
**CENTRO DE INVESTIGACION Y DE ESTUDIOS AVANZADOS DEL INSTITUTO POLITECNICO NACIONAL**

**DEPARTAMENTO DE INGENIERIA ELECTRICA**

**SECCION DE COMPUTACION**


# HERRAMIENTAS DE COMUNICACIONES PARA MICROSOFT WINDOWS



*Tesis que presenta el Ing. David Solis Pacheco para obtener el grado de Maestro en Ciencias en la especialidad de Ingeniería Eléctrica.*

*Trabajo dirigido por el Doctor Manuel Guzmán Rentería.*

*México D.F. Junio de 1993*

# Contenido

## Parte 1 Introducción



## Parte 2 Implementación

Parte

# 1 Introducción

Capítulo

# 1 Introducción

Este capítulo inicia con una discusión breve de los sistemas operativos y los ambientes gráficos más populares, describe el objetivo de este trabajo, define las características de las herramientas desarrolladas, y lista el equipo usado para desarrollo, pruebas y documentación.

***Contenido de este capítulo***



## Antecedentes

El surgimiento de las PCs hace 10 años fue debido a la necesidad de un hardware estándar y en aquella época solo IBM podía ofrecerlo. En estos momentos se necesita una plataforma de software estándar, que proporcione servicios como memoria virtual y multitarea. Se debe tener una interfaz común con el usuario de tal manera que la computación pueda llegar a ser tan familiar para la gente como manejar un coche.

El sistema operativo DOS tiene muchas limitaciones como son el tamaño de memoria y la arcaica arquitectura segmentada impuesta por INTEL. En aplicaciones grandes es un problema el manejo adecuado de la memoria, el intercambio de datos entre aplicaciones DOS es prácticamente inexistente y cada aplicación tiene su propia interfaz con el usuario, existiendo miles de estas.

OS/2 tuvo la oportunidad de ser el sucesor de DOS, pero debido quizá a una mala mercadotecnia no obtuvo la aceptación esperada. UNIX tiene una gran oportunidad en estos momentos, sin embargo existe una gran desorganización en las compañías (el soporte técnico es malo, muchas aplicaciones tienen demasiados errores), además de ser poco amigable (la documentación es difícil de leer, los comandos son complicados). Sin embargo, UNIX ha sido ampliamente aceptado ya que es un sistema operativo multiusuario, multitarea y multiplataforma. Antes, una desventaja de UNIX era que no tenía una interfaz gráfica, pero ahora la tiene. Esta interfaz es XWINDOWS o simplemente X, la única desventaja es que las terminales X o las estaciones de trabajo son muy

costosas. Casi todos las compañías de Manejadores de Bases de Datos poseen una versión para UNIX.

La unión DOS-Windows, DOS como sistema operativo y Windows como ambiente gráfico, ha tenido una gran aceptación. Windows proporciona una interfaz común con el usuario, de tal manera que diversas aplicaciones funcionan de manera semejante. El intercambio de datos entre aplicaciones (Clipboard, DDE, OLE) es muy fácil (para usarlo, no para programarlo). La estrategia de Microsoft en estos momentos está basada en Windows. No obstante, Windows no es la solución final; probablemente Windows NT (New Technology) lo sea, al menos asi lo espera Microsoft.

NT es un nuevo sistema operativo que soporta múltiples y diferentes subsistemas. La primera versión soportará un subsistema para aplicaciones DOS, uno para aplicaciones Windows de 16 y 32 bits y otro para aplicaciones POSIX (una variante de UNIX). La habilidad para soportar múltiples subsistemas permitirá a una futura versión de NT, soportar aplicaciones OS/2. NT es un sistema multitarea que opera bajo arquitecturas basadas en procesadores 80386-80486 y MIPS RISC.

Algunas personas pronostican que NT remplazará a Windows 3.1 y a Windows for Workgroups, mientras que otros (más realistas) aseguran que NT tomará de un 15 a un 20 por ciento del mercado de Windows 3.1.

## Objetivo

El objetivo de este trabajo fue realizar un conjunto de herramientas de comunicación para Windows que permitan a una PC obtener servicios que se encuentran en minicomputadoras y macrocomputadoras.

Las herramientas consisten de un emulador de terminal con transferencia de archivos que usan la interfaz gráfica de Windows.

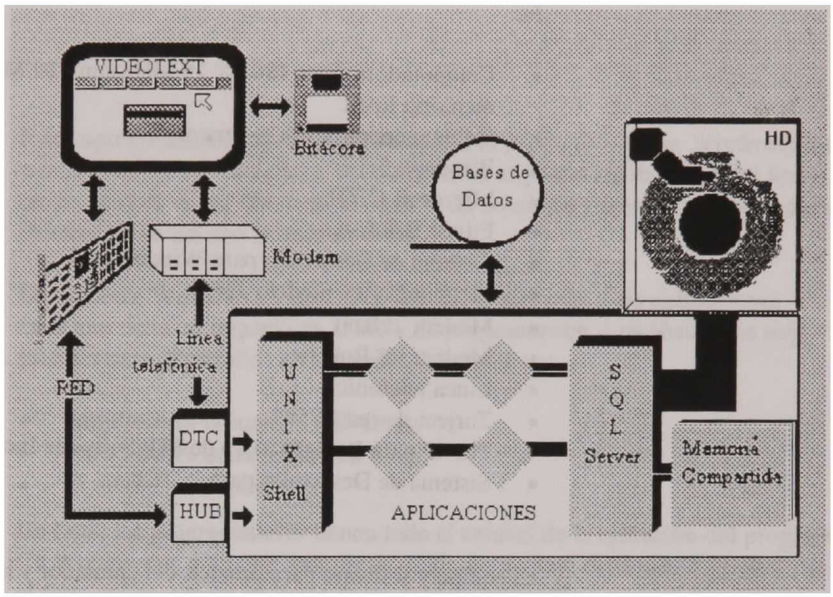
Los programas que se ejecutan en la computadora remota usan las interfaces y herramientas proporcionadas por el sistema operativo. Estos programas pueden ser ejecutados en una terminal tonta.

Estas herramientas pueden ser usadas en diversas aplicaciones, por ejemplo, como el componente para la conexión vía línea conmutada o red con una minicomputadora con sistema operativo UNIX donde reside un Sistema de Información de las bases de datos de una empresa.

La arquitectura del sistema mencionado se muestra en la Figura 1.1.

*Figura 1.1*

*Las herramientas pueden ser usadas para accesar desde una PC información y recursos que se encuentran en una minicomputadora*




## Características

Las características que se tomaron en cuenta para el diseño de las herramientas de comunicación fueron las siguientes:

- Aplicación diseñada especialmente para Windows 3.1, y que no funcionará sin él o con versiones anteriores.
- Operación sobre línea directa, conmutada o redes con protocolo TCP/IP.
- Conexión automática a computadoras con sistema operativo UNIX. (El usuario no tiene que teclear su clave de usuario ni su password).
- Configurable a cualquier tipo de modem.
- Con un control que evite oprimir teclas de uso frecuente como el escape, el retorno, las teclas de movimiento de cursor, etc.
- Emulación de una terminal ANSI (Una PC con ANSI.SYS).
- Archivo de configuración para que el usuario configure el programa y guarde la configuración.
- Con un protocolo para transferencia de archivos.
- Impresión del contenido de la pantalla.

## Equipo Utilizado

El equipo y software para desarrollo y prueba de los programas en Windows fue.

- Computadora Acer 486 SX 25 MHz con 120 Mb de disco duro y 4 Mb de memoria RAM.
- Sistema operativo DOS versión 5.0.
- Windows 3.1.
- Microsoft C 7.0.
- Editor Brief
- Sistema de Desarrollo para Windows 3.1.
- Modem Codex modelo 3266.
- Modem Telebit
- Modem US Robotics
- Línea telefónica
- Tarjeta de red 3C507
- TCP/IP para DOS (PCTCP) de FTP Software Inc.
- Sistema de Desarrollo para PCTCP.

El equipo y software para pruebas del emulador y la transferencia de archivos fue:

- Workstation Apollo modelo 725.
- Sistema operativo HP-UX versión 8.02.
- Modem Codex modelo 3268.
- Línea telefónica

El equipo y software para la elaboración de la documentación fue:

- Computadora AMI 486 DX2 66 MHz con 130 Mb de disco duro y 8 Mb de memoria RAM.
- Windows 3.1
- Microsoft Word for Windows 2.0a
- Editor gráfico Microsoft PaintBrush
- Procesador de imágenes Paint Shop Pro

*Capítulo*

# 2 El Ambiente de Programación Windows

Este capítulo discute tópicos relacionados a la programación en Windows: los módulos de las aplicaciones y las librerías, la arquitectura de Windows desde el punto de vista de las librerías, el sistema de mensajes y conceptos orientados a objetos que se aplican en este entorno.

El ambiente Windows proporciona diferentes servicios a las aplicaciones, y cada uno tiene un impacto en el ciclo de programación. Los obstáculos más grandes para desarrollar aplicaciones Windows son:

- Arquitectura Orientada a Eventos
- Poca y pobre documentación
- El ambiente Windows

En DOS, los programadores tienen todo el control de la ejecución del programa. En Windows es diferente, porque es usado un modelo orientado a eventos para el flujo del programa en lugar de un modelo orientado a procedimientos.

El como y cuando ciertos mensajes son enviados, produce confusión al desarrollador. De la misma manera que DOS, Windows usa demasiadas funciones indocumentadas y mensajes que son restringidos al programador.

El desarrollo exitoso de aplicaciones Windows requiere que el programador entienda el ambiente de una manera completa. Este capítulo tiene como objetivo proporcionar una mejor comprensión de los conceptos básicos del diseño de Windows y su sistema de mensajes. La última sección, Conceptos Orientados a Objetos en Windows, discute brevemente algunos principios orientados a objetos y su relación con el ambiente gráfico Windows.

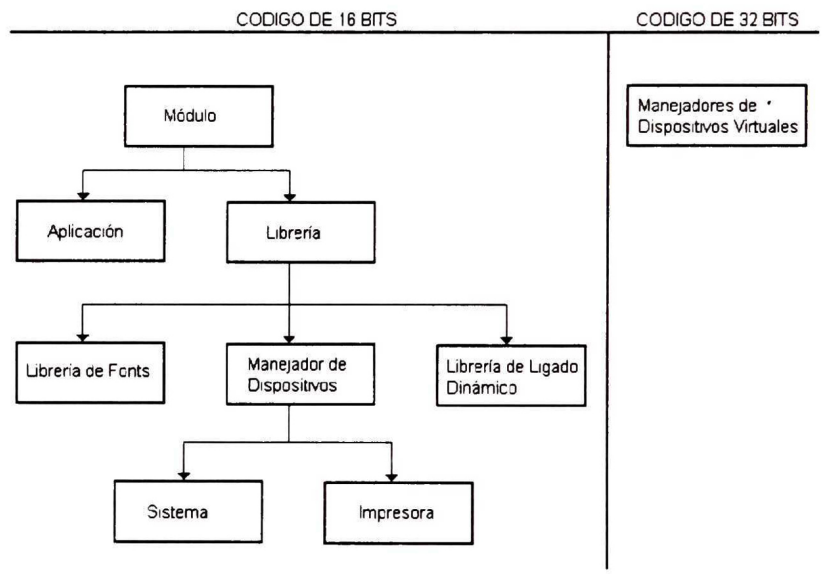
***Contenido de este capítulo***

## Módulos de las Aplicaciones y Librerías

En Windows existen dos unidades de programas fundamentales (Ver Figura 2.1): aplicaciones (archivos ejecutables con extensión .exe) y librerías de ligado dinámico o DLLs (archivos con extensión .dll, .drv, .exe y .fon).

*Figura 2.1*

*Unidades fundamentales de una aplicación Windows*



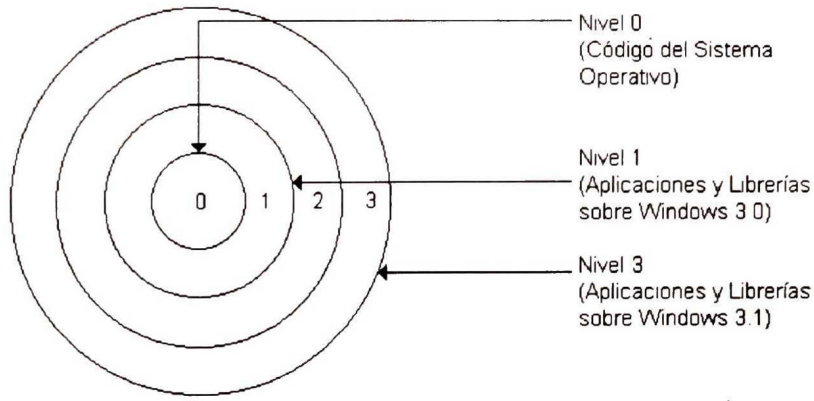

Las aplicaciones se diferencian de las DLLs en que varias copias o instancias de una aplicación pueden estar corriendo. Cada copia de la aplicación es referenciada por un identificador de instancia. Las DLLs, las cuales son cargadas en memoria una sola vez, tienen solo un identificador de instancia. El identificador de instancia es usado por Windows como un identificador al segmento de datos del programa. La razón es simple: Windows comparte segmentos de código y de recursos y el segmento de datos es el único segmento que puede ser usado para diferenciar las instancias de un módulo.

En Windows 3.1, todas las aplicaciones y librerías corren en el nivel 3 (Ver Figura 2.2). En Windows 3.0 todos los programas de usuario corren en el nivel 1. En las dos versiones, el nivel 0 esta reservado para el manejador de memoria virtual de Windows y manejadores de dispositivos virtuales. Los programas que

corren en el nivel 0 tienen acceso a toda la memoria y a los recursos del sistema.

*Figura 2.2*

*Niveles de protección en Windows*




La relación módulo/instancia y clase/ventana son similares, porque hay múltiples instancias de un módulo (solo en aplicaciones) o múltiples ventanas de una cierta clase.

## La Relación Instancia/Módulo

La primera vez que una aplicación o librería es cargada en memoria, Windows crea una estructura de datos conocida como base de datos de módulos. La base de datos de módulos, al igual que otros segmentos de programas, es almacenado en un área global de memoria conocida como heap.

La base de datos de módulos contiene información que se encuentra en el encabezado de archivos ejecutables de aplicaciones y DLLs. Este encabezado contiene mucha información referente a todas las funciones del programa que se encuentran en DLLs (funciones que se exportan), recursos (diálogos, mapas de bits, iconos, fonts).

