdeCreateStringHandle (idInst, szItem,0);
DdeClientTransaction (NULL, 0, hConv, hszItem, CF_TEXT,
        XTYP_ADVSTART | XTYPF_ACKREQ, TIMEOUT_ASYNC, NULL);
```

La obtención del dato (archivo) se hará cuando en la ventana de diálogo aparezcan dígitos en la leyenda de "Bytes Recibidos:", entonces podrá usarse el menú de "Opciones" para almacenar con un nombre y directorio elegidos por el usuario mediante cajas de diálogo para guardar el dato (archivo) recibido.

La aplicación *Cliente* descomprime, al dar un clic en el botón de "Aceptar", la cadena recibida para recuperar el archivo original comprimido por el *Servidor*. Para recuperar el archivo, la aplicación *Cliente* obtiene de la biblioteca DDEML el número de bytes de los datos transmitidos llamando a la función :

```
sizefile =  DdeGetData(hData, NULL, 0,0);
```

Después crea el espacio en memoria de acuerdo a *sizefile* e invocando nuevamente a la misma función, se dá como parámetro la dirección en memoria en donde se alojará el dato, y regresa el apuntador de éste.

```
dwResult =  DdeGetData(hData, pFile, sizeFile, 0);
```

En el desarrollo de las rutinas de transmisión entre los equipos en comunicación, se recibían más bytes de los que se transmitían por el *Servidor*, generando problemas al descomprimir el dato recibido, por lo cual, para corregir esta deficiencia, se utilizó un encabezado al inicio del dato que informa al *Cliente* del número de bytes que constituyen el archivo comprimido; de esta forma se recortó la cantidad de datos recibidos al número indicado por el encabezado, después se procedió a su descompresión.

```
dwSize = (UCHAR)pFile[0];
dwSize <<=8;
dwSize |= (UCHAR)pFile[1];
dwSize <<=8;
dwSize |= (UCHAR)pFile[2];
dwSize <<=8;
dwSize |= (UCHAR)pFile[3];
```

Para descomprimir el archivo se usa una rutina shareware con el algoritmo de Lempel-Ziv-Huffman[14] implementada por Haruyasu Yoshizaki (1988/11/20) y comentada por Haruhiko Okumura (1989/04/07), modificada por Paul Edwards (1990/03/23), y Mark Nelson (1990/06/02) para Microsoft (R). En éste trabajo se usaron las rutinas para compresión y descompresión de buffer a buffer debido a que ya se tiene en memoria el archivo seleccionado para transmitir, efectuándose de acuerdo a la función:

```
GetFileName(hWnd,
            FALSE,                  // para archivar
            szPath,                 // Path del archivo
            szName);                // nombre del archivo
rc = ExpandBufferToBuffer( BufferA+4, sizefile, BufferB, dwSize, &ulTextSize);
```

En caso de que el *Servidor* se desconecte, aparecerá un mensaje de que así ocurrió previendo al usuario de continuar en una espera inútil.

3.2 Programa *Servidor*

El programa *Servidor* se activa dando doble clic en el Administrador de Archivos, se presenta con un menú para elegir de cualquiera de las unidades lógicas, mediante una caja de diálogo, el archivo que posteriormente será transmitido a través de otra opción del menú. La conversación con la(s) aplicación(ones) que recibe el dato (archivo) se lleva a cabo mediante la biblioteca DDEML y el nombre reservado "Server$"; cuando se activa la aplicación, se registra el nombre del servicio que se ofrece, entonces es difundido por la biblioteca DDEML a otras aplicaciones en el sistema, las cuales usan los nombres para conectarse a la aplicación *Servidor*. La biblioteca DDEML asegura la compatibilidad entre aplicaciones DDE, requiriéndoles implementar el protocolo DDE de una forma consistente.

[14] [Lehwer 87]

3.2.1 Servicios

Para iniciarse se registra el nombre del servicio que se ofrece, entonces es difundido por la biblioteca DDEML a otras aplicaciones en el sistema, las cuales usan los nombres registrados para conectarse a la aplicación *Servidor*

```
ShareSetup(szShare, "", "", "", "", szAppName, szTopic);
```

Una vez iniciado espera conectarse a alguna aplicación cuando reciba XTYP_CONNECT de la biblioteca DDEML, validando el servicio para la conexión antes de establecerla, a través del manejador

```
DdeQueryString(idInst, hsz2, szBuffer, sizeof(szBuffer)
if ( 0 != strcmp ( szBuffer, szAppName)) return FALSE;
DdeQueryString(idInst, hsz1, szBuffer, sizeof(szBuffer)
if ( 0 != strcmp ( szBuffer, szTopic)) return FALSE;
```

Efectuada la conexión, se inicia la conversación y cada vez que haya un cambio en los datos, se activa la bandera XTYP_REQUEST o XTYP_ADVREQ que harán el envío de los datos (archivo) hacia el *Cliente*.

```
return DdeCreateDataHandle (idInst, pFile, sizefile, 0, hsz2, CF_TEXT, 0);
```

Para seleccionar el archivo que se transmitirá, el usuario activa una ventana de diálogo del menú de opciones haciendo clic en "Archivo"; una vez elegido, se comprime para ser depositado en memoria principal mediante la rutina :

```
CompressBufferToBuffer(ptrFile, dwSize, BufferA, dwSize, &sizefile);
```

en donde "ptrFile" es la dirección del archivo a comprimir de "dwSize" bytes, y una vez comprimida se depositará en "BufferA" con el tamaño final de "sizefile" bytes. En caso de no elegirse ningún archivo, después de abrirse la caja de diálogo, se le indica al usuario del hecho mediante una caja de mensaje.

Para dar a conocer a la aplicación *Cliente* el tamaño final del archivo comprimido, se incluyó como encabezado del dato a transmitir el número de bytes que corresponden al tamaño del archivo comprimido; esto último se agregó para indicar a la aplicación *Cliente* del número exacto de bytes del archivo comprimido, ya que en la transmisión a través de la red se recibía un excedente en bytes.