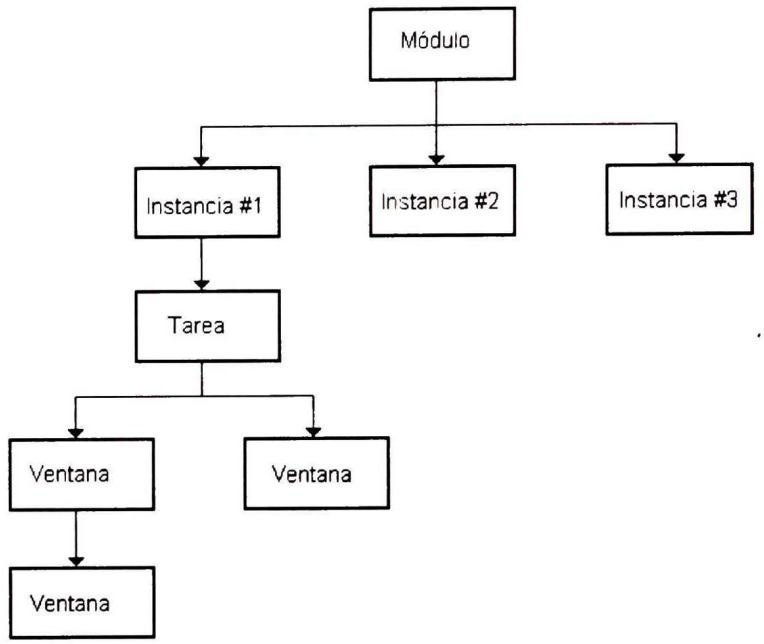
Con cada instancia de una aplicación, Windows también asocia una tarea, la que es referenciada por un identificador de tareas. Este último apunta a la base de datos de la tarea de la aplicación conocida como TDB (task database). La TDB contiene información de la instancia como: la tabla de identificadores de archivos de DOS, la trayectoria de DOS (path) y un apuntador a la cola de mensajes de la aplicación.

Las DLLs no tienen una TDB asociada, esta es la razón por la que se asocian todas las ventanas creadas por la librería con la tarea activa (siempre es una aplicación). Los términos tarea y proceso son sinónimos en Windows 3.x.

Tareas, instancias, módulos e identificadores de ventanas pueden ser representados gráficamente por medio de un árbol (Figura 2.3).

*Figura 2.3*

*La relación entre tarea, instancia, módulo y ventana*




Instancia = Proceso
Tarea = Entidad ejecutable (Existe una por aplicación en Windows)

## Aplicaciones

Las aplicaciones son llamadas tareas ejecutables, porque cada aplicación en Windows es manejada por una sola tarea y responde a una serie de eventos externos (mensajes). Actualmente existe una tarea por instancia de una aplicación, pero en las siguientes versiones de Windows se podrán tener varias tareas por aplicación. A esta característica se le conoce como multithreaded.

Las aplicaciones en Windows contienen un pequeño código de inicialización y a continuación son manejados por mensajes. Todas las aplicaciones obtienen el control del procesador a través de su función WinMain. Esta última tiene algunas responsabilidades. La primera acción que se tiene que hacer en WinMain es llamar al código de inicialización del módulo porque múltiples instancias de una aplicación pueden estar en ejecución. Este código se tiene que

ejecutar una sola vez. Básicamente es usado para registrar las clases de las ventanas usadas por la aplicación e inicializar variables globales.

La segunda acción de WinMain es crear las ventanas de la aplicación. WinMain es llamada por Windows con 4 parámetros: el identificador de la instancia de la aplicación, el identificador de la instancia de otra copia de la aplicación (si es que existe), un apuntador a la línea de comandos (nombre de la aplicación y argumentos), y un valor entero que indica como debería ser mostrada la aplicación (minimizada, normal o maximizada).

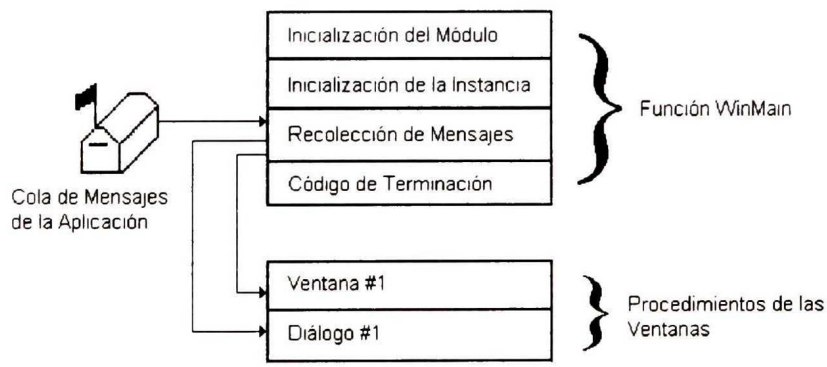
En el momento en el que el programa ha creado al menos una ventana, el control del procesador es transferido a WinMain y de esta manera se ejecuta el código que procesa y despacha mensajes. Este código generalmente se implementa como un ciclo (con un estatuto while) y es conocido como ciclo de recolección de mensajes o ciclo de mensajes.

El ciclo de mensajes es el corazón de todas las aplicaciones, porque su tarea es enviar mensajes a todas las ventanas de la aplicación. No existe un límite en el número de ventanas que las aplicaciones pueden crear, mantener y destruir. Sin embargo el límite del sistema es de 500 a 700 ventanas.

En el momento en que una aplicación llama a la función PostQuitMessage, Windows envía un mensaje WM_QUIT a la aplicación, y esta acción hace que se salga del ciclo de mensajes. En este punto, se puede ejecutar código de terminación. A continuación la aplicación finaliza. La Figura 2.4 muestra el flujo de mensajes.

*Figura 2.4*

*Flujo de los mensajes en una aplicación Windows*




## Librerías de Ligado Dinámico

Las DLLs no tienen asociada una tarea propia. Tampoco tienen una pila, ciclo de mensajes o cola de mensajes, porque usan los de la aplicación que las

invocó. El propósito principal de las DLLs es proporcionar funciones y recursos. Una DLL no es liberado de la memoria hasta que todos los módulos que la usan hayan sido a su vez liberados de la memoria.

Las DLLs son usadas para los siguientes propósitos:

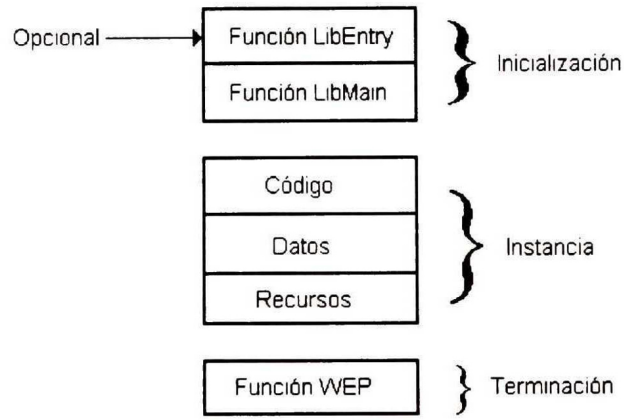
- Para registrar y almacenar clases globales de ventanas.
- Creación de manejadores de dispositivos y rutinas de servicio de interrupciones.
- Almacenamiento de grandes cantidades de recursos (mapas de bits, iconos, diálogos).
- Código para interceptar y filtrar mensajes.
- Proporcionar una libería de funciones

Las DLLs proporcionan un alto nivel de abstracción. Las DLLs son usadas para almacenar código común a varias aplicaciones. A este concepto se le conoce como código reusable y es uno de los mayores beneficios de la aplicación de análisis y diseño orientado a objetos.

La primera vez que una DLL es cargada en memoria, Windows llama a la función LibEntry. La tarea de LibEntry es (posiblemente) inicializar datos de la DLL y llamar a la función LibMain. En esta última función es donde normalmente se lleva a cabo todo el código de inicialización. Cuando LibMain concluye su trabajo y regresa a LibEntry, el control del procesador regresa a Windows y la librería puede ser accesada. La DLL no obtiene el control del procesador hasta que un módulo la llama o hasta que es liberada de la memoria. En el segundo caso, Windows llama a la función WEP de la DLL, donde se realiza el código de terminación. La Figura 2.5 muestra la estructura de una DLL.

*Figura 2.5*

*Inicialización, funciones y terminación de una librería de ligado dinámico*



## Librerías de Windows

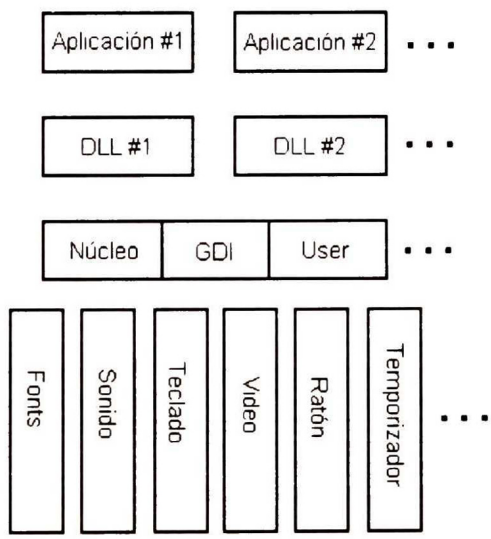
En la inicialización de Windows, algunos manejadores de dispositivos, fonts y librerías del sistema operativo son cargadas en memoria, como se muestra a continuación.

| *DLL* | *Propósito* |
|---|---|
| comm.drv | Comunicación serie |
| display.drv | Video |
| keyboard.drv | Teclado |
| mouse.drv | Ratón |
| sound.drv | Sonido |
| system.drv | Temporizador y coprocesador matemático |
| gdi.exe | Interfaz con dispositivos gráficos |
| krnlx86.exe | Multitarea, memoria y manejo de recursos (se le conoce como núcleo) |
| user.exe | Manejo de ventanas |
| TipoDeFont.fon | Fonts |

Estas librerías proporcionan servicios a las aplicaciones y a otros módulos. Nuevos servicios tales como extensiones de multimedia, extensiones a OLE y DDE, consisten de archivos con extensiones .drv y .dll y reemplazos a los manejadores de dispositivos previamente mencionados. Es posible crear un medio ambiente completamente diferente, por ejemplo que las ventanas tengan una apariencia tipo MOTIF. La Figura 2.6 muestra la arquitectura simplificada de Windows.

*Figura 2.6*

*Arquitectura simplificada de Windows*



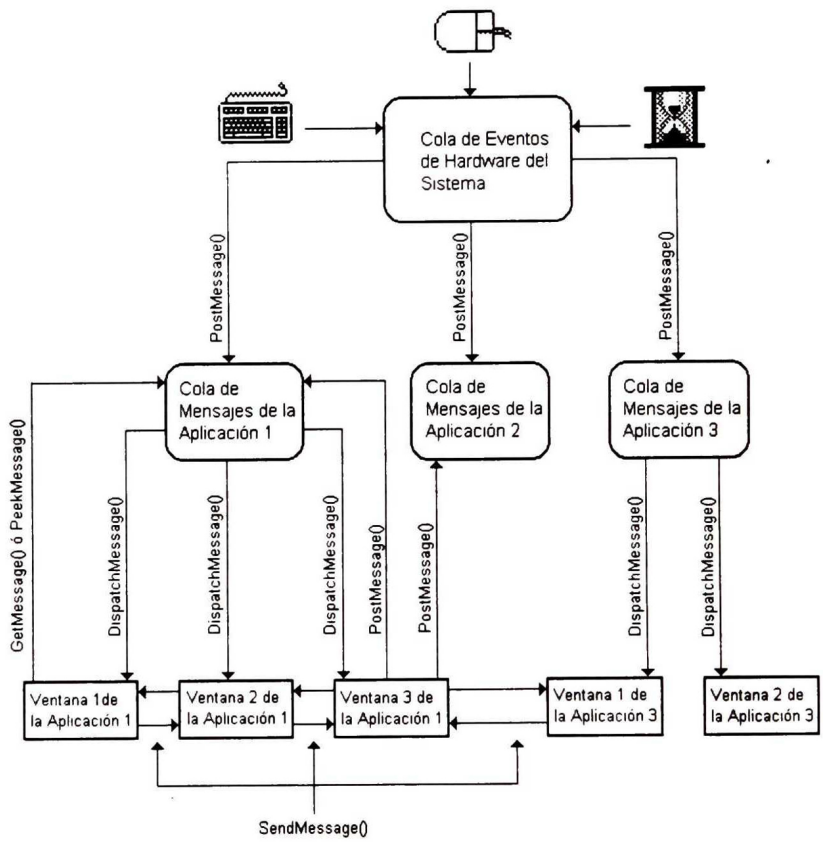
## El Sistema de Mensajes de Windows

Los mensajes son el corazón de cualquier sistema orientado a objetos. Un objeto necesita un constante flujo de mensajes.

Existen dos maneras de enviar mensajes a una ventana en Windows. La primera es por medio de la función PostMessage. Esta última pone el mensaje en la cola de mensajes de la aplicación y retorna inmediatamente. La segunda es a través de la función SendMessage. SendMessage llama al procedimiento que procesa los mensajes de la ventana y no retorna hasta que el mensaje haya sido procesado.

*Figura 2.7*

*Generación y adquisición de mensajes de bajo nivel en Windows*




Como se muestra en la Figura 2.7, Windows genera y adquiere mensajes de diferentes fuentes. En el nivel más bajo se encuentran los mensajes generados por el teclado, ratón y el temporizador. Estos mensajes son guardados en la cola de mensajes del sistema.

Los mensajes son generados también por diferentes funciones del API de Windows. Los mensajes pueden ser generados por las aplicaciones. Una aplicación o librería puede poner mensajes en su cola (la de la aplicación) o en la cola de otra aplicación, y puede enviar mensajes directamente a su(s) ventana(s) o a la(s) ventana(s) de otra aplicación.

Una aplicación o librería puede leer y opcionalmente remover mensajes de su cola. Para realizar esta acción se requiere llamar a una función que intercepte mensajes.

## Funciones de la Ventana

Todas las ventanas tienen por lo menos dos funciones que procesan sus mensajes: una función privada, la cual es notificada a la función RegisterClass, y una función común (la función DefWindowProc o DefDlgProc).