```
*((PCHAR)pFile) =(UCHAR)((sizefile & 0xff000000L)>>24);
*((PCHAR)pFile+1)=(UCHAR)((sizefile & 0xff0000L)>>16);
*((PCHAR)pFile+2)=(UCHAR)((sizefile & 0xff00)>>8);
*((PCHAR)pFile+3)=(UCHAR)(sizefile & 0xff);
```

Para terminar la aplicación, se dan dos clics en la esquina superior izquierda, o se usa la opción "Salir" del menú.

3.3 Pruebas de Rendimiento

La eficiencia de utilización del enlace es la razón del tiempo desde que se inicia la transmisión del primer bit de una trama o dato al tiempo en que se recibe el último bit de acuse de recibo de esa trama (Idle RQ: protocolo RQ ocioso). Suponiendo tiempos de procesamiento de la trama y del acuse de recibo muy pequeños respecto a los tiempos de transmisión y propagación de una trama, (ver figura 3.1), la ecuación que se obtiene es :

$$U = \frac{T}{T_t} = \frac{\text{tiempo para transmitir una trama o dato } (T_{ix})}{T_{ix} + \text{tiempo para recibir acuse de recibo}}$$

$$U = \frac{T_{ix}}{T_{ix} + 2\ T_p} = \frac{1}{1 + 2\ a}$$

$$a = T_p / T_{ix}$$

En ambiente ruidoso (o con colisiones), un promedio de N_r retransmisiones serán requeridas:

$$U = \frac{T_{ix}}{N_r T_{ix} + 2 N_r\ T_p} = \frac{1}{N_r (1 + 2\ a)}$$

en donde $N_r = 1 / (1 - P_f)$ y

$P_f = 1 - (1 - P)^N = i\ \ N_i P$ si $N_i P >> 1$

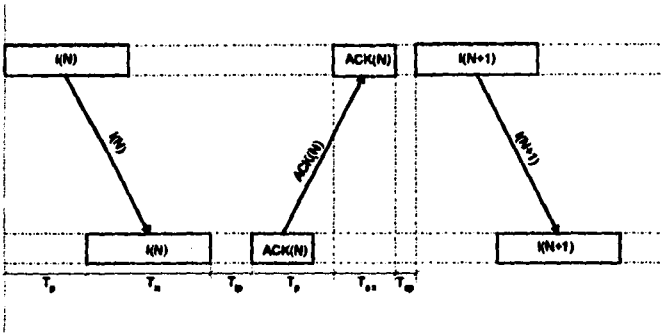

Figura 3.1- Esquema de utilización del enlace ocioso RQ.

para un enlace continuo, la eficiencia de éste está influida por el tamaño K de la ventana

$$U = \frac{K\,T_{ix}}{T_{ix} + 2\,T_p} = \frac{K}{1 + 2a}$$

Si $T_{ix} = T_p$ y K = 1 (RQ ocioso); U = 1/3.

Para alcanzar 100% deberá cumplirse :

K > 1 + 2 a; por tanto K deberá ser mayor que 3

Cualquier error reducirá la eficiencia en la utilización del enlace puesto que habrá que retransmitir las tramas :

$$U = \frac{K}{N_r(1 + 2a)} = \frac{K(1 - P_f)}{1 + 2a} \qquad \text{si } K < 1 + 2a$$

$$U = 1 - P_f \qquad \text{si } K > 1 + 2a$$

Halsall[92] expone la eficiencia de los diferentes métodos de acceso en redes de área local; las pruebas fueron realizadas con segmentos de la misma longitud (2.5 Km) y operando a una misma razón de datos (10 Mbps), el número de estaciones conectadas a cada red fue de 100 nodos en cada caso. Las gráficas de la figura 3.2 muestran el tiempo medio que una trama utiliza para ser transferida

a través de la red en función de la carga ofrecida . La carga está expresada como una fracción de la razón de bits disponible y se refiere al rendimiento normalizado. La gráfica en (a) relacionan tramas de 512 bits de longitud y en (b) de 12,000 bits. Como puede observarse el rendimiento medio es mayor con tramas de mayor tamaño debido a que la sobrecarga en las tramas es menor en éstas últimas.

El tipo de red de área local que se encuentra con mayor frecuencia instalado es CSMA/CD. En general las redes Ethernet trabajan bién provisto que la razón T_p/T_{ix} es una fracción pequeña. Aquí es mucho menor que Tix. El tiempo gastado cuando sucede una colisión (2 Tp en el peor de los casos) es breve comparado con el tiempo de transmisión de una trama. Este tiempo determina el tamaño mínimo de la trama que deberá usarse. Por ejemplo para un bus con T_p= 25μs (2.5Km) y un tamaño de trama media de 10,000 bits obtendríamos

10 Mbps: $T_p/T_{ix} = 25 / 1000 = 1 / 40$
100 Mbps: $T_p/T_{ix} = 25 / 100 = 1 / 4$

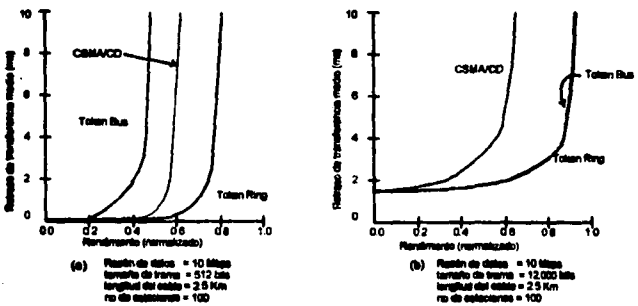

Figura 3.2 Rendimiento de redes de área local

Para probar las aplicaciones *Servidor* y *Cliente*, se usaron las siguientes configuraciones de equipo:

Aplicación "*Servidor*"

Computadora Personal 486, 16 Mb en RAM,
Sistema Operativo Advanced Server Windows NT 3.1
Tarjeta de red Ethernet 10BaseT

Aplicación "*Cliente*"

Computadora Personal 486, 4 Mb en RAM,
Sistema Operativo Windows Trabajo en Grupo 3.1
API Win32, NDDEML:DLL
Tarjeta de red Ethernet 10BaseT

Se activaron en cada equipo las aplicaciones *Cliente* para mostrar la recepción de los datos recibidos que enviaba el *Servidor*, se obeservaron los siguientes resultados:

	Archivo Comprimido		Archivo sin Comprimir	
	Servidor	*Cliente*	*Servidor*	*Cliente*
# Bytes TX	407	412	791	796
# Bytes TX	1919	1948	6301	6332
# Bytes TX	6601	6620	20536	20540

4.- Conclusiones

El desarrollo de rutinas para transmisión de datos ha sido recurrente en los diversos formatos en que se han presentado, ya sea por el puerto serie o paralelo, o por las diferentes tecnologías de los sistemas operativos de red. La interfaz NDDEML sigue un modelo de transmisión basado en el llamado de funciones, en lugar de mensajes, para establecer comunicación con otros procesos. Sin embargo, debido a la dependencia de la interfaz para 32 bits, WIN32, no es fácilmente instalable en todas las posibles arquitecturas de cómputo actuales, es decir, está limitado a trabajar en Windows para Trabajo en Grupo, Windows 95 y Windows NT.

Inclusive, antes de que sea posible usarla en Windows Trabajo en Grupo es indispensable instalar la interfaz WIN32; esta interfaz viene con los kits de desarrollo SDK (Software Development Kit) de Microsoft, o en los compiladores modernos como Visual C++ 2.0 o Borland 4.5.

La optimización que se logra al poder transmitir a través del medio de comunicación una cantidad menor de datos debido a la compresión alcanzada, permitirá desahogar redes de alto tráfico de datos que estén basados en transferencia de información entre ellos.

El desarrollo de este trabajo permitió también utilizar la metodología Cliente/Servidor, la cual es un nuevo paradigma para desarrollos de aplicaciones, en donde se establece una colaboración entre dos aplicaciones que se comunican entre sí para llevar a cabo una tarea; la inscripción de servicios por el *Servidor* así como la solicitud de los mismos por el *Cliente* a la biblioteca DDEML permiten activar más de una aplicación *Cliente* y acceder al servicio, del mismo modo, más de un *Servidor* puede ofrecer el servicio; el sistema operativo decidirá aleatoriamente la elección de uno de los servidores para atender a la petición. En este último caso, se tiene la ventaja de no depender de un solo servidor para la atención de servicios comunes, que serían cuello de botella en caso de fallas del servidor o de la comunicación con él.

Un ejemplo de aplicación práctico para este trabajo es en salones de clase en donde el profesor a través de su computadora envía cambios a los equipos de cómputo de los alumnos en una clase progresiva, o para mostrar en etapas controladas el avance de alguna técnica. Los alumnos no necesariamente tendrían que ubicarse en el mismo salón o campus que el profesor para atender a la clase .

En un banco, puede verificarse la autenticidad de las firmas de la documentación crítica para transacciones bancarias, obteniéndose una imagen de las firmas autorizadas desde los espacios de almacenamiento ubicados para tal efecto y haciendo factible este servicio bancario

La red Internet trabaja actualmente con ambientes gráficos y la velocidad promedio de transferencia, para éste tipo de ambiente, es de 64,000 bps con resultados altamente favorables en la respuesta que tiene un usuario; los ambientes multimedia que empiezan a proliferar harán que en las redes locales se explote fuertemente el medio de comunicación. El programa presentado es aplicable como plataforma de desarrollo para transferencia de datos en ambientes Windows como el descrito anteriormente.

La optimización de los medios de almacenamiento también han considerado la compresión de la información que se almacena en los medios magnéticos; las versiones modernas del sistema operativo DOS, así como utilerías independientes como DoubleSpace, incluyen herramientas residentes para compresión de los archivos que residen en el disco duro

5.- Bibliografía

[Cormack 85]
Cormack, G.V.
"Data Compression on a Database System",
Comm ACM 28, 12 (Dec) 1336-1342., 1985

[Lelewer 87]
Lelewer, Debra A. and Hirschberg, Daniel S.
"Data Compression",
ACM Computing Surveys, Vol 19, No 3, September, 1987, 261-293

[Tannenbaum 91]
Tanenbaum, Andrew S.
Redes de Ordenadores, 2a Ed.
Prentice-Hall Hispanoamericana, S.A., México, 1991

[Halsall 92]
Halsall, Fred
Data Communications, Computer Networks and Open Systems 3rd ed.
Addison Wesley Pub, 1992, USA.

[Baker 93]
Baker, Steven
An Overview of Network Programming Interfaces for Windows and Windows NT
Books and Periodicals Microsoft Systems Journals, 1993, vol 8, Nov 1993, No 11

[Ezzel 93]
Ezzell, Ben
Windows NT Programming
Zeff-Davis Press, USA, 1993.

[Gellock 93]
Gellock, Scot
Developing Distributed Applications with Windows NT
Microsoft Corporation, 1993.

[Groves 93]
Groves, Jim
Windows NT Answer Book
Microsoft Press,USA, 1993.

[Myers 93]
Myers, Brian and Hamer, Eric
Mastering Windows NT Programming
Sybex, USA, 1993.

[Allard 93]
Allard, J.
Advanced Internetworking with TCP/IP on Windows NT
Microsoft Corporation, USA, 1993

[Allard 93]
Allard, J. Moore, Keith and Treadwell, David
Plug into serious Networking Programming with the Windows Sockets API
Microsoft Systems Journal, USA, Vol 8, Jul 1993, No 7.

[Asche 94]
Asche, Ruediger R.
Communication with Class
Microsoft Developer Network Technology Group, July 29, 1994.

[Asche 94]
Asche, Ruediger R.
Garden Hoses at Work
Microsoft Developer Network Technology Group, July 29, 1994.

[King 94]
King, Adrian
Inside Windows 95
Microsoft Press, USA, 1994.

[Knoeller 94]
Knoeller, John M.
Writing Multithreaded Applications
Microsoft Developer Network, Conference and Seminars, Tech Ed, March 1994.

6.- Glosario de Términos

80X86: Linea de procesadores de la Compañía Intel para computadoras personales

API: Application Program Interface: concepto aplicable para usar y/o programar interfaces para el ambiente Windows de Microsoft.

AppleTalk: protocolo de comuniciacion para una red de área local para la computadora personal Macintosh

Cold Link: es un enlace iniciado para requerimiento único de datos entre dos aplicaciones; después de establecer la petición se remueve el enlace entre las aplicaciones.

DDE: Dynamic Data Exchange : protocolo basado en mensajes para intercambio de datos entre aplicaciones en ambientes de Windows 3X.

DDEML: protocolo basado en funciones de biblioteca para intercambio de datos entre aplicaciones de Windows 3X.

DLL: Dynamic Link Library, bibliotecas de propósito general que se cargan al momento de ejecutarse programas que las usen reduciendo el uso de la memoria principal y secundaria.

DOS: Disk Operating System, sistema operativo desarrollado para las computadoras personales

Hot Link: es un enlace entre dos aplicaciones; el servidor se mantiene actualizando los datos a la biblioteca DDEML y ésta envía al cliente el nuevo dato.

Idle RQ (Protocolo RQ ocioso), se establece una comunicación entre dos entidades de manera que haya una comunicación confiable enviando sólo una trama (o paquete) y esperando el acuse de recibo del *nodo receptor*.

NDDEAPI.DLL: Biblioteca de ligado dinámico para NDDE proporcionada para ejecutar aplicaciones de Windows cuando se ejecutan programas que trabajan en red.

NDDEML: protocolo basado en funciones de biblioteca para intercambio de datos entre aplicaciones remotas de Windows 3X, Windows para Trabajo en Grupo, Windows NT, Windows 95, etc.

NDDENB: protocolo Netbios basado en las primitivas de comunicación DDE.

Netbios: es una interfaz de programación de alto nivel introducido con la tarjeta de red IBM PC Network en marzo de 1984, por Sytek Inc.; diseñado para suministrar comunicaciones par a par

OLE: Object Linking and Embedding: conjunto de protocolos y procedimientos propuestos por la Compañía Aldus Corporation para la creación de documentos compuestos, es decir documentos que reciben datos de otras aplicaciones en forma de objetos.

TCP/IP: (Transport Control Protocol/Internet Protocol), es una red apilada encima de la cual pueden construirse una rica familia de aplicaciones de red tales como Telnet para emulación de terminales, FTP para transferencia de archivos, etc.

Warm Link: es un enlace entre dos aplicaciones; el servidor se mantiene actualizando los datos a la biblioteca DDEML; el cliente es prevenido del nuevo dato y puede optar su actualización.

Windows 3.X Ambiente gráfico para la administración de los recursos de una computadora personal. Puede considerarse como un *shell* para el sistema operativo DOS.

Windows 95: Sistema operativo personal y concurrente de Microsoft Inc. Sus principales atributos respecto a sus antecesores son la concurrencia de tareas, manejo de 32 bits, y mejorado desempeño gráfico

Windows for Work Groups (Windows para Grupos de Trabajo), ambiente gráfico equivalente al de Windows 3.X; con capacidad de comunicación en redes de área local para usar y compartir recursos de/con las computadoras pertenecientes al "Grupo de Trabajo".

Windows NT: Sistema operativo para compartir y administrar recursos y perfiles de usuario para grupos de trabajo o dominios.

Windows NT Advanced Server: sistema operativo para administrar dominios y compartir recursos.

WNet: las funciones de llamadas Wnet son especificas a Windows 3.x, hacen llamadas estándar dentro de un manejador de red disponible al programador; suministran acceso a red cuando se utilizara el Administrador de Archivos, Administrador de Impresoras, y otras utilerias del sistema

7.- Manual del Usuario

Instalación

El software desarrollado en este trabajo se presenta mediante dos programas en archivos denominados *Server.exe* y *Client.exe* los cuales establecen una comunicación que permite la transmisión de archivos comprimidos desde el programa *Server.exe* hacia el programa *Client.exe*

Para instalar el programa *Server.exe*, deberá hacerse en un servidor que cuente con el sistema operativo Windows NT 3.1, o Windows NT 3.5 Workstation. Es conveniente que la aplicación se deje en un directorio de red con permisos de sólo lectura para protegerlo. Es suficiente copiar el archivo *Server.exe* a ese directorio para que sea activado por los usuarios que lo requieran. En caso de optarse por asignarlo a un icono, este es actualmente el icono por omisión de Windows con lo cual es posible activarlo más fácilmente.

La instalación del programa *Cliente.exe*, es indispensable hacerla en un equipo que cuente por lo menos con el sistema operativo Windows para Trabajo en Grupo con la interfaz Win32.DLL la cual viene en el programa Win32s. Este último lo proveen los compiladores modernos del lenguaje C, tal como Visual C++ de Microsoft o Borland C/C++ de Borland, así como es posible obtenerlo en el shareware de Microsoft en Compuserve, América on Line o Internet. Microsoft ha asegurado que las aplicaciones de 32 bits que corren en Windows 3.1 o Windows Trabajo en Grupo, también lo harán en Windows 95.

Uso de la aplicación *Server*

Es suficiente dar dos clics al icono asociado al programa *Server* para que este se active; si la aplicación no se instaló asociada a un icono, se localiza mediante el *Administrador de Archivos* y deberán darse dos clics. En caso de que la aplicación no inicie por el error:

"No se puede enlazar al servicio DSDM"

este se corrige activando los servicios *DDE de RED* y *DSDM de DDE de red* que se encuentran en *Servicios* del *Panel de Control*, lo cual podrá efectuarse sólo con privilegios de Administrador de la red.

Server tiene un menú con las siguientes opciones:

Archivo: Elección del archivo a transmitir.
La elección se efectúa a través de una caja de diálogo, la cual permite elegir el archivo por mascarilla (*.*, *.txt, *.doc, etc), cambiar de directorio o de unidad lógica. Si no se eligió ningún archivo, se le notifica al usuario mediante un mensaje; el nombre del archivo elegido es presentado en la ventana mostrándose el numero de bytes del mismo una vez comprimido. Si existen más archivos por transmitirse, deberán seleccionarse aisladamente cada uno.

Transmitir: Transmite el archivo elegido.
Una vez seleccionado el archivo que se transmitirá, éste se envía dando un clic. Las aplicaciones *Cliente* que se registren a la biblioteca DDEML requiriendo este servicio recibirán el archivo transmitido.

Salir: Termina la aplicación.
En caso de haber aplicaciones *Cliente* registradas por la biblioteca DDEML, son prevenidas de la desconexión del *Server*, mostrándose el mensaje:

" *El servidor se ha desconectado* "

Uso de la aplicación *Client*.

De la misma forma que a la aplicación *Server*, también deberán darse dos clics al ícono asociado al programa *Client* para que este se active; si la aplicación no se instaló asociada a un ícono, se localiza mediante el *Administrador de Archivos* y deberán darse dos clics.

Client tiene un menú con las siguientes opciones:
Conectar: Elección del nodo que ofrece la aplicación *Server* con la transmisión de archivos comprimida. En caso de que no se haya escrito correctamente el nombre del nodo al cual se conectará Client o que este no esté activo aparecerá el mensaje:

" *El servidor se ha desconectado* "

si se establece la conexión a través de la biblioteca DDEML, se inhibe una nueva elección, tornándose esta alternativa del menú más tenue, y deshabilitándola.

Desconectar: La aplicación *Client* se desconecta del servidor
En caso de decidir su desconexión de la biblioteca DDEML podrá realizarse a través de esta alternativa siempre y cuando estuviera conectada la aplicación Cliente; si no estaba conectada previamente, esta alternativa se inhibe presentándose de color mas tenue.

Guardar: Guarda el archivo recibido.
Puede guardar el archivo recibido en el directorio por omisión, o en los subdirectorios de la unidad actual, o cambiar de unidad lógica. si se dá un nombre sin extensión se añade la extensión por omisión " *trs* ". El archivo recibido puede guardarse con diferentes nombres mientras no haya actualizaciones. la única forma de reconocer que se ha recibido un nuevo archivo es observando cambios en la ventana de la aplicación en donde se muestra

"Bytes recibidos: 6619 -> 6615".

el número exacto de bytes efectivos recibidos corresponde al que aparece a la derecha del símbolo "->", (en el ejemplo son 6615). el primer número corresponde al numero de bytes recibidos, los cuales en la transmisión llegan a ser más de los bytes que corresponden al archivo comprimido. Si se activa esta opción sin haber recibido previamente ningún archivo, se producirá un error que hará terminar la aplicación.

Salir: Termina la aplicación.
Finaliza la aplicación

8.- Anexos programas fuente

Server.c
Client.c
Lzhuf.h
Huffman.h
Huffman1.c
Huffman2.c
Huffman3.c
Huffman4.c
Huffman5.c

```
/*****************************************************************************
 Server.c - A server application for transmiting files

 This application creates a NDDE share, Server$, and then sends data to any
 connected client applications requesting data.  The file to send is readed
 in the menu control in the main window.
 Program base routines for

   DDEML from: CD Windows NT ver 3.5 and Win32 SDK
   pre-release; CD ROM Developer Network Development Plataform, Microsoft.