Todas las funciones que procesan mensajes de la ventana deben ser declaradas FAR, porque Windows necesita cambiar los segmentos de código cuando las invoca. Estas funciones son también declaradas PASCAL porque esta convención es la usada por Windows y fue escogida porque reduce el tamaño del código considerablemente. La siguiente línea muestra la declaración de estas funciones:

```
ValorDeRetorno FAR PASCAL WndProcName(HWND hWnd, UINT Msg, WPARAM wParam,
LPARAM lParam);
```

Estas funciones pueden retornar diferentes tipos de datos. La siguiente lista muestra los tipos más comunes.

| *Valor de Retorno* | *Uso* |
|---|---|
| LRESULT | Para ventanas y diálogos creados con una librería de terceros. |
| int | Para diálogos que no permiten activar a otras ventanas de la aplicación (conocidos como modal). |
| BOOL | Para diálogos que permiten activar a otras ventanas de la aplicación (conocidos como modeless). |

## Mensajes

Un mensaje que es recibido por la función que procesa mensajes de una ventana, es dividido en cuatro partes: un identificador de ventana (hWnd) que especifica el destino del mensaje, un valor entero sin signo (Msg) que representa el mensaje, y dos parámetros adicionales (wParam y lParam) que contienen información adicional al mensaje. El valor de wParam es usado

como un índice, una bandera, o como el identificador de una ventana, mientras lParam es usado como un apuntador a una función o a una cadena, o como un valor que contiene un código de notificación y un identificador de ventana. Generalmente el mensaje es procesado en un estatuto switch en la función que procesa mensajes de la ventana, donde cada estuto case actúa sobre un mensaje diferente.

El código siguiente muestra una función que procesa mensajes de una ventana:

```
LRESULT FAR PASCAL MainWndProc(HWND hWnd, UINT Msg, WPARAM wParam, LPARAM
    lParam)
{
    switch(Msg) {
        case WM_CREATE:
                // Opcionalmente se ejecuta código de inicialización
                break;
        case WM_CLOSE:
                DestroyWindow(hWnd);
                return(0L);
        case WM_DESTROY:
                PostQuitMessage(0);
                break;
        case WM_COMMAND:
                switch(wParam) {
                        // Procesa mensajes de menús
                }
                break;
        default:
                break;
    }
    // Procesa mensajes por omisión
    return(DefWindowProc(hWnd, Msg, wParam, lParam);
}
```

Un mensaje es definido por la estructura MSG. La siguiente declaración fue extraída del archivo WINDOWS.H.

```
typedef struct tagMSG
{
  HWND      hwnd;    // Ventana que recibe el mensaje
  UINT      message; // Código del mensaje
  WPARAM    wParam;  // Parámetro asociado con el mensaje
  LPARAM    lParam;  // Parámetro asociado con el mensaje
  DWORD     time;    // Hora en el que el mensaje fue depositado
  POINT     pt;      // Posición del cursor cuando el mensaje fue depositado
} MSG;
```

## *Procesamiento de Eventos en la Cola*

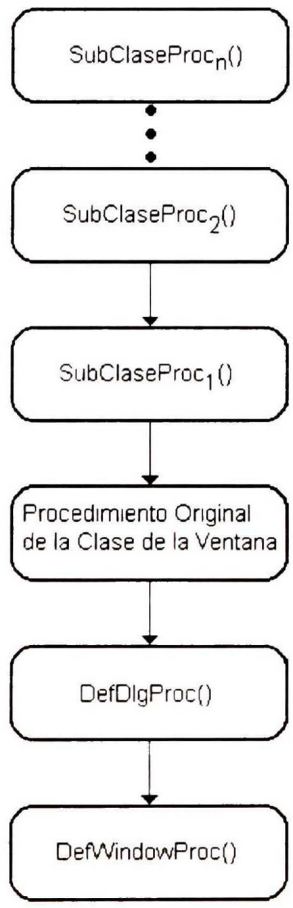
Los mensajes pueden ser procesados internamente, pueden ser ignorados (transferidos al procedimiento que tienen asociadas las ventanas por omisión), y pueden ser interceptados.

Existen técnicas para alterar el comportamiento (el procesamiento por omisión) de un mensaje. Una de ellas consiste en llamar al procedimiento por omisión primero y a continuación realizar los procedimientos que proporcionen el comportamiento esperado. A este técnica se le conoce como subclase y es de uso muy común cuando se programa en lenguajes orientados a objetos.

Como y cuando retornar el control a Windows o al procedimiento que procesa los mensajes de la ventana es muy importante. La respuesta es compleja, porque depende del tipo de mensaje que está siendo procesado y a que tipo de objeto pertenece (ventana, diálogo, botón, ventana de lista, etc.). Muchas capas de código pueden estar involucradas en el proceso, como se muestra en la Figura 2.8.

*Figura 2.8*

*Capas involucradas en el procesamiento de mensajes*



La acción de transferir los mensajes a las capas inferiores es conocido como herencia. Si no se procesa un mensaje, las otras capas heredan la capacidad de procesarlo y de esta manera se modifica el comportamiento de las ventanas.

Dependiendo del mensaje que se necesita procesar y del tipo de ventana (la discusión se centra en controles y diálogos), los APIs de Windows proporcionan las siguientes funciones para el manejo del comportamiento por omisión de las ventanas:

| *Función por omisión* | *Uso* |
|---|---|
| CallWindowProc | Llamada para transferir el procesamiento de un mensaje a una capa inferior en la cadena. La capa inferior deberá llamar a DefWindowProc o DlgDlfProc. |
| DefDlgProc | Proporciona el procesamiento por omisión para los diálogos. |
| DefWindowProc | Proporciona el procesamiento por omisión para las ventanas. |

## Depósito y Envío de Mensajes

Existen dos formas de mandar mensajes a una ventana: por depósito (posting) o por envío (sending). Una aplicación recibe mensajes depositados de una manera indirecta, esto es por medio de su cola de mensajes.

Cuando una aplicación deposita un mensaje, un valor, que indica el resultado de esta acción, es retornado inmediatamente. El depósito de mensajes garantiza que la tarea activa no será interrumpida, porque el control es devuelto inmediatamente después de poner el mensaje en la cola.

En el envío de un mensaje se llama directamente a la función que procesa los mensajes de la ventana destino en lugar de depositar el mensaje en la cola de mensajes de la aplicación.

PostMessage es reentrante, y es la única función que puede ser usada por una rutina de servicio de una interrupción para notificar que se necesita procesar eventos.

No todos los mensajes que Windows genera para una aplicación son depositados. En general, mensajes de entrada tales como teclado, ratón y temporizador son depositados, mientras que mensajes relacionados con el manejo de la ventana tales como WM_PAINT y WM_ERASEBKGND son siempre enviados.

Cada aplicación Windows tiene un ciclo usado para la recolección de mensajes de la cola. El siguiente código ilustra una manera básica de recolección de mensajes:

```
while(GetMessage(&msg, 0, 0, 0) {
    TranslateMessage(&msg);
    DispacthMessage(&msg);
}
```

Este ciclo se ejecuta continuamente. Cada iteración representa la extracción de un mensaje de la cola del sistema o de un mensaje de la cola de la aplicación. Con la excepción del mensaje WM_QUIT, GetMessage regresa un valor de TRUE. Cuando se recibe el mensaje WM_QUIT, el programa debe salirse del ciclo y terminar. La función TranslateMessage llama al manejador de teclado para convertir mensajes WM_KEYDOWN (código de la tecla) a WM_CHAR (valores ASCII ). La función DispatchMessage llama a la función que procesa los mensajes de la ventana apropiada.

## Multitarea a través de Mensajes

Una tarea cede el control del procesador a Windows por medio de 4 funciones:

- GetMessage
- PeekMessage
- WaitMessage
- Yield

En el momento en que alguna de estas 4 funciones es llamada, el control del procesador es devuelto a Windows, y los mensajes de otras aplicaciones pueden ser atendidos. Las siguientes funciones también ceden el control a Windows:

- DialogBox
- DialogBoxIndirect
- DialogBoxParam
- DialogBoxIndirectParam
- MessageBox

La función GetMessage devuelve el control a Windows y regresa a la tarea que la llamó solo cuando hay mensajes en la cola. GetMessage es comúnmente usada dentro de la función WinMain en la sección de código usada para recolectar mensajes. El comportamiento anteriormente descrito no es deseable en algunas ocasiones, tales como: procesamiento de eventos relacionados con periféricos (puerto serie, jostick, tarjetas de red, etc.), en protocolos para transferencias de archivos, procesamiento de algoritmos que consumen demasiado tiempo, etc. En estos casos, se debería usar PeekMessage. Esta última función monitorea la cola de mensajes de la aplicación y en caso de que existan mensajes, los extrae y opcionalmente los puede remover de la cola. A continuación el mensaje puede ser procesado o ignorado. A través de una

bandera se le puede indicar a PeekMessage que no devuelva el control a Windows.

La función WaitMessage es similar a GetMessage en que espera que un evento ingresa a la cola antes de regresar el control. Sin embargo, WaitMessage no borra el mensaje de la cola. La función Yield devuelve el control a Windows cuando no existen mensajes en la cola de la aplicación.

El código siguiente ilustra una manera más sofisticada de recolección de mensajes que permite monitorear periféricos o realizar labores que consumen demasiado tiempo.

```
while(TRUE) {
    while(PeekMessage(&msg, NULL, NULL, NULL, PM_REMOVE)) {
        if(msg.message == WM_QUIT) break;
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }
    if(msg.message == WM_QUIT) break;
    * Realiza código para monitoreo */
}
```

## Conceptos Orientados a Objetos en Windows

Windows ofrece pocas extensiones orientadas a objetos y esta es la razón por la que no cualquier aplicación Windows es "orientada a objetos". No existe una especificación que detalle cuales son las características para clasificar a una aplicación como orientada a objetos.

## Clases y Objetos

El uso de clases y objetos es uno de los principios de programación orientada a objetos. El beneficio de clases y objetos es que a través de ellos el desarrollador define los componentes de software necesarios para resolver un problema del mundo real. El análisis y el diseño con objetos y clases es muy similar a la manera usual como son resueltos los problemas. Un tópico es dividido en partes más pequeñas. Cada una de estas partes, es posiblemente divididas de nuevo en más objetos. En algún punto el proceso de clasificación termina y a cada objeto le es asignado un comportamiento.

Windows soporta solo objetos ventana - esto es, objetos que son visibles. Algunas veces esto no es deseable, porque no hay manera de supervisar objetos que no tienen un componente ventana.

Windows proporciona soporte para objetos ventana y clases para crear estos objetos, sin embargo el soporte no es tan completo como debería ser. En algunos casos es primitivo. En Windows, una clase de ventana no existe como una unidad independiente hasta que al menos una ventana de esta clases haya sido creada. Las clases no pueden enviar mensajes o ser manipuladas de la misma manera que las ventanas. Cuando los atributos de una ventana necesitan ser modificados, un mensaje es enviado directamente a la ventana. Cuando los atributos de una clase de ventana necesitan ser leídos o modificados, es necesario pasar un indicador de ventana a la función. Esto es porque Windows asocia un identificador de ventana con un nombre, y no con la clase de la ventana.

## Herencia

La herencia define una relación entre clases, donde una clase comparte la estructura o comportamiento definidos en una o más clases.

Windows proporciona herencia por medio de dos conceptos: subclases y superclases.

Subclase es cuando el flujo de mensajes enviado a una ventana es interceptado y reenrutado. Existen dos tipos de subclases disponibles al desarrollador: subclases globales y subclases relativas a la instancia. Ambas son similares, solo difieren en el ámbito y en la manera en que la función que procesa los mensajes es cambiada. La siguiente lista muestra el tipo de subclases, el ámbito y la función que es usada.

| *Tipo de Subclase* | *Ambito* | *Función* |
|---|---|---|
| Global | Clase | SetClassLong |
| Relativa a la instancia | Ventana | SetWindowLong |

Una superclase es cuando una nueva clase de ventana es creada y contiene el comportamiento de una clase existente, como podría ser una ventana de lista (listbox) o una ventana de edición. Esta técnica involucra a las funciones GetClassInfo y RegisterClass. La primera función es usada para obtener los campos de la estructura de la clase de la ventana, la cuál será modificada, y la estructura resultante será pasada como parámetro a RegisterClass.

## Encapsulación y Abstracción de Datos

Una abstracción indica las características esenciales de un objeto que lo distinguen de otros tipos de objetos y así proporciona límites definidos. Una abstracción sirve para separar el comportamiento esencial de un objeto de su implementación.

La encapsulación es el proceso de ocultar todos los detalles de un objeto que no son parte de sus características esenciales.

La encapsulación y abstracción de datos son conceptos claves para proyectos grandes y automáticamente ayudan al desarrollador a escribir código reentrante. Windows proporciona métodos simples para incorporar técnicas de encapsulación y abstracción, incluyendo el uso de librerías de ligado dinámico, reservaciones de memoria local, variables locales, clases y ventanas, y propiedades de ventanas.

Parte

# 2 Implementación

*Capítulo*

# 3 Emulador de Terminal

El objetivo de este capítulo es mostrar la construcción de un programa que permite que una PC emule una terminal y de esta manera haga posible la comunicación con macrocomputadoras y minicomputadoras a través de líneas conmutadas y líneas directas.

***Contenido de este capítulo***



## Antecedentes

Alguien podría preguntarse ¿Por qué hacer un programa que haga que un equipo con capacidad de procesamiento se comporte como un equipo sin esa capacidad?. La respuesta es que se obtienen servicios en la PC que no puede proporcionar ésta. Servicios como correo electrónico, consulta a bases de datos grandes, mayor capacidad de procesamiento, etc.

El emulador de terminal es un programa en tiempo real, es decir, es un programa que debe atender a dos dispositivos de entrada: el teclado y el puerto serie. Estos dispositivos operan en tiempo real y ninguno de ellos está sincronizado al programa. Cuando un carácter llega del puerto serie, el

emulador de terminal lo debe leer y procesarlo o almacenarlo para futuro procesamiento. De igual manera el emulador de terminal debe leer los caracteres que llegan del teclado. En algún momento el emulador de terminal debe procesar los caracteres almacenados y desplegarlos en la pantalla.

Los programas de comunicaciones en Windows son aplicaciones que consumen muchos recursos del sistema. Estas aplicaciones tienen que atender varias tareas como por ejemplo: deben de monitorear el puerto de comunicaciones, dar el control a otras aplicaciones, ejecutar interactivamente el código que maneja la interfaz del usuario mientras se atiende los eventos relacionados con la comunicación, usar la memoria virtual y accesar el disco a la vez que se atiende al puerto serie. Como resultado de procesar todas estas tareas, a una velocidad de transmisión mayor o igual a 9600 bauds, las aplicaciones en Windows se ejecutan más lentamente que las aplicaciones en DOS.

La ventaja de los servicios de comunicación en Windows es que proporcionan de una manera portable, el control del puerto de comunicación y la entrada/salida a través de colas.

Poca información ha sido publicado acerca de la interfaz de comunicaciones de Windows. En este programa se usó el sistema de desarrollo para Windows versión 3.1. La cual fue liberada al público en abril de 1992 , encontrándose la documentación exclusivamente en los manuales de referencia.