   FILE from: Myers, Brian and Hammer, Erick
   "Mastering Windows NT Programming", Sybex, USA, 1993.

   COMPRESS/EXPAND from    Haruyasu Yoshizaki 1988/11/20, shareware

 Alberto Paz Gutiérrez                  UACPyP del CCH, UNAM
                                  Mayo,Noviembre 1995
*****************************************************************************/
#include <windows.h>
#include <ddeml.h>
#include <string.h>

#include "server.h"
#include "nddeinit.h"
#include "constant.h"

/* DIBFILE */
#include <windowsx.h>
#include <commdlg.h>

#include "dibfile.h"

/* Huffman */
#include "huffman.h"

LPSTR ReadFileToSend(HWND hWnd, LPSTR lpszName, LPSTR lpszPath, BOOL bAsk );
static BOOL DoOpenFileDlg( HWND hWnd, LPSTR lpszName, LPSTR lpszPath );


/*****************************************************************************
* Global Variables
*****************************************************************************/
char  szShare[] = SHARENAME;
char  szAppName[]  = "Servidor" ;
char  szTopic[]    = "Datos";
char  szItem[]     = "Primero";
char  szData[33];  // buffer for DDE data
HDDEDATA hData;
DWORD idInst ;
HWND  hwnd ;
HMENU hmenu ;
HINSTANCE hInst;

/* For file reading */
char szFileName[_MAX_FNAME];
char szPathName[_MAX_PATH];
LPSTR pFile;
char *pMapView;              // pointer to a view
#define PATH_OFFSET 0        // place in buffer where path begins
#define FNAME_OFFSET  MAX_PATH   // place in buffer where name begins
unsigned long sizefile;      // size of the file to send

/* For Huffman compressing */
```

```
HANDLE hBufA;
HANDLE hBufB;
LPSTR BufferA;
LPSTR BufferB;

/*****************************************************************************
* Procedure: WinMain
*
* Purpose: Creates and initializes the main window and the DDE conversation
*****************************************************************************/
int PASCAL WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    LPSTR lpszCmdLine, int nCmdShow)
    {
    FARPROC  pfnDdeCallback ;
    MSG      msg ;
    WNDCLASS wc ;


    // register and create main window
    if (hPrevInstance)
        return FALSE ;

    hInst = hInstance;
    *szData = 0;

    wc.style         = 0 ;
    wc.lpfnWndProc   = ServerWndProc ;
    wc.cbClsExtra    = 0 ;
    wc.cbWndExtra    = 0 ;
    wc.hInstance     = hInstance ;
    wc.hIcon         = LoadIcon (NULL, IDI_APPLICATION) ;
    wc.hCursor       = LoadCursor (NULL, IDC_ARROW) ;
    wc.hbrBackground = GetStockObject (WHITE_BRUSH) ;
    wc.lpszMenuName  = MAKEINTRESOURCE(IDM_SERVER) ;
    wc.lpszClassName = szAppName ;

    RegisterClass (&wc) ;

    hwnd = CreateWindow (szAppName, "ESME Server H -> C",
                         WS_OVERLAPPEDWINDOW,
                         CW_USEDEFAULT, CW_USEDEFAULT,
                         200, 120,
                         NULL, NULL, hInstance, NULL) ;
    hmenu = GetMenu(hwnd) ;
    ShowWindow (hwnd, SW_NORMAL) ;
    UpdateWindow (hwnd) ;

    // setup NDDE share using static app/topic list
    if (!ShareSetup(szShare, ',', ',',szAppName,szTopic))
      return FALSE;

    // initialize DDEML for conversation
    pfnDdeCallback = MakeProcInstance ((FARPROC) DdeCallback, hInstance) ;

    if (DdeInitialize (&idInst, (PFNCALLBACK) pfnDdeCallback,
                       APPCLASS_STANDARD, 0L))
        {
        MessageBox (hwnd, "No puede inicializarse el servidor!",
                    szAppName, MB_ICONEXCLAMATION | MB_OK) ;

        DestroyWindow (hwnd) ;
        return FALSE ;
        }
    SendMessage (hwnd, WM_USER_INITIATE, 0, 0L) ;
```

```
    // process messages until quit
    while (GetMessage (&msg, NULL, 0, 0))
        {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
        }

    // uninitialize DDEML
    DdeUninitialize (idInst) ;

    return msg.wParam ;
    }

/*********************************************************************
* Function: DdeCallback ( UINT, UINT, HCONV, HSZ, HSZ, HDDEDATA,
*                         DWORD, DWORD )
*
* Purpose: Handle DDE messages from DDEML
*********************************************************************/
HDDEDATA CALLBACK DdeCallback (UINT iType, UINT iFmt, HCONV hConv,
                               HSZ hsz1, HSZ hsz2, HDDEDATA hData,
                               DWORD dwData1, DWORD dwData2)
{
    char szBuffer [80] ;

    switch (iType)
        {
        case XTYP_CONNECT:          // hsz1 = topic
                                    // hsz2 = service
            // validate service for connection
            DdeQueryString (idInst, hsz2, szBuffer, sizeof (szBuffer), 0) ;

            if (0 != strcmp (szBuffer, szAppName))
                return FALSE ;

            // validate topic for connection
            DdeQueryString (idInst, hsz1, szBuffer, sizeof (szBuffer), 0) ;

            if (0 != strcmp (szBuffer, szTopic))
                return FALSE ;

            return TRUE ;

        case XTYP_ADVSTART:
            // start advising - send initial data
            PostMessage (hwnd, WM_TRANSMIT, 0, 0L) ;
            return TRUE ;

        case XTYP_REQUEST:
        case XTYP_ADVREQ:
            // send data
            *((PCHAR)pFile)  =(UCHAR)((sizefile & 0xff000000L)>>24);
            *((PCHAR)pFile+1)=(UCHAR)((sizefile & 0xff0000L)>>16);
            *((PCHAR)pFile+2)=(UCHAR)((sizefile & 0xff00)>>8);
            *((PCHAR)pFile+3)=(UCHAR)(sizefile & 0xff);

            return DdeCreateDataHandle (idInst, pFile,
                                sizefile+4,0, hsz2, CF_TEXT, 0);

        case XTYP_ADVSTOP:
            // stop advising client
            return TRUE ;
```

```
    }

  return 0 ;
  }

/*********************************************************************
*  Function: ServerWndProc ( HWND, UINT, UINT, LONG )
*
*  Purpose: handle main window message loop
*********************************************************************/
LRESULT CALLBACK ServerWndProc (HWND hwnd, UINT message, WPARAM wParam,
                                LPARAM lParam)
{
 static HSZ  hszService, hszTopic, hszItem;
 static HWND hwndEditBox;
 HDC         hdc;
 PAINTSTRUCT ps;
 char szBuffer[50];
 switch (message)
   {
   case WM_CREATE:
      // initialization of main window
      // creates an edit control for data entry
      *szData = 0;
// *****************************************************************
      hwndEditBox = CreateWindow (
          "edit",
          NULL,
          WS_CHILD|WS_BORDER|ES_AUTOHSCROLL|ES_LEFT,
          5, 30, 180, 25,
          hwnd,
          (HMENU)IDE_EDITBOX,
          hinst,
          NULL );

      if (!hwndEditBox) {
         PostQuitMessage (0) ;
         return 0 ;
        }
      SendDlgItemMessage ( hwnd, IDE_EDITBOX, EM_LIMITTEXT,32, 0L );
      ShowWindow(hwndEditBox, SW_SHOWNORMAL);
      SetFocus(hwndEditBox);

     return 0;

   case WM_USER_INITIATE:
      // DDEML registration and string handle creation
      hszService = DdeCreateStringHandle (idInst, szAppName, 0) ;
      hszTopic   = DdeCreateStringHandle (idInst, szTopic,   0) ;
      hszItem    = DdeCreateStringHandle (idInst, szItem,    0) ;

      DdeNameService (idInst, hszService, 0, DNS_REGISTER) ;
      return 0;

   case WM_PAINT:
      // paint static text
      hdc = BeginPaint (hwnd, &ps) ;
      wsprintf(szBuffer, "Archivo : %s con %d bytes", szFileName,sizefile);
      TextOut (hdc, 5, 5, szBuffer,strlen(szBuffer));
      EndPaint (hwnd, &ps) ;
      return 0 ;

   case WM_TRANSMIT:
      // tell DDEML to send data in file
```

```
            SendDlgItemMessage ( hwnd, IDE_EDITBOX, WM_GETTEXT,
                                 32, (LPARAM)(LPSTR) szData) ;
            DdePostAdvise (idInst, hszTopic, hszItem) ;
         return 0;

   case WM_COMMAND:
      switch( LOWORD(wParam) ) {

         case IDM_SELECT_FILE :
            pFile = ReadFileToSend( hwnd, // load file
            szFileName,                   // store file name here
            szPathName,                   // store file path here
            TRUE );                       // ask user to choose file
            if (!pFile)
               {
                  MessageBox (hwnd, "no se escogió archivo.",
                  szAppName, MB_ICONASTERISK | MB_OK) ;
                  return -1;              // assume user canceled
               }
            return 0 ;

         case IDM_TRANSMIT :
            PostMessage(hwnd, WM_TRANSMIT, 0, 0L );
            return 0;
         case IDM_EXIT:
            PostMessage( hwnd, WM_CLOSE, 0, 0L );
            return 0 ;
          }
      break;

   case WM_CLOSE:
         PostQuitMessage (0) ;
         return 0 ;

   case WM_DESTROY:
      // clean up DDEML
      DdeNameService (idInst, hszService, 0, DNS_UNREGISTER) ;
      DdeFreeStringHandle (idInst, hszService) ;
      DdeFreeStringHandle (idInst, hszTopic) ;
      DdeFreeStringHandle (idInst, hszItem) ;

      PostQuitMessage (0) ;
      return 0 ;

      }
   return DefWindowProc (hwnd, message, wParam, lParam) ;
}


/*-------------------------------------------
   from Mastering Windows NT Programming
   copyright 1993 by Brian Myers & Eric Hamer

   ReadFileToSend
   Ask the user what file to open.  Load the file from the
   file into memory.  Return a pointer to a valid file (or NULL
   for errors).

   RETURN
   A valid file pointer for success; NULL for failure.  Also,
   if lpszName and lpszPath are not NULL ReadFileToSend passes back
   the  name and path of the file opened.
-------------------------------------------*/
```

```
LPSTR  ReadFileToSend (
   HWND hWnd,
   LPSTR lpszName,                  // name of file
   LPSTR lpszPath,                  // full path of file
   BOOL bAsk)                       // ask user for file name?
{
   BY_HANDLE_FILE_INFORMATION FileInfo;// info about an open file
   HANDLE hFile;                    // handle to the file
   ULONG dwSize;                    // number of bytes in file
   char szName[_MAX_FNAME];         // name of disk file
   char szPath[_MAX_PATH];          // full path of disk file
   DWORD dwBytesRead;               // number of bytes readden from disk
   DWORD dwBytesWritten;            // number of bytes written to disk
   BOOL bTest;                      // for testing function returns
   LPVOID ptrFile;                  // pointer to the loaded file
   short rc;
   ULONG ulTextSize;                // tamaño texto expandido


   /* copy user's strings to full-sized buffers */
   if (lpszName)
   {
      lstrcpy( szName, lpszName );
   }
   if (lpszPath)
   {
      lstrcpy( szPath, lpszPath );
   }

   /* if caller said to confirm name, show dialog */
   if (bAsk)
   {
      bTest = GetFileName( hWnd,     // ask the user for a file name
         TRUE ,                      // TRUE  to open
         szPath,                     // buffer for path name
         szName );                   // buffer for file name

      if (!bTest)
      {
         return( NULL );             // the user pressed Cancel
      }
   }

   /* open the file */
   hFile = CreateFile( szPath,              // path of file
         GENERIC_READ,                      // read-only access
         FILE_SHARE_READ,                   // permit file sharing
         NULL,                              // default security
         OPEN_EXISTING,                     // do not create new file
         FILE_ATTRIBUTE_NORMAL,             // file attributes
         NULL );                            // template for attributes
   if (hFile == INVALID_HANDLE_VALUE)
   {
      MessageBox(hWnd, "No puede abrirse el archivo", "Error", MB_OK);
      return( NULL );
   }

   /* determine the number of data bytes in the file */
   bTest = GetFileInformationByHandle(hFile,&FileInfo);
   if (!bTest)
   {
      CloseHandle(hFile);
      MessageBox(hWnd,"No puede leerse el archivo","Error",MB_OK);
```