En esta versión se mejoró el manejador de la comunicación s e. Entre las mejoras se encuentra el uso de la cola (FIFO) de la pastilla electrónica 16550, este último es una versión mejorada del viejo 8250 y del no muy nuevo 16450. Otra mejora es que las aplicaciones Windows 3.1 son notificadas por mensajes cuando ciertos eventos relacionados con la comunicación serie ocurren, de esta manera estas aplicaciones no tienen que monitorear constantemente el puerto serie como ocurría en la versión 3.0.

## Manejo de Botones

Un programa típico de Windows contiene varios diálogos, generalmente a éstos se encuentran asociados botones de comandos (pushbutton), que sirven para cancelar o aceptar determinada opción. Los más frecuentes son "OK" (Aceptar) o "Cancel" (Cancelar). Estos diálogos son construidos usando la función MessageBox, que a su vez construye el diálogo y los botones, y regresa los resultados de la acción del usuario al programa.

Otra manera de usar botones es incluirlos en los diálogos. A cada uno de ellos se les asigna un identificador único para que el programa pueda reconocerlos en la función que procesa los mensajes del diálogo.

En estos dos casos, Windows crea los botones. En el caso del llamado a la función MessageBox, los botones son creados, manejados y destruidos de una

manera transparente. En el caso del diálogo, los botones son creados por la llamada a la función DialogBox (o las variaciones de ella), los mensajes del botón son manejados por la rutina del diálogo y son destruidos por la llamada a EndDialog.

Los botones son muy fáciles de usar porque Windows hace la mayor parte del trabajo. La inflexibilidad es el pago por esta facilidad de uso. Si se quiere un botón con un comportamiento más sofisticado o con diferentes atributos de texto, es necesario programarlos.

Para crear y usar un botón se requieren tres pasos:

1. Crear el botón.
2. Desplegar el botón.
3. Monitorear los mensajes del botón y realizar alguna acción cuando se reciba un mensaje.

La creación de un botón es a través de la llamada a la función CreateWindow, como se muestra a continuación:

```
ButtonHandle = CreateWindow(
"BUTTON",                    * Clase *
"Aceptar",                   /* Título del botón */
BS_PUSHBUTTON |              /* Estilos de la ventana *
WS_CHILD |
WS_VISIBLE,
xOffset,                     * Posición de la coordenada x */
yOffset,                     * Posición de la coordenada y */
btnWidth,                    * Ancho de la ventana */
btnHeigth,                   * Altura de la ventana */
hWnd,                        * Identificador de la ventana padre */
BUTTON_ID,                   /* Identificador de la ventana */
hInst,                       /* Instancia */
NULL
);
```

El segundo paso es automático. El botón fue creado con los estilos WS_CHILD y WS_VISIBLE, entonces Windows lo muestra automáticamente en el momento que hace visible la ventana padre.

El paso tres, el monitoreo y respuesta al botón es realizado a través del mensaje WM_COMMAND: cada vez que un botón es liberado, un WM_COMMAND es generado. Otros controles (como los menús y aceleradores) también generan mensajes WM_COMMAND, por lo que hay que poder identificarlos.

Cuando un botón genera un mensaje WM_COMMAND, su identificador (el cuál fue pasado como parámetro en la llamada a CreateWindow) es copiado a la

variable wParam. De esta manera., cuando el programa reciba un mensaje del botón podrá tomar la acción asociada con este.

## Servicios de Comunicación Serie en Windows

Los servicios de comunicación para el puerto serie en Windows están definidos por un conjunto de 17 funciones (Ver la tabla siguiente)

*Tabla 3.1*

*API de Windows para la comunicación serie*

| Nombre de la función | Descripción breve |
|---|---|
| BuildCommDCB | Traduce la descripción del dispositivo a un DCB |
| ClearCommBreak | Restablece la transmisión |
| CloseComm | Cierra el dispositivo de comunicación |
| EnableCommNotification | Habilita o deshabilita la notificación por el mensaje WM_COMMNOTIFY |
| EscapeCommFunction | Realiza una función extendida |
| FlushComm | Vacía la cola de transmisión o recepción |
| GetCommError | Devuelve el estado del dispositivo de comunicación |
| GetCommEventMask | Devuelve el evento del dispositivo |
| GetCommState | Devuelve el bloque de control del dispositivo |
| OpenComm | Abre el dispositivo de comunicación |
| ReadComm | Lee del dispositivo de comunicación |
| SetCommBreak | Suspende la transmisión |
| SetCommEventMask | Habilita la notificación por eventos |
| SetCommState | Cambia el estado del dispositivo de comunicación |
| TransmitCommChar | Inserta un carácter en la cola de transmisión |
| UngetCommChar | Inserta un carácter en la cola de recepción |
| WriteComm | Escribe al dispositivo de comunicación |

Al abrir un puerto serie en Windows, usando la función OpenComm, el valor que devuelve esta última es un entero, el cual es usado como un identificador del puerto en llamadas posteriores a otras funciones relacionadas con la comunicación. Una vez que el puerto esta abierto, es común cambiar parámetros de comunicación tales como: velocidad de transmisión, paridad, control de flujo, bits de parada, etc. Estos parámetros se cambian a través de la función SetCommState y una estructura de datos llamada DCB (device control block). Esta estructura proporciona control sobre el puerto serie, y por cierto, es de uso complejo.

La función GetCommState permite obtener los parámetros actuales del puerto serie.

Ante un error de comunicación, Windows bloquea el puerto serie, para desbloquearlo se tiene que llamar a la función GetCommError. Esta función proporciona el tipo de error ocurrido.

## Arquitectura del Emulador de Terminal

La programación en Windows esta basada en el procesamiento de eventos a través de mensajes. Windows es un ambiente gráfico multitarea, sin embargo los programas deben liberar el control del CPU voluntariamente. Con este lineamiento en mente se diseñó el emulador de terminal.

Las figuras 3.1 y 3.2 ilustran gráficamente el flujo de datos del emulador de terminal.

*Figura 3.1*

*Flujo de eventos cuando el usuario presiona una tecla*

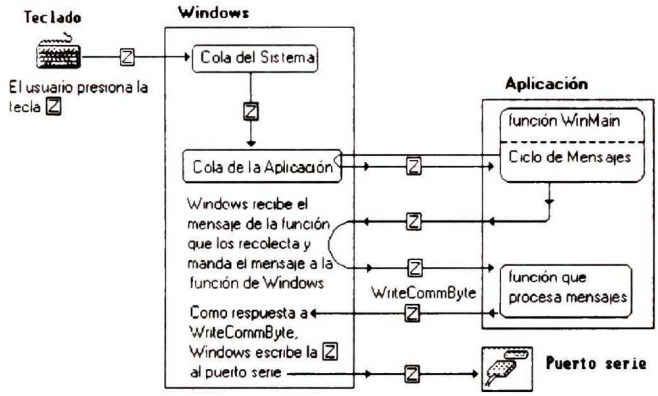
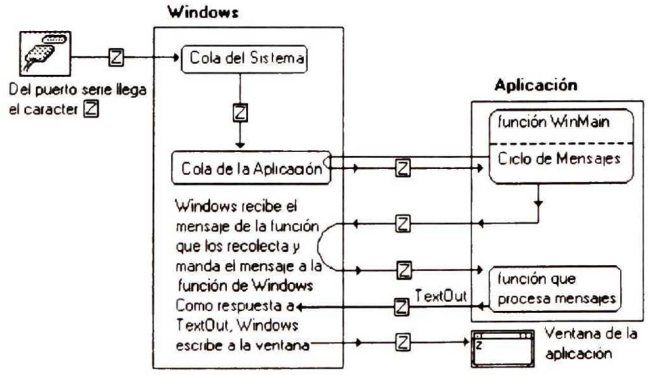



*Figura 3.2*

*Secuencia de eventos cuando se recibe un carácter del puerto serie*



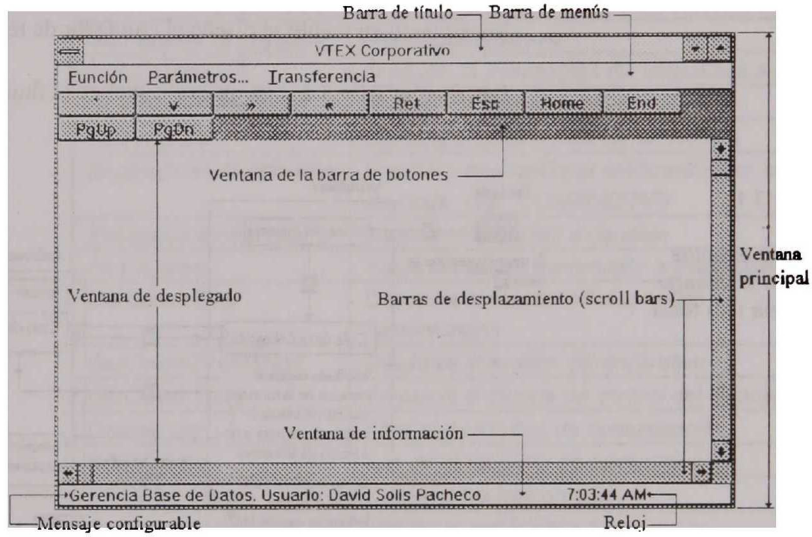
El emulador de terminal tiene 4 ventanas:

- La ventana principal.
- La ventana de desplegado.
- La ventana de información y de reloj.
- La ventana de la barra de botones.

La figura 3.3 ayuda a identificar las 4 ventanas de la aplicación.

*Figura 3.3*

*Las ventanas del emulador de terminal*




## La Ventana Principal

La ventana principal crea a las demás ventanas, tiene como característica que sus ventanas hijas la traslapan. La función que procesa los mensajes relacionados con ella se llama TTYWndProc. El primer mensaje que recibe la función encargada de procesar los mensajes de una ventana es WM_CREATE. Cuando TTYWndProc, recibe este mensaje, crea un temporizador y lo programa para que envíe mensajes cada segundo, además crea a las otras ventanas de la aplicación. Cuando recibe un evento relacionado con el temporizador, le envía un mensaje a la ventana del reloj para que este último se actualice. Los mensajes relacionados con los menús y el teclado se le pasan a la función que procesa los mensajes de la ventana de desplegado de la siguiente manera:

```
/* Para eventos relacionados con el teclado, los pasa
   a la ventana de desplegado
*/
```

```
        case WM_CHAR:
        case WM_KEYDOWN
        case WM_SETFOCUS
        case WM_KILLFOCUS:
            SendMessage(hClientWnd, wMsg, wParam, lParam);
            break;
         * Para eventos relacionados con los menús, también *
        case WM_COMMAND:
            switch((WORD) wParam) {
                default:
                    SendMessage(hClientWnd, WM_USER, wParam, lParam);
                    break;
                case IDM_EXIT:
                    PostMessage(hWnd, WM_CLOSE, NULL, 0L);
                    break;
            }
            break;
```

## La Ventana de Desplegado

En la región de la ventana de desplegado se imprimen todos los caracteres que no son de control. La función que procesa los mensajes relacionados con la ventana de desplegado se llama ClientWndProc. A continuación se presenta el código de ClientWndProc.

```
LRESULT FAR PASCAL
ClientWndProc(HWND hWnd, UINT wMsg, WPARAM wParam, LPARAM lParam)
{
    switch(wMsg) {
        case WM_KEYDOWN:
            * Manda por el puerto serie la secuencia esperada para la tecla */
            KeyEmulate(hWnd, wParam); break;
        case WM_CREATE:
            /* Crea la estructura de datos que describe a la terminal */
            return(CreateTTYInfo(hWnd));
        case WM_USER:
            * Recibe mensajes del padre que indican que debe procesar un menú
              o recibe mensajes de la barra de botones para que envíe la secuencia
              de tecla emulada con el botón
            *
            switch((WORD) wParam) {
                * Procesa menús *
        case WM_USER:
            * Procesa mensajes de menús y botones *
            switch((WORD) wParam) {
                * Conexión *
                case IDM_CONNECT: {
```

```
        NPTTYINFO  npTTYInfo;

        npTTYInfo = (NPTTYINFO) GetWindowWord(hWnd,
                                        GWW_NPTTYINFO);
        * Si no esta conectado, se conecta */
        if(!CONNECTED(npTTYInfo)) {
            if(!OpenConnection(hWnd))
                MessageBox(hWnd, "Falla en conexiÃxF3n", gszAppName,
                    MB_ICONEXCLAMATION);
        }
        /* en caso contrario se desconecta */
        else CloseConnection(hWnd);
        break;
    }
    /* Parámetros -  opciones de puerto serie y terminal */
    case IDM_SETTINGS: {
        NPTTYINFO  npTTYInfo;

        npTTYInfo = (NPTTYINFO) GetWindowWord(hWnd,
                                        GWW_NPTTYINFO);
        /* Crea el diálogo */
        GoModalDialogBoxParam(GETHINST(hWnd),
        MAKEINTRESOURCE(SETTINGSDLGBOX), hWnd, SettingsDlgProc,
                (LPARAM) (LPSTR) npTTYInfo);
        if(CONNECTED(npTTYInfo)) {
            /* Cambia parametros del puerto serie */
            if(!SetupConnection(hWnd))
                /* Error */
                MessageBox(hWnd, "Falla en configuración!", gszAppName,
                    MB_ICONEXCLAMATION);
        }
        break;
    }
    /* Muestra el diálogo de Acerca De */
    case IDM_ABOUT:
        GoModalDialogBoxParam(GETHINST(hWnd),
        MAKEINTRESOURCE(ABOUTDLGBOX), hWnd, AboutDlgProc, NULL);
        break;
    /* Imprime el contenido de la ventana */
    case IDM_PRINTSCREEN:
        PrintScreen(hWnd);
        break;
     * Salva Opciones */
    case IDM_SAVESETTINGS:
        SaveStettings(hWnd);
        break;
     * Ver bitácora */
    case IDM_LOOKLOG: {
```

```
                    char szTemp[32];

                    LoadString(GETHINST(hWnd), IDS_LOGNAME, szTemp, sizeof(szTemp));
                    * Invoca a la aplicación notepad */
                    SpawnApp(szTemp);
                    break;
                }
                * Emula teclas especiales */
                case ID_HOME
                case ID_END
                case ID_PGUP
                case ID_PGDN
                case ID_UP
                case ID_DOWN
                case ID_RIGHT
                case ID_LEFT
                    KeyEmulate(hWnd, wParam); break;
                case ID_RETURN :
                case ID_ESCAPE :
                    SendMessage(hWnd, WM_CHAR, wParam, 0L); break;
            }
            break;
        case WM_COMMNOTIFY:
            * Procesa eventos relacionados con el puerto serie */
            ProcessCOMMNotification(hWnd, (WORD) wParam, (LONG) lParam);
            break;
        case WM_PAINT:
            * Despliega datos en la pantalla */
            BeginPaint(hWnd, (LPPAINTSTRUCT)&ps);
            PaintTTY(hWnd, (LPPAINTSTRUCT)&ps);
            EndPaint(hWnd, (LPPAINTSTRUCT)&ps);
            break;
        case WM_CHAR:
            * Envía los caracteres tecleados al puerto serie */
            ProcessTTYCharacter(hWnd, LOBYTE(wParam));
            break;
        case WM_DESTROY:
            /* Devuelve memoria ocupada por el emulador */
            DestroyTTYInfo(hWnd);
            break;
        default:
            * Envía el mensaje al procedimiento por omisión */
            return(DefWindowProc(hWnd, wMsg, wParam, lParam));
    }
    return(0L);
}
```

Cuando ClientWndProc recibe el mensaje WM_CREATE, reserva e inicializa una área de memoria donde se guardan parámetros del puerto y de la terminal. ClientWndProc también procesa todos los mensajes relacionados con el puerto serie, menús, y teclado. Cuando una ventana se destruye, la función que procesa sus mensajes recibe el mensaje WM_DESTROY. Cuando ClientWndProc recibe el mensaje WM_DESTROY, se libera el bloque de memoria que se reservó al inicio. El proceso del mensaje WM_PAINT es muy importante. Este mensaje se genera cuando es necesario actualizar el contenido de la ventana. Existen dos eventos por los que hay que llevar a cabo esta actualización:

1. Un diálogo del emulador u otra aplicación traslaparon la ventana.
2. Se recibieron caracteres del puerto serie que se necesitan mostrar.