```
    return (NULL);
  }
  /* size of file */
  dwSize = (DWORD) FileInfo.nFileSizeLow;
  // Allocate space in memory for an object that size
  pbrFile = GlobalAllocPtr(GHND,dwSize);
  if (!pbrFile)
  {
    CloseHandle(hFile);
    MessageBox(hWnd,"No hay memoria disponible","Error",MB_OK);
    return (NULL);
  }
  // Read the data into the new buffer
  bTest = ReadFile( hFile,                    // file to read in
        pbrFile,                  // buffer to fill
        dwSize,                   // num bytes to read
        &dwBytesRead,             // value returned
        NULL );                   // no overlapping I/O
  if ((!bTest) || (dwBytesRead != dwSize))
  {
    CloseHandle( hFile );
    GlobalFreePtr(pbrFile);
    MessageBox( hWnd, "Bits no fueron leidos", "Error", MB_OK );
    return( NULL );
  }

  /* If the caller gave name pointers, use them to return strings */
  if (pszName)
  {
    lstrcpy( pszName, szName );
  }
  if (pszPath)
  {
    lstrcpy( pszPath, szPath );
  }

/* Comprime archivo a transmitir */
  hBuffA = GlobalAlloc(GMEM_MOVEABLE,dwSize+4);
  if (!hBuffA)
  {
    CloseHandle(hFile);
    MessageBox(hWnd,"No hay memoria disponible para compresion","Error",MB_OK);
    return (NULL);
  }
  BufferA=(LPSTR)GlobalLock(hBuffA);
  rc=CompressBufferToBuffer(pbrFile,(ULONG)dwSize,BufferA+4,(ULONG)dwSize,&sizefile);
  ulTestSize = (UCHAR)BufferA[4];
  ulTestSize <<= 8;
  ulTestSize |= (UCHAR)BufferA[5];
  ulTestSize <<= 8;
  ulTestSize |= (UCHAR)BufferA[6];
  ulTestSize <<= 8;
  ulTestSize |= (UCHAR)BufferA[7];
  hBuffB = GlobalAlloc(GMEM_MOVEABLE,dwSize);
  if (!hBuffB)
  {
    CloseHandle(hFile);
    MessageBox(hWnd,"No hay memoria disponible para descompresion","Error",MB_OK);
    return (NULL);
  }
  BufferB=(LPSTR)GlobalLock(hBuffB);

// Se expande archivo comprimido para supervisar compresion antes de la transmision
```

```
   rc = ExpandBufferToBuffer(BufferA+4,(ULONG)sizefile,BufferB,(ULONG)dwSize,&ulTextSize);

   hFile = CreateFile( "LZH_EBB.RCV",        // Name of file
           GENERIC_WRITE,                    //
           0,                          // no permit file sharing
           NULL,                       // default security
           CREATE_ALWAYS,              // do not create new file
           FILE_ATTRIBUTE_NORMAL,      // file attributes
           NULL );                     // template for attributes
   if (hFile == INVALID_HANDLE_VALUE)
   {
      MessageBox(hWnd,"No se abrio archivo  para expansion","Error",MB_OK);
      return(NULL);
      }

   WriteFile(hFile,          // File to write in
      BufferB,               // data to write
      ulTextSize,            // num bytes to write
      &dwBytesWritten,       // value returned
      NULL) :                // no overlapping I/O
   CloseHandle( hFile );

   return( BufferA );
}



/*---------------------------------------------------
    GET FILE NAME
    Invoke the File Open or File Save As common dialog box.
    If the bOpenName parameter is TRUE, the procedure runs
    the OpenFileName dialog box.

    The lpszFile and lpszFileTitle parameters should point to
    buffers of size MAX_PATH and _MAX_FNAME, respectively.

    RETURN
    TRUE if the dialog box closes without error.  If the dialog
    box returns TRUE, then lpszFile and lpszFileTitle point to
    the new file path and name, respectively.
---------------------------------------------------*/

BOOL GetFileName (
   HWND  hWnd,
   BOOL  bOpenName,              // open file dlg or save as dlg
   LPSTR lpszFile,               // buffer for file path
   LPSTR lpszFileTitle )         // buffer for file name
{
   OPENFILENAME ofn;
   CONST char *szFilter[] =    // filters for the file name combo box
   {
      "All Files (*.*)\0 *.* \0\0 transfer files (*.tra)\0 *.tra\0 "
   };

   /* initialize structure for the common dialog box */
   ofn.lStructSize       = sizeof( OPENFILENAME );
   ofn.hwndOwner         = hWnd;
   ofn.hInstance         = GetWindowInstance( hWnd );
   ofn.lpstrFilter       = szFilter[0];
   ofn.lpstrCustomFilter = NULL;
   ofn.nMaxCustFilter    = 0;
   ofn.nFilterIndex      = 1;
   ofn.lpstrFile         = lpszFile;
   ofn.nMaxFile          = MAX_PATH;
```

```
    ofn.lpstrFileTitle   = lpszFileTitle;
    ofn.nMaxFileTitle    = _MAX_FNAME;
    ofn.lpstrInitialDir  = NULL;
    ofn.lpstrTitle       = NULL;
    ofn.nFileOffset      = 0;
    ofn.nFileExtension   = 0;
    ofn.lpstrDefExt      = "";
    ofn.lCustData        = 0;
    ofn.lpfnHook         = NULL;
    ofn.lpTemplateName   = NULL;

    /* invoke the common dialog box */
    if (bOpenName)                      // Open a file
    {
        ofn.Flags = OFN_HIDEREADONLY | OFN_PATHMUSTEXIST |
                OFN_FILEMUSTEXIST;
        return( GetOpenFileName(&ofn) );
    }
    else                                // Save As...
    {
        ofn.Flags = OFN_HIDEREADONLY | OFN_OVERWRITEPROMPT;
        return( GetSaveFileName(&ofn) );
    }
}
```

```
/*****************************************************************************
Client.c - client application to receive files trough NetDDE APIs

This application connects to the Server application through the NDDE share,
Server$.  This share is created when the server app is started. Once created
the client app will cause the server to startup upon connection to the share.
The DDE is implemented using the DDE Management Library.
Program base routines for

   DDEML from: CD Windows NT ver 3.5 and Win32 SDK
   pre-release; CD ROM Developer Network Development Plataform, Microsoft.

   FILE from: Myers, Brian and Hammer, Erick
   "Mastering Windows NT Programming", Sybex, USA, 1993.

   COMPRESS/EXPAND from   Haruyasu Yoshizaki 1988/11/20, shareware

Alberto Paz Gutiérrez                    UIACPyP del CCH, UNAM
                                         Mayo, Noviembre de 1995
*****************************************************************************/
#include <windows.h>
#include <ddeml.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "client.h"
#include "constant.h"


/* DlbFILE */
#include <windowsx.h>
#include <commdlg.h>

#include "dblib.h"
/* Huffman */
#include "huffman.h"


/*-------------------------------------------------
     PROTOTYPE of private function
  -------------------------------------------------*/

static BOOL DoOpenFileDlg( HWND hWnd, LPSTR lpszName, LPSTR lpszPath );
BOOL WriteRcvdFile(HWND hwnd, LPSTR szFileName, LPSTR szPathName, BOOL bTest,
                   PVOID pFile);
int sizestr(char * str);

/* DbFile */
/*****************************************************************************
* Global Variables
*****************************************************************************/
char szShare[]  = SHARENAME;          // for NDDE - name of share
char szAppName[] = "Cliente" ;
char szServerName[MAX_COMPUTERNAME_LENGTH+3]; // name of remote computer
char szService[32];                   // for NDDE - remote computer name
char szItem[] = "Primero" ;           // data item to request

DWORD idInst ;
HCONV hConv ;
HWND  hwnd ;
HMENU hmenu ;
HINSTANCE hInst;
/* For file handling */
char szFileName[_MAX_FNAME];     /* file name for received file */
```

```
char szPathName[_MAX_PATH];       /* path name for received file */
char *lpMapView;                  /* pointer to a view           */
ULONG sizeFile;                   /* size of received Data       */
ULONG dwSize;                     /* size of bytes send          */
LPSTR pFile;                      /* pointer to the File in memory */
#define PATH_OFFSET 0             /* place in buffer where path begins */
#define FNAME_OFFSET  MAX_PATH    /* place in buffer where name begins */
/*********************************************************************
* Procedure: WinMain
*
* Purpose: Creates and initializes the main window and the DDE conversation
*********************************************************************/
int PASCAL WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    LPSTR lpszCmdLine, int nCmdShow)
    {
    MSG     msg ;
    FARPROC pfnDdeCallback ;
    WNDCLASS wc ;

    // register and create main window
    if (!hPrevInstance)
        {
        wc.style         = CS_HREDRAW | CS_VREDRAW ;
        wc.lpfnWndProc   = ClientWndProc ;
        wc.cbClsExtra    = 0 ;
        wc.cbWndExtra    = 0 ;
        wc.hInstance     = hInstance ;
        wc.hIcon         = LoadIcon (NULL, IDI_APPLICATION) ;
        wc.hCursor       = LoadCursor (NULL, IDC_ARROW) ;
        wc.hbrBackground = GetStockObject (WHITE_BRUSH) ;
        wc.lpszMenuName  = MAKEINTRESOURCE(IDM_CLIENT) ;
        wc.lpszClassName = szAppName ;

        RegisterClass (&wc) ;
        }

    hInst = hInstance;

    hwnd = CreateWindow (szAppName, "ESIME Cliente H->C",
                         WS_OVERLAPPEDWINDOW,
                         CW_USEDEFAULT, CW_USEDEFAULT,
                         300, 100,
                         NULL, NULL, hInstance, NULL) ;

    hmenu = GetMenu( hwnd );

    ShowWindow (hwnd, nCmdShow) ;
    UpdateWindow (hwnd) ;

    // Initialize DDEML conversation
    pfnDdeCallback = MakeProcInstance ((FARPROC) DdeCallback, hInstance) ;

    if (DdeInitialize (&idInst, (PFNCALLBACK) pfnDdeCallback,
                       APPCLASS_STANDARD | APPCMD_CLIENTONLY, 0L))
        {
        MessageBox (hwnd, "No se inicializó el cliente!",
                    szAppName, MB_ICONEXCLAMATION | MB_OK) ;

        DestroyWindow (hwnd) ;
        return FALSE ;
        }

    // process messages until quit
    while (GetMessage (&msg, NULL, 0, 0))
```

```
        {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
        }

    // Uninitialize DDEML
    DdeUninitialize (idInst) ;

    return msg.wParam ;
    }

/************************************************************************
* Function: DdeCallback ( UINT, UINT, HCONV, HSZ, HSZ, HDDEDATA,
*                         DWORD, DWORD )
*
* Purpose: Handle DDE messages from DDEML
************************************************************************/
HDDEDATA CALLBACK DdeCallback (UINT iType, UINT iFmt, HCONV hConv,
                               HSZ hsz1, HSZ hsz2, HDDEDATA hData,
                               DWORD dwData1, DWORD dwData2)
{

    unsigned dwResult;          /* bytes received */
    switch (iType)
        {
        case XTYP_ADVDATA:

            // Check for matching format and data item
            if (iFmt != CF_TEXT)
                return DDE_FNOTPROCESSED ;

            // getdata being advised
            sizeFile = DdeGetData(hData,NULL, 0,0);
            if (sizeFile) {
             /* allocate space for the receiving file */
                pFile = GlobalAllocPtr(GHND,sizeFile);
                if (!pFile) {
                    MessageBox (hwnd, "No se dio espacio de memoria",
                        szAppName, MB_ICONEXCLAMATION | MB_OK) ;
                    return DDE_FNOTPROCESSED;
                }
                dwResult = DdeGetData (hData, pFile, sizeFile, 0);
                dwSize = (UCHAR)pFile[0];
                dwSize <<=8;
                dwSize |= (UCHAR)pFile[1];
                dwSize <<=8;
                dwSize |= (UCHAR)pFile[2];
                dwSize <<=8;
                dwSize |= (UCHAR)pFile[3];
                /* compare data length to bytes copied */
                if (dwResult != sizeFile){
                    GlobalFree(pFile);
                    MessageBox (hwnd, "No se leyeron bytes",
                        szAppName, MB_ICONEXCLAMATION | MB_OK) ;
                    pFile = NULL;
                    return DDE_FNOTPROCESSED ;
                }
            }

            InvalidateRect (hwnd, NULL, TRUE) ;
//**/       WriteRcvdFile(hwnd, "archivo","d:\\trabajo\\metr\\notdde",FALSE,pFile);
            return DDE_FACK ;

        case XTYP_DISCONNECT: // server is closing connection
```

```
            hConv = 0 ;

            // clear data
            *szServerName = 0;
             GlobalFree(pFile);
             pFile = NULL;