PaintTTY es la función que ClientWndProc llama cuando se recibe el mensaje WM_PAINT.

ClientWndProc recibe el mensaje WM_COMMNOTIFY cuando llegan caracteres del puerto serie. En el momento que este evento ocurre se llama a la función ProcessCOMMNotification. Esta última función extrae los caracteres de la cola de recepción y llama a la función WriteTTYBlock, donde son procesados. Se lista a continuación la función ProcessCOMMNotification.

```
BOOL NEAR
ProcessCOMMNotification(HWND hWnd, WORD wParam, LONG lParam)
{
    int     nLength;
    MSG     msg;
    BYTE    abIn[MAXBLOCK + 1];
    NPTTYINFO  npTTYInfo;

    // LOWORD ==  evento
    if(CN_EVENT & LOWORD(lParam) != CN_EVENT)
        return(FALSE);
    if(NULL == (npTTYInfo = (NPTTYINFO) GetWindowWord(hWnd, GWW_NPTTYINFO)))
        return(FALSE);
    GetCommEventMask(COMDEV(npTTYInfo), EV_RXCHAR);
    do {
        if(nLength = ReadCommBlock(hWnd, (LPSTR) abIn, MAXBLOCK)) {
            WriteTTYBlock(hWnd, (LPSTR) abIn, nLength);
            /* Genera un mensaje WM_PAINT */
            UpdateWindow(hWnd);
        }
    }
    while(!PeekMessage((LPMSG)&msg, NULL, 0, 0, PM_NOREMOVE) || (nLength > 0));
    return(TRUE);
}
```

Cuando un usuario presiona una tecla, el manejador del teclado notifica de este evento a Windows. Windows guarda el evento en la cola de mensajes y a continuación lo transfiere a la cola de mensajes de la aplicación que se encuentra activa en ese momento. Los programas que necesitan procesar caracteres que llegan del teclado, deben procesar los mensajes WM_CHAR. Para las teclas que no generan caracteres tales como: la tecla shift, teclas de funciones, teclas de movimiento de cursor y teclas especiales como Insert y Delete, Windows envía los mensajes WM_KEYDOWN y WM_KEYUP.

Cuando ClientWndProc recibe un mensaje WM_CHAR, invoca a la función ProcessTTYCharacter, donde el carácter es enviado por el puerto serie. Cuando recibe un mensaje WM_KEYDOWN llama a la función KeyEmulate, la cual envía la secuencia de la terminal correspondiente a la tecla. La siguiente tabla muestra estas correspondencias.

| *Tecla* | *Secuencia de control* |
|---|---|
| ↑ | ESC[A |
| ↓ | ESC[B |
| → | ESC[C |
| ← | ESC[D |
| ↵ | \x0D |
| Esc | ESC |
| Home | ESC[H |
| End | ESC[F |
| Page Up | ESC[I |
| PageDown | ESC[G |
| F1 | ESC[M |
| F2 | ESC[N |
| F3 | ESC[O |
| F4 | ESC[P |
| F5 | ESC[Q |
| F6 | ESC[R |
| F7 | ESC[S |
| F8 | ESC[T |

## La Ventana de Información y del Reloj

En la ventana de información y del reloj se muestra un mensaje configurable (pueden ser el nombre de la aplicación y los datos del usuario como nombre y apellidos) y un reloj digital. La función que procesa los mensajes de esta ventana se llama StatLineWndProc. StatLineWndProc recibe un mensaje WM_PAINT cada segundo, y llama a StatLineWndPaint, para que muestra la hora en la ventana. Se muestra a continuación el código de StatLineWndPaint.

```
VOID NEAR PASCAL
```

```
StatLineWndPaint(HWND hWnd, HDC hDC)
{
    char     cBuffer[2*MAXLEN_TEMPSTR];
    char     cTmpStr[MAXLEN_TEMPSTR];
    long     lTime;
    short    nLength;
    struct tm  *datetime;
    RECT     rcWindow;
    NPTTYINFO  npTTYInfo;

    if(NULL == (npTTYInfo = (NPTTYINFO) GetWindowWord(hClientWnd,
     GWW_NPTTYINFO)))
        return;
    /* Obtiene el mensaje a mostrar */
    LoadString(GETHINST(hTTYWnd), IDS_DEF_MESSAGE, cTmpStr, sizeof(cTmpStr));
    nLength = wsprintf(cBuffer, cTmpStr, (LPSTR) szName);
    /* Obtiene la hora actual */
    time(&lTime);
    datetime = localtime(&lTime);
    /* Despliega la hora según la configuración de Windows
      las horas son mostradas como 0-23:0-60:0-60
      o 0-12:0-60:0-60 [AP]M
      el separador (en este caso ":" se encuentra en la variable sTime
    */
    if(iTime == 1)
        nLength += wsprintf(cBuffer + nLength, "%02d%s%02d%s%02d",
                    datetime- tm_hour,
                    (LPSTR) sTime, datetime- tm_min, (LPSTR) sTime,
                    datetime- tm_sec);
    else
        nLength += wsprintf(cBuffer + nLength, "%d%s%02d%s%02d %s",
                    datetime- tm_hour % 12 ? datetime- tm_hour % 12 : 12,
                    (LPSTR) sTime, datetime- tm_min, (LPSTR) sTime,
                    datetime- tm_sec,
                    (LPSTR) sAMPM[datetime- tm_hour / 12]);
        ExtTextOut(hDC, 0, 0, ETO_OPAQUE, &rcWindow, (LPCSTR)cBuffer, nLength, NULL);
}
```

## La Ventana de la Barra de Botones

Esta ventana contiene botones que evitan el uso de algunas teclas. Cuando el usuario presiona uno de estos botones, se simula que se tecleó determinado carácter. La siguiente lista muestra la relación que existe entre botones, teclas simuladas y secuencias de control de la terminal.

| Botón | Tecla simulada |
|---|---|
| ^ | ↑ |
| v | ↓ |
| → | → |
| ← | ← |
| Ret | ↵ |
| Esc | Esc |
| Home | Home |
| End | End |
| PgUp | Page Up |
| PgDn | PageDown |

ButtonBarWndProc es la función que procesa los mensajes de esta ventana. A continuación su listado.

```
LRESULT FAR PASCAL
ButtonBarWndProc(HWND hWnd, UINT wMsg, WPARAM wParam, LPARAM lParam)
{
    // Función que procesa los mensajes de la barra de botones

    switch(wMsg) {
        case WM_COMMAND:
            switch(wParam) {
                case ID_HOME:
                case ID_END:
                case ID_PGUP:
                case ID_PGDN:
                case ID_UP:
                case ID_DOWN:
                case ID_RIGHT:
                case ID_LEFT:
                case ID_ESCAPE:
                case ID_RETURN:
                    /* Envía un mensaje a la ventana de desplegado */
                    SendMessage(hClientWnd, WM_USER, wParam, 0L);
                    SetFocus(GetParent(hWnd));
                    break;
            }
            break;
        default:
            return(DefWindowProc(hWnd, wMsg, wParam, lParam));
    }
    return(0L);
}
```

La función ClientWndProc recibe los mensajes que le envía ButtonBarWndProc y a continuación simula que se presionó la tecla correspondiente.

## Interfaz con el API de Comunicación Serie de Windows

El usuario inicia la conexión con el sistema remoto a través de la opción Conexión del menú Función. Windows envía un mensaje al emulador, la función que atiende los mensajes, ClientWndProc, invoca a OpenConnection. Esta última función abre el puerto serie usando OpenComm, cambia los parámetros del puerto, habilita la notificación del evento EV_RXCHAR y activa la señal DTR con la función EscapeCommFunction.

Cuando el usario selecciona el menú Parámetros, Windows notifica a el programa y ClientWndProc en respuesta muestra un diálogo que permite al usuario cambiar opciones del puerto o de la terminal. Una vez realizada esta acción se llama a la función SetupConnection para cambiar los parámetros de la comunicación serie. Esta función llama a la función GetCommState para obtener los parámetros del puerto serie en la estructura dcb y después cambia algunos de los campos de esta y llama a la función SetCommState pasándole como parámetro la estructura. Se muestra a continuación la función SetupConnecction.

```
BOOL NEAR
SetupConnection(HWND hWnd)
{
    BOOL    fRetVal;
    BYTE    bSet;
    DCB     dcb;
    NPTTYINFO  npTTYInfo;

    if(NULL == (npTTYInfo = (NPTTYINFO) GetWindowWord(hWnd, GWW_NPTTYINFO)))
        return(FALSE);
    /* Obtiene la configuración del puerto */
    GetCommState(COMDEV(npTTYInfo), &dcb);
    /* Cambia algunos campos de la estructura dcb */
    dcb.BaudRate = BAUDRATE(npTTYInfo);
    dcb.ByteSize = BYTESIZE(npTTYInfo);
    dcb.Parity = PARITY(npTTYInfo);
    dcb.StopBits = STOPBITS(npTTYInfo);
    /* Parámetros de control de flujo */
    bSet = (BYTE) ((FLOWCTRL(npTTYInfo) & FC_DTRDSR) != 0);
    dcb.fOutxDsrFlow = dcb.fDtrflow = bSet;
    dcb.DsrTimeout = (bSet) ? 30 : 0;
    bSet = (BYTE) ((FLOWCTRL(npTTYInfo) & FC_RTSCTS) != 0);
    dcb.fOutxCtsFlow = dcb.fRtsflow = bSet;
```

```
    dcb.CtsTimeout = (bSet) ? 30 : 0;
    bSet = (BYTE) ((FLOWCTRL(npTTYInfo) & FC_XONXOFF) != 0);
    dcb.fInX = dcb.fOutX = bSet;
    dcb.XonChar  = ASCII_XON;
    dcb.XoffChar = ASCII_XOFF;
    dcb.XonLim = 100;
    dcb.XoffLim = 100;
    dcb.fBinary = TRUE;
    dcb.fParity = TRUE;
    dcb.fRtsDisable = FALSE;
    dcb.fDtrDisable = FALSE;
     * Cambia parámetros del puerto */
    fRetVal = !(SetCommState(&dcb)    0);
    return(fRetVal);
}
```

Para terminar la sesión, el usuario selecciona la opción √Conexión (El símbolo √ indica que el emulador está conectado) del menú Función. ClientWndProc llama a la función CloseConnection. En esta última, se deshabilita la notificación de mensajes relacionados a eventos ocurridos al puerto serie, se inactiva la señal DTR y se cierra el puerto.

## Interfaces con el Usuario

Las interfaces de usuario a considerar son:

- Menús
- Manejo de diálogos comunes
- Diálogos (Información y Parámetros)

## Menús

El emulador de terminal tiene 2 menús. Un menú llamado Acción y otro llamado Parámetros. El menú Acción tiene opciones para iniciar y terminar la conexión (Conexión), para ver la bitácora (Ver Bitácora), para imprimir el contenido de la ventana (Imprimir Pantalla), para guardar los parámetros de configuración (Salva Opciones), para mostrar información del autor (Acerca De) y para cerrar de la aplicación (Salida). La función del menú Parámetros es poder cambiar algunas opciones del puerto serie tales como: velocidad, paridad, control de flujo, bits de datos, y algunas opciones del control de la terminal como: fonts, auto línea, eco.

La función ClientWndProc recibe un mensaje WM_USER de la función TTYWndProc, el que le indica que tiene que responder a eventos relacionados con los menús y el teclado. A continuación se muestra la parte de

ClientWndProc que procesa el mensaje WM_USER correspondiente a los menús.

```
case WM_USER:
    * Procesa mensajes de menús y botones *
    switch((WORD) wParam) {
        * Conexión *
        case IDM_CONNECT: {
            NPTTYINFO  npTTYInfo;

            npTTYInfo = (NPTTYINFO) GetWindowWord(hWnd,
                                        GWW_NPTTYINFO);
            /* Si no esta conectado, se conecta */
            if(!CONNECTED(npTTYInfo)) {
                if(!OpenConnection(hWnd))
                    MessageBox(hWnd, "Falla en conexi\xF3n", gszAppName,
                        MB_ICONEXCLAMATION);
            }
            /* en caso contrario se desconecta */
            else CloseConnection(hWnd);
            break;
        }
        * Parámetros -  opciones de puerto serie y terminal */
        case IDM_SETTINGS: {
            NPTTYINFO  npTTYInfo;

            npTTYInfo = (NPTTYINFO) GetWindowWord(hWnd,
                                        GWW_NPTTYINFO);
            * Crea el diálogo */
            GoModalDialogBoxParam(GETHINST(hWnd),
            MAKEINTRESOURCE(SETTINGSDLGBOX), hWnd, SettingsDlgProc,
                    (LPARAM) (LPSTR) npTTYInfo);
            if(CONNECTED(npTTYInfo)) {
                /* Cambia parámetros del puerto serie */
                if(!SetupConnection(hWnd))
                    /* Error */
                    MessageBox(hWnd, "Falla en ajuste!", gszAppName,
                        MB_ICONEXCLAMATION);
            }
            break;
        }
        /* Muestra el diálogo de Acerca De */
        case IDM_ABOUT:
            GoModalDialogBoxParam(GETHINST(hWnd),
            MAKEINTRESOURCE(ABOUTDLGBOX), hWnd, AboutDlgProc, NULL);
            break;
        * Imprime el contenido de la ventana */
        case IDM_PRINTSCREEN:
```

```
        PrintScreen(hWnd);
        break;
    * Salva Opciones *
    case IDM_SAVESETTINGS:
        SaveSettings(hWnd);
        break;
    * Ver bitácora *
    case IDM_LOOKLOG: {
        char szTemp[32];

        LoadString(GETHINST(hWnd), IDS_LOGNAME, szTemp, sizeof(szTemp));
        * Invoca a la aplicación notepad */
        SpawnApp(szTemp);
        break;
    }
```

## Manejo de Diálogos Comunes

Un diálogo común es un diálogo que se construye llamando a una sola función, en lugar de crear el procedimiento que maneja los mensajes del diálogo y un archivo de recursos donde se encuentra la definición de este.