            // reset menus
            EnableMenuItem(hmenu, IDM_CONNECT, MF_ENABLED );
            EnableMenuItem(hmenu, IDM_DISCONNECT, MF_GRAYED );
            MessageBox (hwnd, "El servidor se ha desconectado.",
                    szAppName, MB_ICONASTERISK | MB_OK) ;
            InvalidateRect( hwnd, NULL, TRUE );
            return 0 ;
           }

    return 0 ;
}


/*********************************************************************
 * Function: ClientWndProc ( HWND, UINT, UINT, LONG )
 *
 * Purpose: handle main window messages
 *********************************************************************/
LRESULT FAR CALLBACK ClientWndProc (HWND hwnd, UINT message, WPARAM wParam,
                                    LPARAM lParam)
    {
    char        szBuffer [256] ;
     HDC        hdc ;
    HSZ         hszService, hszShare, hszItem ;
    PAINTSTRUCT ps ;
    FARPROC     pfnDlgProc;
    BOOL        bRet;

    switch (message)
       {
       case WM_CREATE:
          // init vars
          *szServerName = 0;
          return 0 ;

       case WM_USER_INITIATE:

             // Try connecting to ndde share "\\servername\NDDE$"
          wsprintf( (LPSTR) szService, "\\\\%s\\NDDE$",
                (LPSTR) szServerName );

          hszService = DdeCreateStringHandle (idInst, szService, 0) ;
          hszShare   = DdeCreateStringHandle (idInst, szShare,   0) ;

          hConv = DdeConnect (idInst, hszService, hszShare, NULL) ;

          DdeFreeStringHandle (idInst, hszService) ;
          DdeFreeStringHandle (idInst, hszShare) ;

          if (!hConv)
             {
             MessageBox (hwnd, "No pudo conectarse al servidor",
                    szAppName, MB_ICONEXCLAMATION | MB_OK) ;

             return 0 ;
             }
```

```
        // Request notification of item
        hszItem = DdeCreateStringHandle (idInst, szItem, 0) ;

        DdeClientTransaction (NULL, 0, hConv, hszItem, CF_TEXT,
                              XTYP_ADVSTART | XTYPF_ACKREQ,
                              TIMEOUT_ASYNC, NULL) ;

        DdeFreeStringHandle (idInst, hszItem) ;

        return 0 ;

    case WM_PAINT:
        // paint info in client area
        hdc = BeginPaint (hwnd, &ps) ;
        wsprintf(szBuffer,"Conectado al servidor: %s", szServerName);
        TextOut (hdc, 5, 5, szBuffer, strlen(szBuffer));
/**/    wsprintf(szBuffer,"Bytes recibidos: %d->%d", sizeFile,dwSize);
        TextOut (hdc, 5, 25, szBuffer, strlen(szBuffer));
        EndPaint (hwnd, &ps) ;
        return 0 ;

    case WM_COMMAND:
      switch( LOWORD(wParam) ) {

        case IDM_CONNECT:

          // get computer name to connect to
              pfnDlgProc=MakeProcInstance((FARPROC)ConnectDlgProc, hInst);
          bRet = (BOOL)DialogBox(hInst, MAKEINTRESOURCE(IDD_CONNECT),
                        hwnd, (DLGPROC)pfnDlgProc);
          FreeProcInstance(pfnDlgProc);

          // server name entered so initiate connection
          if ( bRet ) {
            EnableMenuItem(hmenu, IDM_CONNECT, MF_GRAYED );
            EnableMenuItem(hmenu, IDM_DISCONNECT, MF_ENABLED );
            SendMessage (hwnd, WM_USER_INITIATE, 0, 0L) ;
            }
          InvalidateRect( hwnd, NULL, TRUE );
          return 0 ;

        case IDM_DISCONNECT:
          // Stop the advises
          hszItem = DdeCreateStringHandle (idInst, szItem, 0) ;

          DdeClientTransaction (NULL, 0, hConv, hszItem, CF_TEXT,
                        XTYP_ADVSTOP, TIMEOUT_ASYNC, NULL) ;

          DdeFreeStringHandle (idInst, hszItem) ;

          // Disconnect the conversation
          DdeDisconnect (hConv) ;
          EnableMenuItem(hmenu, IDM_CONNECT, MF_ENABLED );
          EnableMenuItem(hmenu, IDM_DISCONNECT, MF_GRAYED );
          *szServerName = 0;
          InvalidateRect(hwnd, NULL, TRUE);
          return 0 ;
        case IDM_SAVE:
          /********** File Write **********/

           WriteRcvdFile(hwnd,szFileName,szPathName,TRUE,pFile);
          return 0 ;
```

```
            case IDM_EXIT:
               PostMessage( hwnd, WM_CLOSE, 0, 0L );
               return 0 ;
            }
         break;

      case WM_CLOSE:
         if (!hConv)
            break ;

         // Stop the advises
         hszItem = DdeCreateStringHandle (idInst, szItem, 0) ;

         DdeClientTransaction (NULL, 0, hConv, hszItem, CF_TEXT,
                               XTYP_ADVSTOP, TIMEOUT_ASYNC, NULL) ;

         DdeFreeStringHandle (idInst, hszItem) ;

         // Disconnect the conversation
         DdeDisconnect (hConv) ;
         DestroyWindow( hwnd );
         break ;

      case WM_DESTROY:
         PostQuitMessage (0) ;
         return 0 ;
      }
   return DefWindowProc (hwnd, message, wParam, lParam) ;
   }

/*****************************************************************
* Function: ConnectDlgProc ( HWND, UINT, UINT, LONG )
*
* Purpose: Get computer name for NDDE conversation
*****************************************************************/
BOOL CALLBACK ConnectDlgProc(HWND hDlg, UINT msg, WPARAM wParam, LPARAM lParam)
{

   switch (msg)
      {
      case WM_INITDIALOG:
         SendDlgItemMessage(hDlg, IDE_SERVERNAME, EM_LIMITTEXT, MAX_COMPUTERNAME_LENGTH+2 , 0L);
         return TRUE;

      case WM_COMMAND:
         switch (LOWORD(wParam))
            {
            case IDOK:
               GetDlgItemText(hDlg, IDE_SERVERNAME, szServerName, sizeof(szServerName));

               /** if no text entered **/
               if ( *szServerName == 0 ) {
                        MessageBox( hDlg, "Debe darse el nombre de una computadora",
                           szAppName, MB_ICONEXCLAMATION | MB_OK );
                  break;
                  }

               /** remove any prepended "\\" **/
               if ( *szServerName == '\\' )
                  strcpy( szServerName, szServerName+2);

                  EndDialog(hDlg, TRUE);
                  break;
```

```
            case IDCANCEL:
                EndDialog(hDlg, FALSE);
              break;

            default:
              break;
            }
          break;

        default:
          break;
        }

    return FALSE;
}

/*-------------------------------------------------

    from Mastering Windows NT Programming
    copyright 1993 by Brian Myers & Eric Hamer

    WriteRcvdFile
    Write all the data into a disk file.  If the caller
    sets bAsk to TRUE, show the save file dialog first to let the
    user pick a file name.  If the pointer parameters are
    non-null, their contents are used to initialize the dialog
    box settings.

    RETURN
    TRUE for success; FALSE for errors.  Also, if the function
    returns TRUE and lpszName and lpszFile are non-null, then
    afterwards they point to the name and path the user chose.
-------------------------------------------------*/

BOOL WriteRcvdFile(
    HWND hWnd,
    LPSTR lpszName,              // name of file
    LPSTR lpszPath,              // full path of file
    BOOL bAsk,                   // ask user for file name?
    PVOID pDIB )                 // the file
{
    HANDLE hFile;                // handle to the file
    char szName[_MAX_FNAME];     // name of disk file
    char szPath[_MAX_PATH];      // full path of disk file
    DWORD dwBytesWritten;        // number of bytes written to disk
    BOOL bTest;                  // for testing function returns
    ULONG   NData;               // number of bytes in data
    short rc;
    ULONG ulTextSize;

    /* For Huffman compressing */
    HANDLE hBuf;
    LPSTR Buffer;


    /* copy user's strings to full-sized buffers */
    if (lpszName)
    {
      lstrcpy( szName, lpszName );
    }
    if (lpszPath)
    {
      lstrcpy( szPath, lpszPath );
    }
```

```
/* Expande archivo recibido */
  ulTextSize= (UCHAR)pFile[4];
  ulTextSize <<= 8;
  ulTextSize |= (UCHAR)pFile[5];
  ulTextSize <<= 8;
  ulTextSize |=  (UCHAR)pFile[6];
  ulTextSize <<= 8;
  ulTextSize |=  (UCHAR)pFile[7];
  hBuf = GlobalAlloc(GMEM_MOVEABLE,ulTextSize);

  if (!hBuf)
  {
     CloseHandle(hFile);
     MessageBox(hWnd,"No hay memoria disponible para descompresion","Error",MB_OK);
     return (FALSE);
  }
  Buffer=(LPSTR)GlobalLock(hBuf);

  rc=ExpandBufferToBuffer(pFile+4,dwSize,Buffer,ulTextSize,&NData);

  /* If caller said to confirm name, show dialog */
  if (bAsk)
  {
     bTest = GetFileName( hWnd,     // ask the user for a file name
        FALSE,                  // FALSE to save (not to open)
        szPath,                 // buffer for path name
        szName );               // buffer for file name

     if (!bTest)
     {
        return( FALSE );        // the user pressed Cancel
     }
  }

  /* open the file */
  hFile = CreateFile( szPath,             // path of file
        GENERIC_WRITE,              // write-only access
        0,                      // prohibit file sharing
        NULL,                   // default security
        CREATE_ALWAYS,              // overwrite old file
        FILE_ATTRIBUTE_NORMAL,      // file attributes
        NULL );                 // template for attributes
  if (hFile == INVALID_HANDLE_VALUE)
  {
     MessageBox(hWnd, "No puede abrirse el archivo", "Error", MB_OK);
     return( FALSE );
  }

  /* write to the file */

  bTest = WriteFile( hFile,               // file to write in
        Buffer,                 // data to write
        (DWORD)NData,               // num bytes to write
        &dwBytesWritten,            // value returned
        NULL );                 // no overlapping I/O
  CloseHandle( hFile );
  if ((!bTest) || (dwBytesWritten != NData))
  {
     MessageBox( hWnd, "no escribió archivo completo", "Error", MB_OK );
     return( FALSE );
  }
```

```
    /* if the caller gave name pointers, use them to return strings */
    if (lpszName)
    {
        lstrcpy( lpszName, szName );
    }
    if (lpszPath)
    {
        lstrcpy( lpszPath, szPath );
    }

    return( TRUE );
}


/*----------------------------------------------------------
  GET FILE NAME
  Invoke the File Open or File Save As common dialog box. If the bOpenName
  parameter is TRUE, the procedure runs the OpenFileName dialog box.

  The lpszFile and lpszFileTitle parameters should point to buffers of size
  MAX_PATH and _MAX_FNAME, respectively.

  RETURN
  TRUE if the dialog box closes without error. If the dialog box returns TRUE,
  then lpszFile and lpszFileTitle point to the new file path and name, respectively.
  ----------------------------------------------------------*/

BOOL GetFileName (
    HWND hWnd,
    BOOL  bOpenName,              // open file dlg or save as dlg
    LPSTR lpszFile,               // buffer for file path
    LPSTR lpszFileTitle )         // buffer for file name
{
    OPENFILENAME ofn;
    CONST char *szFilter[] =    // filters for the file name combo box
    {
        "Transfer files (*.trs)\0 *.trs\0 All Files (*.*)\0 *.* \0\0"
    };

    /* initialize structure for the common dialog box */
    ofn.lStructSize       = sizeof( OPENFILENAME );
    ofn.hwndOwner         = hWnd;
    ofn.hInstance         = GetWindowInstance( hWnd );
    ofn.lpstrFilter       = szFilter[0];
    ofn.lpstrCustomFilter = NULL;
    ofn.nMaxCustFilter    = 0;
    ofn.nFilterIndex      = 1;
    ofn.lpstrFile         = lpszFile;
    ofn.nMaxFile          = MAX_PATH;
    ofn.lpstrFileTitle    = lpszFileTitle;
    ofn.nMaxFileTitle     = _MAX_FNAME;
    ofn.lpstrInitialDir   = NULL;
    ofn.lpstrTitle        = NULL;
    ofn.nFileOffset       = 0;
    ofn.nFileExtension    = 0;
    ofn.lpstrDefExt       = "rcv";
    ofn.lCustData         = 0;
    ofn.lpfnHook          = NULL;
    ofn.lpTemplateName    = NULL;

    /* invoke the common dialog box */
    if (bOpenName)                      // Open a file
    {
        ofn.Flags = OFN_HIDEREADONLY | OFN_PATHMUSTEXIST |
```

```
            OFN_FILEMUSTEXIST;
        return( GetOpenFileName(&ofn) );
    }
    else                        // Save As...
    {
        ofn.Flags = OFN_HIDEREADONLY | OFN_OVERWRITEPROMPT;
        return( GetSaveFileName(&ofn) );
    }
}
```

```
typedef unsigned char   UCHAR;
typedef char far *      PCHAR;
//typedef unsigned      USHORT;
typedef short           SHORT;
typedef unsigned long   ULONG;
typedef unsigned long far * PULONG;
typedef void far *      PVOID;
typedef int             HFILE;