La librería dinámica COMMDLG.DLL contiene procedimientos y definiciones de los diálogos comunes. Los procedimientos procesan los mensajes y notificaciones de estos diálogos y de sus controles. En la definición se encuentra la especificación de la apariencia de los diálogos comunes y de sus controles.

Los diálogos comunes simplifican el desarrollo de las aplicaciones, además proporcionan al usuario un conjunto estándar de controles para realizar ciertas acciones.

La siguiente lista muestra los tipos de diálogos comunes que fueron usados en el emulador:

| *Nombre* | *Descripción* |
|---|---|
| Font | Muestra una lista de diseños gráficos aplicados a todos los números, símbolos y caracteres del alfabeto (fonts), tamaños y colores que corresponden a los fonts disponibles; después de que el usuario selecciona un font, el diálogo muestra algunas letras con este font. |
| Impresión | Muestra información de la impresora instalada y de su configuración. El usuario inicia el proceso de impresión seleccionando los controles de este diálogo. |

Existen además diálogos comunes para abrir archivos, salvar archivos, búsqueda y remplazo de texto.

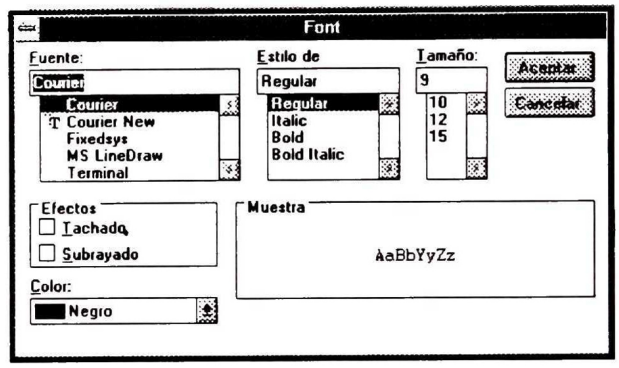
## Diálogo de Font

El diálogo de font contienen controles que permiten al usuario seleccionar: un font, un estilo de font (negritas, itálico o normal), tamaño y efectos (subrayado, color).

La siguiente figura muestra el diálogo de font.

*Figura 3.4*

*Diálogo de font*




El diálogo de font aparece después de que se inicializan los miembros de la estructura CHOOSEFONT y se invoca a la función ChooseFont. Los miembros de la estructura CHOOSEFONT contienen la siguiente información:

- Los atributos del font que se muestra al inicio.
- Los atributos del font que seleccionó el usuario.
- El tamaño del font que seleccionó el usuario.
- Una bandera que indica si la lista de fonts corresponde a la impresora, a la pantalla o a ambas.
- Una bandera que indica si los fonts disponibles son solo de tipo TrueType.
- Una bandera que indica si los mensajes del diálogo deben ser procesados por una función de la aplicación.
- Una bandera que indica si los tamaños de los fonts seleccionados deben ser limitados a un rango especifico.

Para mostrar el diálogo del font, una aplicación debe realizar los siguientes pasos:

1. Obtener un identificador del contexto de la ventana y usarlo para asignarlo al campo hDC.
2. Asignar las banderas apropiadas al campo Flags

3. Si el color negro no es el adecuado, asignar al campo rgbColors el color apropiado
4. Asignarle al campo nFontType la constante apropiada.
5. Si se quiere limitar el tamaño del font, hay que asignar los valores a los campos nSizeMin y nSizeMax y al campo Flags hay que agregarle el valor CF_LIMITSIZE.
6. Llamar a la función ChooseFont.

**Nota**: Todos los campos anteriormente descritos pertenecen a la estructura CHOOSEFONT.

Se muestra a continuación la función SelectTTYFont donde se inicializa la estructura CHOOSEFONT.

```
BOOL NEAR
SelectTTYFont(HWND hDlg)
{
    CHOOSEFONT  cfTTYFont;
    NPTTYINFO   npTTYInfo;

    if(NULL == (npTTYInfo = (NPTTYINFO) GET_PROP(hDlg,
    ATOM_TTYINFO)))
        return(FALSE);
    cfTTYFont.lStructSize    = sizeof(CHOOSEFONT);
    cfTTYFont.hwndOwner      = hDlg;
    cfTTYFont.hDC            = NULL;
    cfTTYFont.rgbColors      = FGCOLOR(npTTYInfo);
    cfTTYFont.lpLogFont      = &LFTTYFONT(npTTYInfo);
    cfTTYFont.Flags          = CF_SCREENFONTS | CF_FIXEDPITCHONLY |
                               CF_EFFECTS | CF_INITTOLOGFONTSTRUCT;
    cfTTYFont.lCustData      = NULL;
    cfTTYFont.lpfnHook       = NULL;
    cfTTYFont.lpTemplateName = NULL;
    cfTTYFont.hInstance      = GETHINST(hDlg);
    if(ChooseFont(&cfTTYFont)) {
        int i, j;

        /* Copia el color del font */
        FGCOLOR(npTTYInfo) = cfTTYFont.rgbColors;
        /* Para todas las filas y columnas */
        for(i = 0; i < MAXROWS; i++)
            for(j = 0; j < MAXCOLS; j++) {
                /* Copia el color */
                FGSCREEN(npTTYInfo, i, j) = FGCOLOR(npTTYInfo);
        }
        /* Restablece la terminal */
        ResetTTYScreen(GetParent(hDlg), npTTYInfo);
```

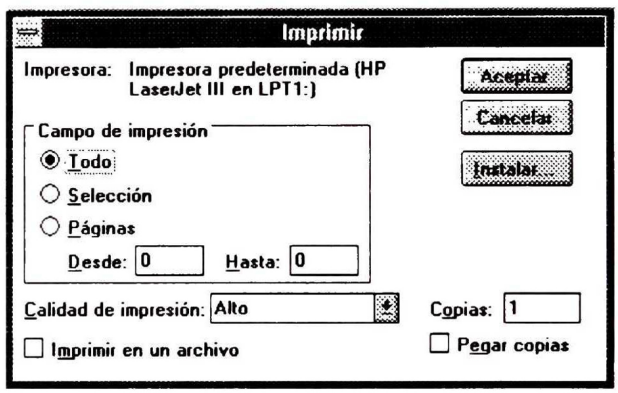
```
            /* Genera un mensaje WM_PAINT */
            InvalidateRect(GetParent(hDlg), NULL, TRUE);
        }
        return(TRUE);
    }
```

## Diálogo de Impresión

El diálogo de Impresión contiene controles que permiten al usuario configurar una impresora para un trabajo particular de impresión. El usuario puede seleccionar la calidad de la impresora, el rango de impresión y el número de copias. La siguiente figura muestra el diálogo de impresión para una impresora HP LaserJet III.

*Figura 3.5*

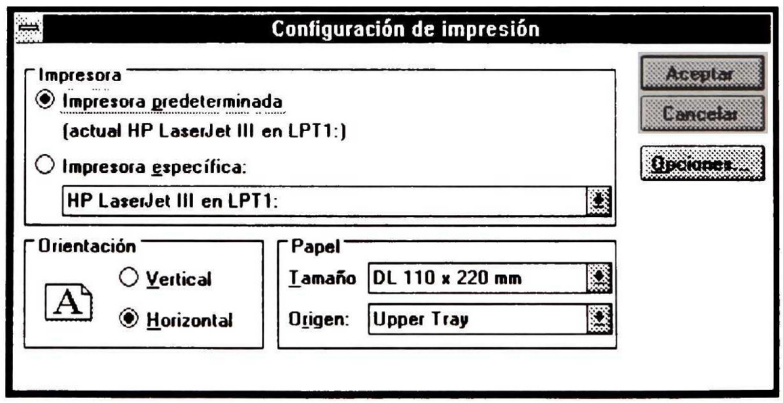
*Diálogo de impresión*




Si se selecciona el botón de Instalar, se muestra el siguiente diálogo de Configuración de impresión.

*Figura 3.6*

*Diálogo de configuración de la impresora*



El diálogo de Configuración de impresión proporciona controles que hacen posible que el usuario reconfigure la impresora.

Para mostrar el diálogo de Impresión para la impresora, una aplicación debe inicializar una estructura de tipo PRINTDLG y llamar a la función PrintDlg.

Los miembros de PRINTDLG son usados para almacenar la siguiente información:

- Otras estructura de datos de la impresora.
- El contexto del dispositivo que maneja la impresora.
- Valores que se desea que aparezcan en los controles del diálogo.
- El apuntador a la función suministrada por la aplicación que ayuda a modificar el comportamiento normal del diálogo.

Para mostrar el diálogo de la Impresión, una aplicación debe realizar los siguientes pasos:

1. Asignar la bandera PD_RETURNDC en el campo Flags.
2. Inicializar los campos lStructSize, hDevMode y hDevNames.
3. Llamar a la función PrinDlg y pasar como parámetro la estructura PRINTDLG.

**Nota**: Todos los campos anteriormente descritos pertenecen a la estructura PRINTDLG.

A continuación el código de la función InitializeStruct, donde se inicializa la estructura PRINTDLG.

```
void NEAR
InitializeStruct(LPPRINTDLG lpPrintChunk, HWND hWnd)
{

    /* Inicializa la estructura PRINTDLG */
    lpPrintChunk->lStructSize      = sizeof(PRINTDLG);
    lpPrintChunk->hwndOwner        = hWnd;
    lpPrintChunk->hDevMode         = (HANDLE)NULL;
    lpPrintChunk->hDevNames        = (HANDLE)NULL;
    lpPrintChunk->hDC              = (HDC)NULL;
    lpPrintChunk->Flags            = PD_RETURNDC;
    lpPrintChunk->nFromPage        = 0;
    lpPrintChunk->nToPage          = 0;
    lpPrintChunk->nMinPage         = 0;
    lpPrintChunk->nMaxPage         = 0;
    lpPrintChunk->nCopies          = 0;
```

```
    lpPrintChunk->  hInstance        = (HANDLE)NULL;
    lpPrintChunk->  lCustData        = 0L;
    lpPrintChunk->  lpfnPrintHook    = NULL;
    lpPrintChunk->  lpfnSetupHook    = NULL;
    lpPrintChunk->  lpPrintTemplateName = (LPSTR)NULL;
    lpPrintChunk->  lpSetupTemplateName = (LPSTR)NULL;
    lpPrintChunk->  hPrintTemplate   = (HANDLE)NULL;
    lpPrintChunk->  hSetupTemplate   = (HANDLE)NULL;
}
```

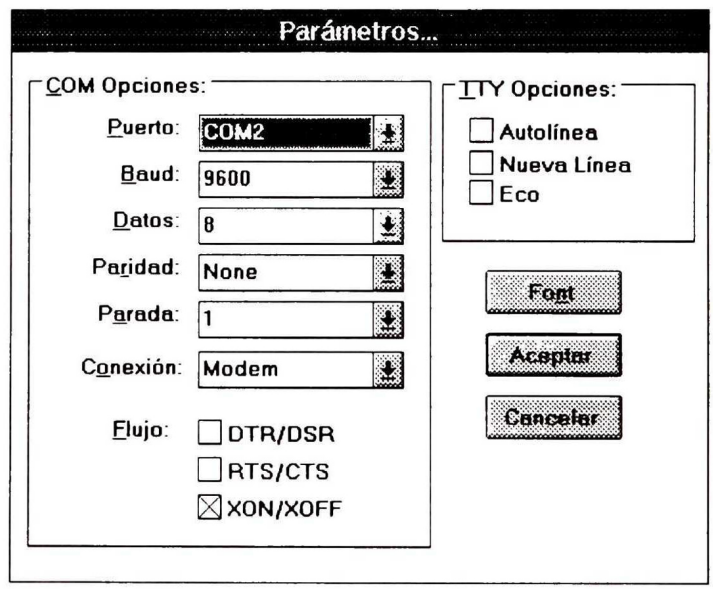
# Manejo de Diálogos

## Diálogo de Configuración de Parámetros

La interfaz de usuario del emulador de terminal cuenta con un diálogo para cambiar los parámetros del puerto serie y los de la terminal. Si se selecciona el botón Font, aparece el diálogo común de Fonts, donde el usuario puede cambiar el font de la terminal.

*Figura 3.7*

*El diálogo de Configuración de Parámetros sirve para cambiar opciones del puerto serie y de la terminal*




La función SettingsDlgProc es la encargada de procesar los mensajes relacionados con este diálogo. Cuando recibe el mensaje WM_INITDIALOG, llama a la función SettingsDlgInit para copiar los valores actuales del puerto serie y de la terminal a los controles del diálogo. Cuando el usuario selecciona cualquiera de los tres botones del diálogo, SettingsDlgProc recibe un mensaje

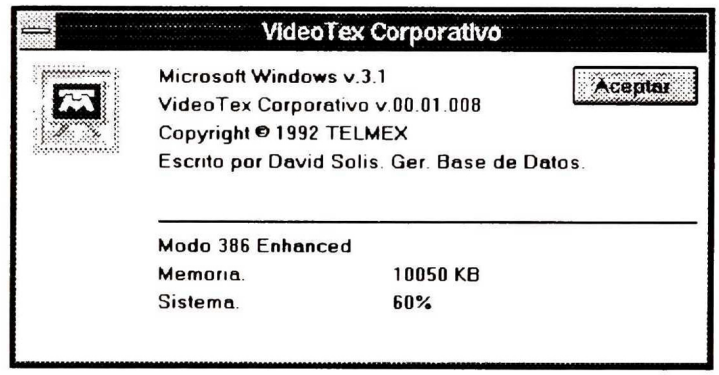
WM_COMMAND. El parámetro wParam ayuda a identificar el botón que seleccionó el usuario. Si el usuario selecciona el botón Font, se invoca a la función SelectTTYFont. Si el usuario selecciona Aceptar, se llama a la función SettingsDlgTerm, la cual copia los valores de los controles a una estructura que guarda los valores actuales del puerto serie y de la terminal. Si el usuario selecciona Cancelar, se llama a la función de Windows EndDialog, la cual destruye el diálogo.

### Diálogo de Información

El diálogo de información se muestra al usuario cuando selecciona del menú Función la opción Acerca De. Este diálogo se muestra a continuación.

*Figura 3.7*

*Diálogo de Información donde se muestra información acerca del autor, versión de Windows, versión del emulador, etc*




La función AboutDlgProc procesa los mensajes relacionados con el diálogo de información. Cuando AboutDlgProc recibe el mensaje WM_INITDIALOG, obtiene la información de la versión del programa, autor, versión de Windows, modo de operación de Windows, memoria y porcentaje de recursos libres, y lo copia a los controles del diálogo. Además muestra una imagen almacenada en forma de matriz de bits (mapa de bits o bitmap). Cuando recibe el mensaje del botón Aceptar, llama a la función EndDialog para que destruya el diálogo.

## Control de la Terminal

Existen dos clases de caracteres de control de una terminal:

1. Caracteres individuales, por ejemplo: retroceso (backspace) y retorno (return)
2. Combinación de caracteres, tales como: la secuencia que limpia el contenido de la pantalla, las secuencias que mueven el cursor, etc.

La función WriteTTYBlock procesa las secuencias de control y los imprimibles en la pantalla. Es fácil de entender el proceso de los caracteres individuales de control a través de una cuidadosa examinación del código. Por lo tanto, esta discusión se concentrará en las secuencias de control que son multicarácter.

Las secuencias de control para una terminal ANSI tienen la siguiente forma:

<ESC> [ <n> ; <m> . .. <SPECCHAR>

La secuencia de control es introducida por una secuencia de dos caracteres, el carácter de ESCAPE (ESC hexadecimal 0x1B) y el paréntesis izquierdo cuadrado ([ hexadecimal 0x5B). Después de esta secuencia hay una serie opcional de parámetros numéricos representados como números ASCII y separados por punto y coma. Dependiendo del comando, si estos últimos son omitidos, tendrán un valor de 0 o 1, . Finalmente, un carácter ASCII individual , <SPECCHAR>, determina la secuencia de escape.

Por ejemplo, la secuencia <ESC> [ <n> ; <m> H solicita a la terminal que posicione el cursor en la línea <n>, columna <m>. La secuencia <ESC> [ 2 J limpia el contenido de la pantalla.

La siguiente tabla especifica algunas de las secuencias de control para una terminal ANSI. Se implementó un subconjunto de esta especificación.

*Tabla 3.2*

*Secuencia de control para la especificación IBM ANSI*

| Secuencia | Descripción |
|---|---|
| <ESC> [ <n> A | Mueve el cursor <n> líneas hacia arriba |
| <ESC> [ <n> B | Mueve el cursor <n> líneas hacia abajo |
| <ESC> [ <n> C | Mueve el cursor <n> líneas hacia la derecha |
| <ESC> [ <n> D | Mueve el cursor <n> líneas hacia la izquierda |
| <ESC> [ <n> , <m> H | Posiciona el cursor en la línea <n>, columna <m> |
| <ESC> [ 2 J | Limpia el contenido de la pantalla |
| <ESC> [ K | Limpia hasta el fin de línea |
| <ESC> [ 10 ; 10 ^ | Inicia la recepción de archivos, usando el protocolo Kermit (Esta es una extensión a la especificación) |
| <ESC> [ <n> ; <m> f | Igual que <ESC> [ <n> ; <m> H |
| <ESC> [ <$a_1$> , ... , <$a_n$> m | Selecciona el atributo para el desplegado de caracteres |
| <ESC> [ = <n> h | Cambia el modo de video |
| <ESC> [ = <n> l | Restablece el modo de video <n> |
| <ESC> [ <$a_1$> , ... ; <$a_n$> p<br><ESC> [ "comando" p<br><ESC> [ 'comando' p | Permite reasignar el teclado |
| <ESC> [ s | Salva la posición actual del cursor |
| <ESC> [ u | Restablece la posición del cursor |

A continuación se presenta el listado de la función WriteTTYBlock

```
BOOL NEAR
WriteTTYBlock(HWND hWnd, LPSTR lpBlock, int nLength)
{
    int     i;
    NPTTYINFO  npTTYInfo;

    if(NULL == (npTTYInfo = (NPTTYINFO) GetWindowWord(hWnd, GWW_NPTTYINFO)))
        return(FALSE);
    for(i = 0; i    nLength; i++) {
        /* Para depuración
        LogSerial(lpBlock[i]);
        /* Ve si el carácter es un  ESC
        if(lpBlock[i] == ASCII_ESC) {
            State = BRACKET_STATE;
            EscapeMode = TRUE;          * Indica que se encontró un ESC */
            StoreChar(lpBlock[i]);   * Almacena el ESC */
        }
```

```
    Procesa la secuencia de escape
else if(EscapeMode) ProcessEscape(hWnd, lpBlock + i);
else {
    if(PrintMode) {
        continue;
    }
    // Procesa caracteres individuales
    switch(lpBlock[i]) {
        case ASCII_NUL:
            break;
        case ASCII_BEL:
            MessageBeep(0);
            break;
        case ASCII_BS:
            if(COLUMN(npTTYInfo)    0)
            COLUMN(npTTYInfo)--;
            MoveTTYCursor(hWnd);
            break;
        case ASCII_TAB:
            do {
                WriteTTYBlock(hWnd, " ", 1);
            } while(COLUMN(npTTYInfo) % 8);
            break;
        case ASCII_CR:
            COLUMN(npTTYInfo) = 0;
            MoveTTYCursor(hWnd);
            if(!NEWLINE(npTTYInfo))
                break;
        case ASCII_LF:
            ProcessLF(hWnd);
            break;
        default:
            // Mapea algunos caracteres especiales como XON, XOFF, etc.
            switch(lpBlock[i]) {
                case 0x83: lpBlock[i] = 0x18; break;
                case 0x84: lpBlock[i] = 0x19; break;
                case 0x85: lpBlock[i] = 0x1A; break;
                case 0x86: lpBlock[i] = 0x1B; break;
            }
            // Escribe el carácter en el buffer
            WriteTTYChar(hWnd, lpBlock + i);
            // Line wrap
            if(COLUMN(npTTYInfo)    MAXCOLS - 1)
                COLUMN(npTTYInfo)++;
            else if(AUTOWRAP(npTTYInfo))
                WriteTTYBlock(hWnd, "\r\n", 2);
            //MoveTTYCursor(hWnd);
            break;
```

```
            }
        }
    }
    return(TRUE);
}
```

La función ProcessEscape implementa un autómata manejado por una variable llamada State. Los valores posibles de State son:

- BRACKET_STATE
- SPECIAL_STATE
- CLS_STATE
- PARAM_STATE

El estado BRACKET_STATE reconoce el carácter [ y pasa al estado SPECIAL_STATE. En este estado se investiga si la secuencia de escape és para borrar la pantalla (<ESC> [ 2 J), en caso afirmativo se pasa al estado CLS_STATE, en caso contrario a PARAM_STATE. En este último, se procesan los parámetros numéricos, si es que los hay, y se llama a la función ProcessCommand, que es la que ejecuta la acción asociada a la secuencia de control. Todos los caracteres de la secuencia se guardan en un buffer, si en algún momento se detecta que se ha procesado una secuencia inválida, los caracteres del buffer son procesados como caracteres individuales. Ver el listado del código fuente de la función ProcessEscape que se muestra en las líneas siguientes.

```
VOID NEAR
ProcessEscape(HWND hWnd, LPSTR lpBlock)
{
    NPTTYINFO npTTYInfo;

    if(NULL == (npTTYInfo = (NPTTYINFO) GetWindowWord(hWnd, GWW_NPTTYINFO)))
        return;
    switch(State) {
        case BRACKET_STATE:
            State = ESPECIAL_STATE;
            if(*lpBlock == '[') StoreChar(*lpBlock);
            else FlushBuffer(hWnd, lpBlock);
            break;
        case ESPECIAL_STATE:
            State = CLS_STATE;
            if(*lpBlock == '2') StoreChar(*lpBlock);
            else {
                State = PARAM_STATE;
                if(*lpBlock == '=') StoreChar(*lpBlock);
                else ParamState(hWnd, lpBlock);
```

```
                }
                break;
            case CLS_STATE:
                State = PARAM_STATE;
                if(*lpBlock == 'J') {
                    ProcessCommand(hWnd, lpBlock);
                }
                else EscapeRegister[0] = 2;
            case PARAM_STATE:
                ParamState(hWnd, lpBlock);
                break;
        }
}
```

La función ProcessCommand realiza una búsqueda binaria para reconocer el carácter <SPECCHAR> de la secuencia de control y ejecuta la acción apropiada. En ProcessCommand se define una estructura que tiene un campo para <SPECCHAR> de tipo char y un apuntador a función, donde se almacena la dirección de la función a ejecutar. A continuación se muestra el listado de la función ProcessCommand.

```
VOID NEAR
ProcessCommand(HWND hWnd, LPSTR lpBlock)
{
    int i;
    NPTTYINFO npTTYInfo;
     struct {
        char letter;
        VOID (NEAR PASCAL *ptrf) (HWND);
    } *Low, *Mid, *High, CommandTable[] = {
#define COM_LOW 0
#define COM_HIGH sizeof(CommandTable) / sizeof(CommandTable[0]) - 1
        {'A',  AnsiCurUp},
        {'B', AnsiCurDown},
        {'C', AnsiCurRight},
        {'D', AnsiCurLeft},
        {'H', AnsiGotoxy},
        {'J',  AnsiClear},
        {'K', AnsiClearEol},
        {'^', KermitAutoDownLoad},
        {'f', AnsiGotoxy},
        {'h',  dummy},
        {'i',  ProcessPrinter},
        {'l', dummy},
        {'m',  AnsiSetColor},
        {'n', dummy},
        {'p', dummy},
        {'s',  dummy},
```

```
        {'u', dummy }
    };

    if(NULL == (npTTYInfo = (NPTTYINFO) GetWindowWord(hWnd, GWW_NPTTYINFO)))
        return;
    * Realiza una busqueda binaria *
    Low  = &CommandTable[COM_LOW];
    High = &CommandTable[COM_HIGH];
    while(Low <= High) {
        Mid = Low + (High - Low) / 2;
        if((i = *lpBlock - Mid->letter) < 0) High = Mid - 1;
        else if(i > 0) Low = Mid + 1;
        * La búsqueda fue exitosa *
        else {
            (*Mid->ptrf)(hWnd);
            ResetState();
            return;
        }
    }
    * La búsqueda fracasó, vaciar el buffer *
    FlushBuffer(hWnd, lpBlock);
}
```

## Impresión

Cuando el usuario selecciona la opción Imprimir pantalla del menú de Función, Windows envía un mensaje y ClientWndProc invoca a la función PrintScreen, la que a su vez llama a StartPrint, a continuación imprime cada una de las líneas de la pantalla y por último llama a EndPrint.

La descripción de este proceso es muy simple pero fueron necesarias 166 líneas de código, y algunos días para su realización.

```
LPSTR NEAR
AllocAndLockMem(HANDLE *hChunk, WORD wSize)
{
    LPSTR lpChunk;

    * Reserva memoria *
    *hChunk = GlobalAlloc(GMEM_FIXED, wSize);
    if(*hChunk) {
        lpChunk = GlobalLock(*hChunk);
        if(!lpChunk) {
            GlobalFree(*hChunk);
```

```
                lpChunk = NULL;
            }
        }
        else {
            lpChunk = NULL;
        }
        return(lpChunk);
}

HWND hAbortDlgWnd;
BOOL bAbort;

int FAR PASCAL
AbortDlg(HWND hDlg, UINT msg, WORD wParam, LONG lParam)
{

    /* Función que procesa los mensajes del diálogo que sirve para
       cancelar la impresión
    */
    switch(msg) {
        case WM_COMMAND:
            /* Indica que se canceló la impresión */
            return(bAbort = TRUE);
        case WM_INITDIALOG:
            SetFocus(GetDlgItem(hDlg, IDCANCEL));
            return(TRUE);
    }
    return(FALSE);
}

BOOL FAR PASCAL
AbortProc(HDC hPr, int Code)
{
    MSG msg;

    /* Función que permite procesar otros mensajes mientras se
       está imprimiendo
    */
    while(!bAbort && PeekMessage(&msg, NULL, NULL, NULL, TRUE))
        if(!IsDialogMessage(hAbortDlgWnd, &msg)) {
            TranslateMessage(&msg);
            DispatchMessage(&msg);
        }
    /* La impresión se abortó */
    return(!bAbort);
}

int     LineSpace;      /* Magnitud de la línea */
```

```
int     LinesPerPage;   * Número de líneas por página */
int     nPageSize;      * Magnitud de la página *
int     IOStatus;
int     CurrentLine;    * Contador de líneas *
BOOL    PrintMode;      * Bandera que indica que se esta imprimiendo */
ABORTPROC lpAbortProc;  * Apuntador a función */
DLGPROC   lpAbortDlg;   * Idem */
LPPRINTDLG lpPDChunk;   * Apuntador a la estructura PRINTDLG */
HANDLE    hPDChunk;     * Identificador del bloque de memoria reservado
                          para la estructura PRINTDLG */

VOID NEAR
StartPrint(HWND hWnd)
{
    TEXTMETRIC TextMetric;
    HFONT    hFont;
    LOGFONT  lfFont;

    * Reserva memoria */
    if(!(lpPDChunk = (LPPRINTDLG)AllocAndLockMem(&hPDChunk, sizeof(PRINTDLG))))
      return;
    * Inicializa la estructura PRINTDLG *
    InitializeStruct(lpPDChunk, hWnd);
    if(PrintDlg(lpPDChunk) != 0) {
        /* Cambia el font de la impresora */
        hFont = SelectObject(lpPDChunk->hDC, GetStockObject(OEM_FIXED_FONT));
        GetObject(hFont, sizeof(LOGFONT), &lfFont);
        lfFont.lfCharSet = OEM_CHARSET;
        hFont = CreateFontIndirect(&lfFont);
        SelectObject(lpPDChunk->hDC, hFont);
        * lpAbortDlg apunta a la función que procesa los mensajes para el diálogo */
        lpAbortDlg = (DLGPROC)  MakeProcInstance((FARPROC) AbortDlg,
    GETHINST(hWnd));
        /* lpAbortProc apunta a la función AbortProc */
        lpAbortProc = (ABORTPROC) MakeProcInstance((FARPROC) AbortProc,
    GETHINST(hWnd));
        Escape(lpPDChunk->hDC, SETABORTPROC, NULL, (LPSTR) (long) lpAbortProc,
            (LPSTR) NULL);
        Escape(lpPDChunk->hDC, STARTDOC, 8, gszTTYClass, NULL);
        bAbort = FALSE;
        hAbortDlgWnd = CreateDialog(GETHINST(hWnd),
    MAKEINTRESOURCE(ABORTDLGBOX), hWnd, lpAbortDlg);
        ShowWindow (hAbortDlgWnd, SW_NORMAL);
        /EnableWindow(hWnd, FALSE);
        * Calcula algunas constantes */
        GetTextMetrics(lpPDChunk->hDC, &TextMetric);
        LineSpace = TextMetric.tmHeight + TextMetric.tmExternalLeading;
        nPageSize = GetDeviceCaps(lpPDChunk->hDC, VERTRES);
```

```
        LinesPerPage = nPageSize  LineSpace - 1;
        * La linea actual es la =1 *
        CurrentLine = 1;
        PrintMode = TRUE;
    }
}

VOID NEAR
EndPrint(HWND hWnd)
{

    * Finaliza la impresión *
    if(IOStatus  = 0 && !bAbort) {
        Escape(lpPDChunk-  hDC, NEWFRAME, 0, 0L, 0L);
        Escape(lpPDChunk-  hDC, ENDDOC, 0, 0L, 0L);
    }
    /EnableWindow(hWnd, TRUE);
    * Destruye el diálogo *
    DestroyWindow(hAbortDlgWnd);
    FreeProcInstance((FARPROC) lpAbortDlg);
    FreeProcInstance((FARPROC) lpAbortProc);
    DeleteDC(lpPDChunk-  hDC);
    if(lpPDChunk-  hDevMode)
    /* Libera la memoria reservada *
    GlobalFree(lpPDChunk-  hDevMode);
    if(lpPDChunk-  hDevNames)
    GlobalFree(lpPDChunk-  hDevNames);
    GlobalUnlock(hPDChunk);
    GlobalFree(hPDChunk);
    PrintMode = FALSE;
}

VOID NEAR
PrintLine(LPSTR lpzStr, int nLen)
{

    * Imprime una línea de texto */
    TextOut(lpPDChunk-  hDC, 0, CurrentLine*LineSpace, lpzStr, nLen);
    if(++CurrentLine  LinesPerPage) {
        CurrentLine = 1;
        IOStatus = Escape(lpPDChunk-  hDC, NEWFRAME, 0, 0L, 0L);
        /if(IOStatus  0 || bAbort)
            /EndPrint();
    }
}

VOID NEAR PASCAL
PrintScreen(HWND hWnd)
```

```
{
    int nLines;
    NPTTYINFO npTTYInfo;

    if(NULL == (npTTYInfo = (NPTTYINFO) GetWindowWord(hWnd, GWW_NPTTYINFO)))
        return;
    StartPrint(hWnd);
    for(nLines = 0; nLines < MAXROWS; nLines++)
        PrintLine((LPSTR)(SCREEN(npTTYInfo) + nLines * MAXCOLS), MAXCOLS);
    EndPrint(hWnd);
}
```