#ifdef INCLUDE_HUFFSTUFF        /* Just for huffman routines    */

#define N       4096            /* buffer size */
#define F       60              /* lookahead buffer size */
#define THRESHOLD   2
#define NIL     N               /* leaf of tree */


#define N_CHAR      (256 - THRESHOLD + F)
            /* kinds of characters (character code = 0..N_CHAR-1) */
#define T       (N_CHAR * 2 - 1) /* size of table */
#define R       (T - 1)         /* position of root */
#define MAX_FREQ    0x8000      /* updates tree when the */
                        /* root frequency comes to this value. */

struct _HuffStuff{
   SHORT    sType;

   HFILE    hInput;
   PVOID    pInput;
   SHORT    (*fGetNextChar)(struct _HuffStuff far *);
   ULONG    ulMaxInput;

   HFILE    hOutput;
   PVOID    pOutput;
   SHORT    (*fPutNextChar)(struct _HuffStuff far *, USHORT);
   ULONG    ulMaxOutput;

   ULONG    textsize;
   ULONG    codesize;

   UCHAR    text_buf[N + F - 1];

   SHORT    match_position;
   SHORT    match_length;
   SHORT    lson[N + 1];
   SHORT    rson[N + 257];
   SHORT    dad[N + 1];

   USHORT   freq[T + 1];        /* frequency table */

   SHORT    prnt[T + N_CHAR];   /* pointers to parent nodes, except for the */
                        /* elements [T..T + N_CHAR - 1] which are used to get */
                        /* the positions of leaves corresponding to the codes. */
```

```
    SHORT   son[T];         /* pointers to child nodes (son[i], son[i] + 1) */

    USHORT  getbuf;
    UCHAR   getlen;
    USHORT  putbuf;
    UCHAR   putlen;
    USHORT  code;
    USHORT  len;
    };
typedef struct _HuffStuff far * PHUFFSTUFF;
#endif

#define HUFFMAN_FILE_TO_FILE        0x0001
#define HUFFMAN_FILE_TO_BUFFER      0x0002
#define HUFFMAN_BUFFER_TO_FILE      0x0004
#define HUFFMAN_BUFFER_TO_BUFFER    0x0008

#define HUFFMAN_NO_ERROR            0
#define HUFFMAN_FILE_WRITE_ERROR    100
#define HUFFMAN_FILE_LENGTH_ERROR   101
#define HUFFMAN_MEMORY_ERROR        102
#define HUFFMAN_FILE_READ_ERROR     103
#define WEP_SYSTEM_EXIT             1     /* Definidos en Windows.h */
#define WEP_FREE_DLL                0     /* MSVC V 1.5 */

/*
*  HUFFMAN1
*/
SHORT FAR PASCAL CompressFileToFile(HFILE , HFILE );
SHORT FAR PASCAL ExpandFileToFile(HFILE , HFILE );
SHORT FAR PASCAL CompressFileToBuffer(HFILE , PCHAR, ULONG, PULONG);
SHORT FAR PASCAL ExpandFileToBuffer(HFILE , PCHAR, ULONG, PULONG);
SHORT FAR PASCAL CompressBufferToFile( PCHAR, ULONG, HFILE);
SHORT FAR PASCAL ExpandBufferToFile( PCHAR, ULONG, HFILE);
/*
*  HUFFMAN5
*/
SHORT FAR PASCAL CompressBufferToBuffer(PCHAR, ULONG, PCHAR, ULONG, PULONG);
SHORT FAR PASCAL ExpandBufferToBuffer(PCHAR, ULONG, PCHAR, ULONG, PULONG);
/*
*  HUFFMAN1
*/
#ifdef INCLUDE_HUFFSTUFF        /* Just for huffman routines    */
SHORT GetNextFileChar(PHUFFSTUFF);
SHORT PutNextFileChar(PHUFFSTUFF, USHORT);
/*
*  HUFFMAN2
*/
SHORT Expand(PHUFFSTUFF);
SHORT ExpandChar(PHUFFSTUFF);
SHORT ExpandPosition(PHUFFSTUFF);
SHORT PutCode(PHUFFSTUFF, SHORT, USHORT);
```

```
/*
*  HUFFMAN3
*/
SHORT Compress(PHUFFSTUFF);
SHORT CompressChar(PHUFFSTUFF, USHORT);
SHORT CompressPosition(PHUFFSTUFF, USHORT);
SHORT CompressEnd(PHUFFSTUFF);
SHORT GetBit(PHUFFSTUFF);
SHORT GetByte(PHUFFSTUFF);
/*
*  HUFFMAN4
*/
PHUFFSTUFF StartHuff(void);
void EndHuff(PHUFFSTUFF);
void ReConstructTree(PHUFFSTUFF);
void UpdateTree(PHUFFSTUFF, SHORT);
void InitTree(PHUFFSTUFF);
void InsertNode(PHUFFSTUFF, SHORT);
void DeleteNode(PHUFFSTUFF, SHORT);
/*
*  HUFFMAN5
*/
SHORT GetNextBufferChar(PHUFFSTUFF);
SHORT PutNextBufferChar(PHUFFSTUFF, USHORT);
#endif
```

```
#include <windows.h>
#include "sys\types.h"
#include "sys\stat.h"

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

#define INCLUDE_HUFFSTUFF
#include "huffman.h"


SHORT FAR PASCAL CompressFileToFile(HFILE Input, HFILE Output)
{
   SHORT     rc;
   PHUFFSTUFF pHS;
   HANDLE    hStatBuf;
   struct stat * pStatBuf;

   pHS = StartHuff();
   if (pHS == NULL)
      return(HUFFMAN_MEMORY_ERROR);
   pHS->sType = HUFFMAN_FILE_TO_FILE;
   pHS->hInput = Input;
   pHS->hOutput = Output;
   pHS->fGetNextChar = GetNextFileChar;
   pHS->fPutNextChar = PutNextFileChar;

   hStatBuf = LocalAlloc(LMEM_MOVEABLE, sizeof(struct stat));
   pStatBuf = (struct stat *)LocalLock(hStatBuf);
   fstat(pHS->hInput, pStatBuf);
   pHS->textsize = pStatBuf->st_size;
   LocalUnlock(hStatBuf);
   LocalFree(hStatBuf);
   if (PutNextFileChar(pHS, (USHORT)((pHS->textsize & 0xff000000L) >> 24)) == EOF){
      EndHuff(pHS);
      return(HUFFMAN_FILE_WRITE_ERROR);
      }
   if (PutNextFileChar(pHS, (USHORT)((pHS->textsize & 0xff0000L) >> 16)) == EOF){
      EndHuff(pHS);
      return(HUFFMAN_FILE_WRITE_ERROR);
      }
   if (PutNextFileChar(pHS, (USHORT)((pHS->textsize & 0xff00) >> 8)) == EOF){
      EndHuff(pHS);
      return(HUFFMAN_FILE_WRITE_ERROR);
      }
   if (PutNextFileChar(pHS, (USHORT)((pHS->textsize & 0xff))) == EOF){
      EndHuff(pHS);
      return(HUFFMAN_FILE_WRITE_ERROR);
      }
   if (pHS->textsize == 0){
      EndHuff(pHS);
```

```
        return(HUFFMAN_FILE_LENGTH_ERROR);
        }

    rc = Compress(pHS);

    EndHuff(pHS);
    return(rc);
}


SHORT FAR PASCAL CompressFileToBuffer(HFILE hInput, PCHAR pOutput,
                                      ULONG ulOutSize, PULONG pulOutSize)
{
    SHORT rc;
    PHUFFSTUFF pHS;
    HANDLE     hStatBuf;
    struct stat * pStatBuf;

    pHS = StartHuff();
    if (pHS == NULL)
        return(HUFFMAN_MEMORY_ERROR);
    pHS->sType = HUFFMAN_FILE_TO_FILE;
    pHS->hInput = hInput;
    pHS->pOutput = (PVOID)pOutput;
    pHS->fGetNextChar = GetNextFileChar;
    pHS->fPutNextChar = PutNextBufferChar;
    pHS->ulMaxOutput = ulOutSize;

    hStatBuf = LocalAlloc(LMEM_MOVEABLE, sizeof(struct stat));
    pStatBuf = (struct stat *)LocalLock(hStatBuf);
    fstat(pHS->hInput, pStatBuf);
    pHS->textsize = pStatBuf->st_size;
    LocalUnlock(hStatBuf);
    LocalFree(hStatBuf);

    if (PutNextBufferChar(pHS, (USHORT)((pHS->textsize & 0xff000000L) >> 24)) == EOF){
        EndHuff(pHS);
        return(HUFFMAN_FILE_WRITE_ERROR);
        }
    if (PutNextBufferChar(pHS, (USHORT)((pHS->textsize & 0xff0000L) >> 16)) == EOF){
        EndHuff(pHS);
        return(HUFFMAN_FILE_WRITE_ERROR);
        }
    if (PutNextBufferChar(pHS, (USHORT)((pHS->textsize & 0xff00) >> 8)) == EOF){
        EndHuff(pHS);
        return(HUFFMAN_FILE_WRITE_ERROR);
        }
    if (PutNextBufferChar(pHS, (USHORT)((pHS->textsize & 0xff))) == EOF){
        EndHuff(pHS);
        return(HUFFMAN_FILE_WRITE_ERROR);
        }
    if (pHS->textsize == 0){
        EndHuff(pHS);
```

```
        return(HUFFMAN_FILE_LENGTH_ERROR);
        }

    rc = Compress(pHS);

    *pulOutSize  = ulOutSize - pHS->ulMaxOutput;

    EndHuff(pHS);
    return(rc);
}


SHORT FAR PASCAL ExpandFileToFile(HFILE Input, HFILE Output)
{
    SHORT rc;
    PHUFFSTUFF pHS;

    pHS = StartHuff();
    if (pHS == NULL)
        return(HUFFMAN_MEMORY_ERROR);
    pHS->sType = HUFFMAN_FILE_TO_FILE;
    pHS->hInput = Input;
    pHS->hOutput = Output;
    pHS->fGetNextChar = GetNextFileChar;
    pHS->fPutNextChar = PutNextFileChar;

    pHS->textsize  = (UCHAR) GetNextFileChar(pHS);
    pHS->textsize <<= 8;
    pHS->textsize |= (UCHAR) GetNextFileChar(pHS);
    pHS->textsize <<= 8;
    pHS->textsize |= (UCHAR) GetNextFileChar(pHS);
    pHS->textsize <<= 8;
    pHS->textsize |= (UCHAR) GetNextFileChar(pHS);
    if (pHS->textsize == 0){
        EndHuff(pHS);
        return(HUFFMAN_FILE_LENGTH_ERROR);
        }

    rc = Expand(pHS);

    EndHuff(pHS);

    return(rc);
}


SHORT FAR PASCAL ExpandBufferToFile(PCHAR pInput, ULONG ulInSize,
                                    HFILE Output)
{
    SHORT rc;
    PHUFFSTUFF pHS;

    pHS = StartHuff();
```

```
    if (pHS == NULL)
        return(HUFFMAN_MEMORY_ERROR);
    pHS->sType = HUFFMAN_FILE_TO_FILE;
    pHS->plnput = (PVOID)plnput;
    pHS->hOutput = Output;
     pHS->fGetNextChar = GetNextBufferChar;
     pHS->ulMaxInput = ulInSize;
     pHS->fPutNextChar = PutNextFileChar;

    if (pHS->ulMaxInput < 5)
        return(HUFFMAN_FILE_READ_ERROR);
    pHS->textsize  = (UCHAR) GetNextBufferChar(pHS);
    pHS->textsize <<= 8;
    pHS->textsize |= (UCHAR) GetNextBufferChar(pHS);
    pHS->textsize <<= 8;
    pHS->textsize |= (UCHAR) GetNextBufferChar(pHS);
    pHS->textsize <<= 8;
    pHS->textsize |= (UCHAR) GetNextBufferChar(pHS);
    if (pHS->textsize == 0){
        EndHuff(pHS);
        return(HUFFMAN_FILE_LENGTH_ERROR);
         }

    rc = Expand(pHS);

     EndHuff(pHS);

     return(rc);
}


SHORT GetNextFileChar(PHUFFSTUFF pHS)
{
char c;

    if (_lread(pHS->hInput, &c, 1) != 1)
        return(EOF);
    return(c);
}


SHORT PutNextFileChar(PHUFFSTUFF pHS, USHORT c)
{
int result;

    result = _lwrite(pHS->hOutput, (LPSTR)&c, 1);
     if (result != 1)
        return(EOF);
    return(0);
}
```

```
#include <windows.h>

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define INCLUDE_HUFFSTUFF
#include "huffman.h"

UCHAR d_code[256] = {
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01,
    0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01,
    0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02,
    0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02, 0x02,
    0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03,
    0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03,
    0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04,
    0x05, 0x05, 0x05, 0x05, 0x05, 0x05, 0x05, 0x05,
    0x06, 0x06, 0x06, 0x06, 0x06, 0x06, 0x06, 0x06,
    0x07, 0x07, 0x07, 0x07, 0x07, 0x07, 0x07, 0x07,
    0x08, 0x08, 0x08, 0x08, 0x08, 0x08, 0x08, 0x08,
    0x09, 0x09, 0x09, 0x09, 0x09, 0x09, 0x09, 0x09,
    0x0A, 0x0A, 0x0A, 0x0A, 0x0A, 0x0A, 0x0A, 0x0A,
    0x0B, 0x0B, 0x0B, 0x0B, 0x0B, 0x0B, 0x0B, 0x0B,
    0x0C, 0x0C, 0x0C, 0x0C, 0x0D, 0x0D, 0x0D, 0x0D,
    0x0E, 0x0E, 0x0E, 0x0E, 0x0F, 0x0F, 0x0F, 0x0F,
    0x10, 0x10, 0x10, 0x10, 0x11, 0x11, 0x11, 0x11,
    0x12, 0x12, 0x12, 0x12, 0x13, 0x13, 0x13, 0x13,
    0x14, 0x14, 0x14, 0x14, 0x15, 0x15, 0x15, 0x15,
    0x16, 0x16, 0x16, 0x16, 0x17, 0x17, 0x17, 0x17,
    0x18, 0x18, 0x19, 0x19, 0x1A, 0x1A, 0x1B, 0x1B,
    0x1C, 0x1C, 0x1D, 0x1D, 0x1E, 0x1E, 0x1F, 0x1F,
    0x20, 0x20, 0x21, 0x21, 0x22, 0x22, 0x23, 0x23,
    0x24, 0x24, 0x25, 0x25, 0x26, 0x26, 0x27, 0x27,
    0x28, 0x28, 0x29, 0x29, 0x2A, 0x2A, 0x2B, 0x2B,
    0x2C, 0x2C, 0x2D, 0x2D, 0x2E, 0x2E, 0x2F, 0x2F,
    0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37,
    0x38, 0x39, 0x3A, 0x3B, 0x3C, 0x3D, 0x3E, 0x3F,
};

UCHAR d_len[256] = {
    0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03,
    0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03,
    0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03,
    0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03,
    0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04,
    0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04,
    0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04,
    0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04,
    0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04,
    0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04, 0x04,
    0x05, 0x05, 0x05, 0x05, 0x05, 0x05, 0x05, 0x05,
    0x05, 0x05, 0x05, 0x05, 0x05, 0x05, 0x05, 0x05,
    0x05, 0x05, 0x05, 0x05, 0x05, 0x05, 0x05, 0x05,
    0x05, 0x05, 0x05, 0x05, 0x05, 0x05, 0x05, 0x05,
    0x05, 0x05, 0x05, 0x05, 0x05, 0x05, 0x05, 0x05,
    0x05, 0x05, 0x05, 0x05, 0x05, 0x05, 0x05, 0x05,
    0x05, 0x05, 0x05, 0x05, 0x05, 0x05, 0x05, 0x05,
    0x06, 0x06, 0x06, 0x06, 0x06, 0x06, 0x06, 0x06,
```

```
    0x06, 0x06, 0x06, 0x06, 0x06, 0x06, 0x06, 0x06,
    0x06, 0x06, 0x06, 0x06, 0x06, 0x06, 0x06, 0x06,
    0x06, 0x06, 0x06, 0x06, 0x06, 0x06, 0x06, 0x06,
    0x06, 0x06, 0x06, 0x06, 0x06, 0x06, 0x06, 0x06,
    0x06, 0x06, 0x06, 0x06, 0x06, 0x06, 0x06, 0x06,
    0x07, 0x07, 0x07, 0x07, 0x07, 0x07, 0x07, 0x07,
    0x07, 0x07, 0x07, 0x07, 0x07, 0x07, 0x07, 0x07,
    0x07, 0x07, 0x07, 0x07, 0x07, 0x07, 0x07, 0x07,
    0x07, 0x07, 0x07, 0x07, 0x07, 0x07, 0x07, 0x07,
    0x07, 0x07, 0x07, 0x07, 0x07, 0x07, 0x07, 0x07,
    0x08, 0x08, 0x08, 0x08, 0x08, 0x08, 0x08, 0x08,
    0x08, 0x08, 0x08, 0x08, 0x08, 0x08, 0x08, 0x08,
};

SHORT Expand(PHUFFSTUFF pHS) /* recover data to original*/
{
    SHORT i, j, k, r, c;
    ULONG count;

    for (i = 0; i < N - F; i++)
        pHS->text_buf[i] = 0x20;
    r = N - F;
    for (count = 0; count < pHS->textsize; ) {
        c = ExpandChar(pHS);
        if (c < 256) {
            if (pHS->fPutNextChar(pHS, c) == EOF) {
                return(HUFFMAN_FILE_WRITE_ERROR);
            }
            pHS->text_buf[r++] = (UCHAR)c;
            r &= (N - 1);
            count++;
        } else {
            i = (r - ExpandPosition(pHS) - 1) & (N - 1);
            j = c - 255 + THRESHOLD;
            for (k = 0; k < j; k++) {
                c = pHS->text_buf[(i + k) & (N - 1)];
                if (pHS->fPutNextChar(pHS, c) == EOF) {
                    return(HUFFMAN_FILE_WRITE_ERROR);
                }
                pHS->text_buf[r++] = (UCHAR)c;
                r &= (N - 1);
                count++;
            }
        }
    }

    return(HUFFMAN_NO_ERROR);
}

SHORT ExpandChar(PHUFFSTUFF pHS)
{
    SHORT c;

    c = pHS->son[R];

    /* travel from root to leaf, */
    /* choosing the smaller child node (son[]) if the read bit is 0, */
    /* the bigger (son[]+1) if 1 */
    while (c < T) {
        c += GetBit(pHS);
        c = pHS->son[c];

    }
```

```
    c -= T;
    UpdateTree(pHS, c);
    return c;
}

SHORT ExpandPosition(PHUFFSTUFF pHS)
{
    USHORT i, j, c;

    /* recover upper 6 bits from table */
    i = GetByte(pHS);
    c = (unsigned)d_code[i] << 6;
    j = d_len[i];

    /* read lower 6 bits verbatim */
    j -= 2;
    while (j--) {
        i = (i << 1) + GetBit(pHS);
    }
    return (SHORT)(c | (i & 0x3f));
}


SHORT PutCode(PHUFFSTUFF pHS, SHORT l, USHORT c)    /* output c bits of code */
{
    pHS->putbuf |= c >> pHS->putlen;
    if ((pHS->putlen += l) >= 8) {
        if (pHS->fPutNextChar(pHS, pHS->putbuf >> 8) == EOF) {
            return(HUFFMAN_FILE_WRITE_ERROR);
        }
        if ((pHS->putlen -= 8) >= 8) {
            if (pHS->fPutNextChar(pHS, pHS->putbuf) == EOF) {
                return(HUFFMAN_FILE_WRITE_ERROR);
            }
            pHS->codesize += 2;
            pHS->putlen -= 8;
            pHS->putbuf = c << (l - pHS->putlen);
        } else {
            pHS->putbuf <<= 8;
            pHS->codesize++;
        }
    }
    return(HUFFMAN_NO_ERROR);
}
```

```
#include <windows.h>

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

#define INCLUDE_HUFFSTUFF
#include "huffman.h"

UCHAR p_len[64] = {
   0x03, 0x04, 0x04, 0x04, 0x05, 0x05, 0x05, 0x05,
   0x05, 0x05, 0x05, 0x05, 0x06, 0x06, 0x06, 0x06,
   0x06, 0x06, 0x06, 0x06, 0x06, 0x06, 0x06, 0x06,
   0x07, 0x07, 0x07, 0x07, 0x07, 0x07, 0x07, 0x07,
   0x07, 0x07, 0x07, 0x07, 0x07, 0x07, 0x07, 0x07,
   0x07, 0x07, 0x07, 0x07, 0x07, 0x07, 0x07, 0x07,
   0x08, 0x08, 0x08, 0x08, 0x08, 0x08, 0x08, 0x08,
   0x08, 0x08, 0x08, 0x08, 0x08, 0x08, 0x08, 0x08
};
UCHAR p_code[64] = {
   0x00, 0x20, 0x30, 0x40, 0x50, 0x58, 0x60, 0x68,
   0x70, 0x78, 0x80, 0x88, 0x90, 0x94, 0x98, 0x9C,
   0xA0, 0xA4, 0xA8, 0xAC, 0xB0, 0xB4, 0xB8, 0xBC,
   0xC0, 0xC2, 0xC4, 0xC6, 0xC8, 0xCA, 0xCC, 0xCE,
   0xD0, 0xD2, 0xD4, 0xD6, 0xD8, 0xDA, 0xDC, 0xDE,
   0xE0, 0xE2, 0xE4, 0xE6, 0xE8, 0xEA, 0xEC, 0xEE,
   0xF0, 0xF1, 0xF2, 0xF3, 0xF4, 0xF5, 0xF6, 0xF7,
   0xF8, 0xF9, 0xFA, 0xFB, 0xFC, 0xFD, 0xFE, 0xFF
};


/* compression */

SHORT Compress(PHUFFSTUFF pHS)  /* compression */
{
   SHORT  i, c, len, r, s, last_match_length;

   pHS->textsize = 0;          /* rewind and re-read */
   InitTree(pHS);
   s = 0;
   r = N - F;
   for (i = s; i < r; i++)
      pHS->text_buf[i] = 0x20;
   for (len = 0; len < F && (c = pHS->fGetNextChar(pHS)) != EOF; len++)
      pHS->text_buf[r + len] = (UCHAR)c;
   pHS->textsize = len;
   for (i = 1; i <= F; i++)
      InsertNode(pHS, r - i);
   InsertNode(pHS, r);
   do {
      if (pHS->match_length > len)
         pHS->match_length = len;
      if (pHS->match_length <= THRESHOLD) {
         pHS->match_length = 1;
         if (CompressChar(pHS, pHS->text_buf[r]))
            return(HUFFMAN_FILE_WRITE_ERROR);

      } else {
         if (CompressChar(pHS, 255 - THRESHOLD + pHS->match_length))
            return(HUFFMAN_FILE_WRITE_ERROR);
         if (CompressPosition(pHS, pHS->match_position))
            return(HUFFMAN_FILE_WRITE_ERROR);
```

```
        }
        last_match_length = pHS->match_length;
        for (i = 0; i < last_match_length &&
                (c = pHS->fGetNextChar(pHS)) != EOF; i++) {
            DeleteNode(pHS, s);
            pHS->text_buf[s] = (UCHAR)c;
            if (s < F - 1)
                pHS->text_buf[s + N] = (UCHAR)c;
            s = (s + 1) & (N - 1);
            r = (r + 1) & (N - 1);
            InsertNode(pHS, r);
        }
        pHS->textsize += i;
        while (i++ < last_match_length) {
            DeleteNode(pHS, s);
            s = (s + 1) & (N - 1);
            r = (r + 1) & (N - 1);
            if (--len) InsertNode(pHS, r);
        }
    } while (len > 0);
    if (CompressEnd(pHS))
        return(HUFFMAN_FILE_WRITE_ERROR);
    return(HUFFMAN_NO_ERROR);
}

SHORT CompressChar(PHUFFSTUFF pHS, USHORT c)
{
    USHORT i;
    SHORT j, k;

    i = 0;
    j = 0;
    k = pHS->prnt[c + T];

    /* travel from leaf to root */
    do {
        i >>= 1;

        /* if node's address is odd-numbered, choose bigger brother node */
        if (k & 1) i += 0x8000;

        j++;
    } while ((k = pHS->prnt[k]) != R);
    if (PutCode(pHS, j, i))
        return(HUFFMAN_FILE_WRITE_ERROR);
    pHS->code = i;
    pHS->len = j;
    UpdateTree(pHS, c);
    return(HUFFMAN_NO_ERROR);
}

SHORT CompressPosition(PHUFFSTUFF pHS, USHORT c)
{
    USHORT i;

    /* output upper 6 bits by table lookup */
    i = c >> 6;
    if (PutCode(pHS, p_len[i], (unsigned)p_code[i] << 8))
        return(HUFFMAN_FILE_WRITE_ERROR);
    /* output lower 6 bits verbatim */
    if (PutCode(pHS, 6, (c & 0x3f) << 10))
        return(HUFFMAN_FILE_WRITE_ERROR);
    return(HUFFMAN_NO_ERROR);
}
```

```
SHORT CompressEnd(PHUFFSTUFF pHS)
{
   if (pHS->putlen) {
      if (pHS->fPutNextChar(pHS, pHS->putbuf >> 8) == EOF) {
         return(HUFFMAN_FILE_WRITE_ERROR);
      }
      pHS->codesize++;
   }
    return(HUFFMAN_NO_ERROR);
}


SHORT GetBit(PHUFFSTUFF pHS)    /* get one bit */
{
   USHORT i;

   while (pHS->getlen <= 8) {
      if ((SHORT)(i = pHS->fGetNextChar(pHS)) < 0) i = 0;
      pHS->getbuf |= i << (8 - pHS->getlen);
      pHS->getlen += 8;
   }
   i = pHS->getbuf;
   pHS->getbuf <<= 1;
   pHS->getlen--;
   return (int)((i & 0x8000) >> 15);
}

SHORT GetByte(PHUFFSTUFF pHS)   /* get one byte */
{
   USHORT i;

   while (pHS->getlen <= 8) {
      if ((SHORT)(i = pHS->fGetNextChar(pHS)) < 0) i = 0;
      pHS->getbuf |= i << (8 - pHS->getlen);
      pHS->getlen += 8;
   }
   i = pHS->getbuf;
   pHS->getbuf <<= 8;
   pHS->getlen -= 8;
   return (SHORT)((i & 0xff00) >> 8);
}
```

```
#include <windows.h>
#include <windowsx.h>

#include <stdlib.h>
#include <string.h>
#include <ctype.h>

#define INCLUDE_HUFFSTUFF
#include "huffman.h"

void HuffMemFill(PVOID, USHORT, USHORT);

/* initialization of tree */

PHUFFSTUFF StartHuff(void)
{
    SHORT i, j;
    PHUFFSTUFF mypHS;
    HANDLE  hHS;

    hHS = GlobalAlloc(GMEM_MOVEABLE, sizeof(struct _HuffStuff));
    mypHS = (PHUFFSTUFF)GlobalLock(hHS);
   if (mypHS == NULL)
       return(NULL);

    _fmemset(mypHS, 0x00, sizeof(struct _HuffStuff));
   mypHS->textsize = mypHS->codesize = 0;
   mypHS->match_position = mypHS->match_length = 0;
   mypHS->getbuf = mypHS->putbuf = mypHS->code = mypHS->len = 0;
   mypHS->getlen = mypHS->putlen = 0;

   for (i = 0; i < N_CHAR; i++) {
      mypHS->freq[i] = 1;
      mypHS->son[i] = i + T;
      mypHS->prnt[i + T] = i;
   }
   i = 0; j = N_CHAR;
   while (j <= R) {
      mypHS->freq[j] = mypHS->freq[i] + mypHS->freq[i + 1];
      mypHS->son[j] = i;
      mypHS->prnt[i] = mypHS->prnt[i + 1] = j;
      i += 2; j++;
   }
   mypHS->freq[T] = 0xffff;
   mypHS->prnt[R] = 0;
    return(mypHS);
}

void EndHuff(PHUFFSTUFF pHS)
{
HANDLE  hHS;
    hHS = LOWORD(GlobalHandle(HIWORD((DWORD)(pHS))));
   while (GlobalFlags(hHS) & GMEM_LOCKCOUNT)
       GlobalUnlock(hHS);
   GlobalFree(hHS);
}

/* reconstruction of tree */

void ReConstructTree(PHUFFSTUFF pHS)
{
    SHORT i, j, k;
    USHORT f, l;
```

```
    /* collect leaf nodes in the first half of the table */
    /* and replace the freq by (freq + 1) / 2. */
    j = 0;
    for (i = 0; i < T; i++) {
        if (pHS->son[i] >= T) {
            pHS->freq[j] = (pHS->freq[i] + 1) / 2;
            pHS->son[j] = pHS->son[i];
            j++;
        }
    }
    /* begin constructing tree by connecting sons */
    for (i = 0, j = N_CHAR; j < T; i += 2, j++) {
        k = i + 1;
        f = pHS->freq[j] = pHS->freq[i] + pHS->freq[k];
        for (k = j - 1; f < pHS->freq[k]; k--);
        k++;
        l = (j - k) * 2;
        _fmemmove(&pHS->freq[k + 1], &pHS->freq[k], l);
        pHS->freq[k] = f;
        _fmemmove(&pHS->son[k + 1], &pHS->son[k], l);
        pHS->son[k] = i;
    }
    /* connect prnt */
    for (i = 0; i < T; i++) {
        if ((k = pHS->son[i]) >= T) {
            pHS->prnt[k] = i;
        } else {
            pHS->prnt[k] = pHS->prnt[k + 1] = i;
        }
    }
}

/* increment frequency of given code by one, and update tree */
void UpdateTree(PHUFFSTUFF pHS, SHORT c)
{
    SHORT i, j, l;
    USHORT k;

    if (pHS->freq[R] == MAX_FREQ) {
        ReConstructTree(pHS);
    }
    c = pHS->prnt[c + T];
    do {
        k = ++pHS->freq[c];

        /* if the order is disturbed, exchange nodes */
        if (k > pHS->freq[l = c + 1]) {
            while (k > pHS->freq[++l]);
            l--;
            pHS->freq[c] = pHS->freq[l];
            pHS->freq[l] = k;

            i = pHS->son[c];
            pHS->prnt[i] = l;
            if (i < T) pHS->prnt[i + 1] = l;

            j = pHS->son[l];
            pHS->son[l] = i;

            pHS->prnt[j] = c;
            if (j < T) pHS->prnt[j + 1] = c;
            pHS->son[c] = j;
```

```
        c = l;
    }
  } while ((c = pHS->prnt[c]) != 0); /* repeat up to root */
}

void InitTree(PHUFFSTUFF pHS)  /* initialize trees */
{
  SHORT  i;

  for (i = N + 1; i <= N + 256; i++)
    pHS->rson[i] = NIL;       /* root */
  for (i = 0; i < N; i++)
    pHS->dad[i] = NIL;        /* node */

}

void InsertNode(PHUFFSTUFF pHS, SHORT r)  /* insert to tree */
{
  SHORT  i, p, cmp;
  PCHAR  key;
  USHORT c;

  cmp = 1;
  key = &pHS->text_buf[r];
  p = N + 1  + key[0];
  pHS->rson[r] = pHS->lson[r] = NIL;
  pHS->match_length = 0;
  for ( ; ; ) {
    if (cmp >= 0) {
      if (pHS->rson[p] != NIL)
        p = pHS->rson[p];
      else {
        pHS->rson[p] = r;
        pHS->dad[r] = p;
        return;
      }
    } else {
      if (pHS->lson[p] != NIL)
        p = pHS->lson[p];
      else {
        pHS->lson[p] = r;
        pHS->dad[r] = p;
        return;
      }
    }
    for (i = 1; i < F; i++)
      if ((cmp = key[i] - pHS->text_buf[p + i]) != 0)
        break;
    if (i > THRESHOLD) {
      if (i > pHS->match_length) {
        pHS->match_position = ((r - p) & (N - 1)) - 1;
        if ((pHS->match_length = i) >= F)
          break;
      }
      if (i == pHS->match_length) {
        if ((c = ((r - p) & (N - 1)) - 1) < (USHORT)pHS->match_position) {
          pHS->match_position = c;
        }
      }
    }
  }
  pHS->dad[r] = pHS->dad[p];
  pHS->lson[r] = pHS->lson[p];
  pHS->rson[r] = pHS->rson[p];
```

```
    pHS->dad[pHS->lson[p]] = r;
    pHS->dad[pHS->rson[p]] = r;
    if (pHS->rson[pHS->dad[p]] == p)
        pHS->rson[pHS->dad[p]] = r;
    else
        pHS->lson[pHS->dad[p]] = r;
    pHS->dad[p] = NIL; /* remove p */
}

#pragma optimize("e",off)       /* Get around regMD, line 3837 bug */
                                /* in both C600 and C600AX      */
void DeleteNode(PHUFFSTUFF pHS, SHORT p)  /* remove from tree */
{
    SHORT  q;

    if (pHS->dad[p] == NIL)
        return;         /* not registered */
    if (pHS->rson[p] == NIL)
        q = pHS->lson[p];
    else
    if (pHS->lson[p] == NIL)
        q = pHS->rson[p];
    else {
        q = pHS->lson[p];
        if (pHS->rson[q] != NIL) {
            do {
                q = pHS->rson[q];
            } while (pHS->rson[q] != NIL);
            pHS->rson[pHS->dad[q]] = pHS->lson[q];
            pHS->dad[pHS->lson[q]] = pHS->dad[q];
            pHS->lson[q] = pHS->lson[p];
            pHS->dad[pHS->lson[p]] = q;
        }
        pHS->rson[q] = pHS->rson[p];
        pHS->dad[pHS->rson[p]] = q;
    }
    pHS->dad[q] = pHS->dad[p];
    if (pHS->rson[pHS->dad[p]] == p)
        pHS->rson[pHS->dad[p]] = q;
    else
        pHS->lson[pHS->dad[p]] = q;
    pHS->dad[p] = NIL;
}
```

```
#include <windows.h>

#include <stdlib.h>
#include <string.h>
#include <ctype.h>

#define INCLUDE_HUFFSTUFF
#include "huffman.h"

#define EOF -1

HANDLE  hDLLInst;

SHORT FAR PASCAL CompressBufferToBuffer(PCHAR pInput, ULONG ulInSize,
                                        PCHAR pOutput, ULONG ulOutSize,
                                        PULONG pulOutSize)
{
   SHORT rc;
   PHUFFSTUFF pHS;

   pHS = StartHuff();
   if (pHS == NULL)
      return(HUFFMAN_MEMORY_ERROR);
   pHS->sType = HUFFMAN_BUFFER_TO_BUFFER;
   pHS->pInput = (PVOID)pInput;
   pHS->pOutput = (PVOID)pOutput;
   pHS->fGetNextChar = GetNextBufferChar;
   pHS->fPutNextChar = PutNextBufferChar;
   pHS->ulMaxInput = ulInSize;
   pHS->ulMaxOutput = ulOutSize - 4;

   pHS->textsize = ulInSize;
   *((PCHAR)pHS->pOutput) = (UCHAR)((pHS->textsize & 0xff000000L) >> 24);
   *((PCHAR)pHS->pOutput+1) = (UCHAR)((pHS->textsize & 0xff0000L) >> 16);
   *((PCHAR)pHS->pOutput+2) = (UCHAR)((pHS->textsize & 0xff00) >> 8);
   *((PCHAR)pHS->pOutput+3) = (UCHAR)(pHS->textsize & 0xff);
   pHS->pOutput = (PVOID)((PCHAR)pHS->pOutput+4);
   if (pHS->textsize == 0)
      return(HUFFMAN_FILE_LENGTH_ERROR);

   rc = Compress(pHS);

   *pulOutSize  = ulOutSize - pHS->ulMaxOutput;

   EndHuff(pHS);
   return(rc);
}


SHORT FAR PASCAL ExpandBufferToBuffer(PCHAR pInput, ULONG ulInSize,
                                      PCHAR pOutput, ULONG ulOutSize,
                                      PULONG pulOutSize)
{
   SHORT rc;
   PHUFFSTUFF pHS;

   pHS = StartHuff();
   if (pHS == NULL)
      return(HUFFMAN_MEMORY_ERROR);
   pHS->sType = HUFFMAN_BUFFER_TO_BUFFER;
   pHS->pInput = (PVOID)pInput;
   pHS->pOutput = (PVOID)pOutput;
   pHS->fGetNextChar = GetNextBufferChar;
   pHS->fPutNextChar = PutNextBufferChar;
```

```
    pHS->ulMaxInput = ulInSize;
    pHS->ulMaxOutput = ulOutSize;

    if (pHS->ulMaxInput < 4)
        return(HUFFMAN_FILE_LENGTH_ERROR);
    pHS->textsize = (UCHAR)*((PCHAR)pHS->pInput);
    pHS->textsize <<= 8;
    pHS->textsize |= (UCHAR)*((PCHAR)pHS->pInput+1);
    pHS->textsize <<= 8;
    pHS->textsize |= (UCHAR)*((PCHAR)pHS->pInput+2);
    pHS->textsize <<= 8;
    pHS->textsize |= (UCHAR)*((PCHAR)pHS->pInput+3);
    pHS->pInput = (PVOID)((PCHAR)pHS->pInput+4);
    pHS->ulMaxInput -= 4;
    if (pHS->textsize == 0)
        return(HUFFMAN_FILE_LENGTH_ERROR);

    rc = Expand(pHS);

    *pulOutSize = pHS->textsize;

    EndHuff(pHS);

    return(rc);
}


SHORT GetNextBufferChar(PHUFFSTUFF pHS)
{
UCHAR c;

    if (!pHS->ulMaxInput)
        return(EOF);
    c = *(PCHAR)pHS->pInput;
    pHS->pInput = (PVOID)((PCHAR)pHS->pInput+1);
    pHS->ulMaxInput--;
    return(c);

}

SHORT PutNextBufferChar(PHUFFSTUFF pHS, USHORT c)
{

    if (!pHS->ulMaxOutput)
        return(EOF);
    *(PCHAR)(pHS->pOutput) = (char)c;
    pHS->pOutput = (PVOID)((PCHAR)pHS->pOutput+1);
    pHS->ulMaxOutput--;
    return(1);

}


int FAR PASCAL WEP (nParm)
int nParm;
{
    if (nParm == WEP_SYSTEM_EXIT){
        return(1);
        }

    if (nParm == WEP_FREE_DLL){
        return(1);
        }
}
```

```
/*
BOOL FAR PASCAL LibMain( hInstance, wDataSeg, wHeapSize, lpCmdLine)
HANDLE   hInstance;
WORD     wDataSeg, wHeapSize;
LPSTR    lpCmdLine;
{
int rc;
    hDLLInst = hInstance;
    if (wHeapSize == 0)
        return(0);
    rc =  LocalInit( wDataSeg, NULL, wHeapSize);
    return(rc);
}

*/
```