## Configuración

El emulador de terminal usa un archivo de configuración, llamado WINVT.CFG, donde se almacenan los parámetros del puerto serie, la inicialización del modem, la identificación y la contraseña (password) del usuario y un mensaje que se muestra en la ventana de desplegado.

El formato de WINVT.CFG es similar al de WIN.INI. Una configuración típica se muestra a continuación.

```
;En esta sección se definen los parámetros del puerto serie
[Port]
Port=1
BaudRate=ff16
ByteSize=8
Parity=0
StopBits=0
FlowCtrl=4
DirectConnection=0

;Parámetros del modem
[Modem]
DialPrefix=ATDP
DialNumber=6628775 6628625 6629125 6627525 6629500
;Configuración para modem Codex
;InitModem=ATV1Q0H0S11=80M1S0=0S10=100S7=60S6=2&D2*mm3
;Configuración para modem Telebit
InitModem=AT -S65=1&FS66=1S52=4
TimeOut=70

;Usuario: vtx1 Password: vtx1
[Script]
Script="" "" ogin:--ogin:--ogin: vtx1 word: vtx1
```

```
;En esta sección se definen las opciones para desplegado
[Display]
; Mensaje para la ventana de desplegado
Message=Gerencia Base de Datos. Usuario: David Solis
```

Si el archivo WINVT.CFG existe, entonces se llama a la función GetConfig. GetConfig abre el archivo de configuración y copia los valores de los parámetros del puerto serie a la estructura de tipo TTYINFO, a través del apuntador npTTYInfo.

En el menú de Acción, existe una opción para salvar los parámetros del puerto serie (Salva Opciones). Cuando se escoge esta opción, se invoca una función llamada SaveSettings, donde se salvan los parámetros del puerto serie.

La función GetFileIniName, forma el nombre completo del archivo de configuración, es decir, la trayectoria donde se encuentra este archivo más su nombre, por ejemplo: C:\WINDOWS\WINVT.CFG.

Las funciones relacionadas con el archivo de configuración (estas tres últimas) se encuentran en INICFG.C.

## Conexión Automática

Se pretendió evitar que el usuario respondiera a los mensajes de "login:" y "password:", es decir que el acceso al sistema remoto fuera directo. A este procedimiento se le conocerá, dentro de este trabajo, como: conexión automática.

La conexión automática esta basada en el procedimiento de conexión con sistemas remotos que usa un programa de Unix llamado uucp.

El procedimiento de conexión del emulador de terminal, lee de WINVT.CFG de la sección Script, la opción Script, donde esta definida una serie de mensajes que serán enviados al sistema remoto alternando con respuestas esperadas para esos mensajes. El retorno de carro (CR) es agregado a las cadenas de respuestas a menos que explícitamente se indique que no se desea este comportamiento con una secuencia especial de escape (\c).

En el emulador de terminal, la secuencia siempre empieza con una palabra esperada, como se ve en el siguiente ejemplo:

```
"" "" login: \dvtx1 Password: vtx1
```

La primera cadena esta vacía (""), indicando que no se espera nada, la segunda indica que no se transmite nada, excepto el CR que es agregado automáticamente. A continuación es esperada la palabra login: y se envía como respuesta vtx1 con dos segundos de retardo indicado por la secuencia \d. La palabra Password es esperada y se responde con vtx1.

A continuación se listan las secuencias de escape soportadas por el emulador de terminal, con excepción de la cadena vacía (""), estas secuencias son usadas solo en las cadenas de respuestas. Las comillas dobles pueden ser usadas para delimitar cadenas que incluyen espacios, tanto en cadenas esperadas como de respuesta.

| *Secuencia de control* | *Significado* |
|---|---|
| "" | Cadena vacía |
| BREAK | Envía una señal de BREAK |
| \b | Inserta un backspace |
| \B | |
| \d | Retardo de dos segundos |
| \D | |
| \m | Inserta un CR |
| \M | |
| \r | |
| \R | |
| \n | Inserta un carácter de nueva línea |
| \t | Inserta un tabulador |
| \T | |
| \s | Inserta un espacio |
| \S | |
| \nnn | Convierte la cadena numérica octal nnn en un carácter |
| \\ | Inserta un \ |

Existe además otra sintaxis especial: para permitir respuestas alternas cuando una cadena esperada no es recibida, el carácter guión (-), delimita las cadenas de respuesta, por ejemplo:

"" "" ogin:--ogin:--ogin:

Si la cadena ogin: no es recibida en 80 segundos, la cadena entre los caracteres guión (vacía seguida por un CR) es enviada, si no se recibe ninguna respuesta, la secuencia es repetida una vez más.

La función CallUp es la encargada de realizar la conexión automática. Las funciones relacionadas con las cadenas de respuestas, cadenas esperadas y búsquedas se encuentran en el archivo SCRIPT.C. A continuación se muestra el código de CallUp.

```
VOID NEAR PASCAL
CallUp()
{
    BOOL     bConnected, ok;
    NPTTYINFO  npTTYInfo;
    char     *flds[10];
    char     szTemp[MAXLEN_TEMPSTR];
    char     szConfFileName[MAXLEN_TEMPSTR];
    char     *exp, *alternate;
    int      i, kflds;

    if(NULL == (npTTYInfo = (NPTTYINFO) GetWindowWord(hClientWnd,
     GWW_NPTTYINFO)))
        return;
     * Cambia el comportamiento de la ventana de desplegado */
    SetWindowLong(hClientWnd, GWL_WNDPROC, (LONG)ClientWndSubProc);
     * Se asume que la operación será exitosa */
    bConnected = TRUE;
    /* Si la conexión es a un modem, llama por la línea telefónica */
    if(!DIRECTCONNECTION(npTTYInfo))
        bConnected = Dial();
    /* Realiza la conexión automática con el sistema remoto */
    if(bConnected) {
        /* Obtiene el nombre del archivo de configuración */
        GetFileIniName(hTTYWnd, szConfFileName);
         * Obtiene la opción Script de la sección Script */
        GetPrivateProfileString("Script", "Script", "", szTemp, sizeof(szTemp), szConfFileName);
         * Copia las cadenas a un arreglo */
        kflds = getargs(szTemp, flds);
        for(i = 0; i    kflds; i += 2) {
             * Cadena esperada */
            exp = flds[i];
            ok = FALSE;
            while(ok != TRUE) {
                /* Cadenas de respuesta alternas */
                alternate = strchr(exp, '-');
                if(alternate != NULL)
                    *alternate++ = '\0';
                /* Espera la cadena 80 segundos */
                ok = expectstr(exp, 80);
                 * Se recibió la cadena */
                if(ok) {
                    break;
                }
                else {
                }
                if(alternate == NULL) {
                    /* La conexión automática fracasó */
```

```
                    SetWindowLong(hClientWnd, GWL_WNDPROC, (LONG)ClientWndProc);
                    bCallUp = FALSE;
                    return;
                }
                 * Obtiene la cadena de respuesta alterna *
                exp = strchr(alternate, '-');
                if(exp != NULL)
                    *exp++ = '\0';
                 * Envia la cadena alterna *
                sendstr(alternate);
            }
             * Envia la cadena de respuesta *
            sendstr(flds[i + 1]);
        }
    }
     * Restablece el comportamiento de la ventana de desplegado */
    SetWindowLong(hClientWnd, GWL_WNDPROC, (LONG)ClientWndProc);
    bCallUp = FALSE;
}
```

## Manejo del Modem

Debido a la amplia variedad de modems, es un poco difícil la construcción de la interfaz con una aplicación.

Para el diseño de la interfaz con el modem se tomaron en cuenta las siguientes consideraciones:

- Debe ser posible conectar nuevos tipos de modems sin cambiar el código fuente.
- La interfaz debe ser lo suficientemente portable, para que otras aplicaciones la puedan usar.
- La configuración del modem será un proceso manual. Es necesario conocer las cadenas de control del modem y escribirlas en el archivo de configuración.

En el archivo de configuración se agregó una sección llamada [Modem], donde se encuentran opciones para el marcado, números de teléfonos, cadena de inicialización y tiempo de respuesta. Se han realizado pruebas con modems Codex, Telebit y US Robotics. Se muestra a continuación la parte del archivo WINVT.CFG que corresponde a la sección de modems.

```
[Modem]
DialPrefix=ATDP
```

```
DialNumber=6628775 6628625 6629125 6627525 6629500
;Configuración para modem Codex
;InitModem=ATV1Q0H0S11=80M1S0=0S10=100S7=60S6=2&D2*mm3
;Configuración para modem Telebit
InitModem=AT -S65=1&FS66=1S52=4
TimeOut=70
```

La función Dial restablece los parámetros del modem, lo inicializa y luego llama al primer número de teléfono de la lista de la opción DialNumber. Si no se logra la conexión porque el número se encuentra ocupado o el modem remoto no contesta, llama al siguiente y así sucesivamente. Si ya se recorrió toda la lista y no se ha logrado la conexión, repite la operación hasta dos veces más. A continuación se muestra el código de la funciones que componen la interfaz con el modem.

```
VOID NEAR
Flush()
{
    NPTTYINFO  npTTYInfo;

    if(NULL == (npTTYInfo = (NPTTYINFO) GetWindowWord(hClientWnd,
     GWW_NPTTYINFO)))
        return;
    * Vacía la cola del puerto serie */
    FlushComm(COMDEV(npTTYInfo), 1);
    Buflen = 0;
}

VOID NEAR
ResetModem()
{
    int i;

    * Restablece los parámetros del modem */
    for(i = 0; i    3; i++)
        if(sendexpect("ATZ", "OK", 2)) break;
}

VOID NEAR
InitModem()
{
    char szInit[MAXLEN_TEMPSTR];
    char szConfFileName[MAXLEN_TEMPSTR];
    char szMessage[MAXLEN_TEMPSTR];
    int  i;

     * Manda un mensaje a la ventana de información para que muestre que se
```

```
        esta inicializando el modem
        */
        LoadString(GETHINST(hTTYWnd), IDS_INITMDM_MESSAGE, szMessage,
         sizeof(szMessage));
        SendMessage(hStatLineWnd, WM_USER + 1, 0, (LPARAM) (LPSTR) szMessage);
        /* Obtiene el nombre del archivo de Configuración */
        GetFileIniName(hClientWnd, szConfFileName);
        /* Obtiene la cadena de inicialización, por omisión es AT */
        GetPrivateProfileString("Modem", "InitModem", "AT", szInit, sizeof(szInit), szConfFileName);
        /* Envía la cadena al modem tres veces hasta que responde con un OK */
        for(i = 0; i < 3; i++) {
            if(sendexpect(szInit, "OK", 5)) break;
            WinSleep(2);
        }
}

BOOL NEAR
DialModem(char *pzPhoneNumber)
{
    char szMessage[MAXLEN_TEMPSTR];
    char szDialNumber[MAXLEN_TEMPSTR];
    char szConfFileName[MAXLEN_TEMPSTR];
    char szDialPrefix[MAXLEN_TEMPSTR];
    int  nTimeOut;

    /* Envía un mensaje a la ventana de información para que muestre que se
       está llamando
    */
    LoadString(GETHINST(hTTYWnd), IDS_CALL_MESSAGE, szMessage, sizeof(szMessage));
    /strcat(szMessage, pzPhoneNumber);
    SendMessage(hStatLineWnd, WM_USER + 1, 0, (LPARAM) (LPSTR) szMessage);
    /* Obtiene el nombre del archivo de Configuración */
    GetFileIniName(hClientWnd, szConfFileName);
    /* Obtiene el prefijo de la cadena de marcado */
    GetPrivateProfileString("Modem", "DialPrefix", "", szDialPrefix, sizeof(szDialPrefix),
     szConfFileName);
    /* Obtiene el tiempo esperado para lograr la conexión */
    nTimeOut = GetPrivateProfileInt("Modem", "TimeOut", 80, szConfFileName);
    /* Forma la cadena de marcado */
    strcpy(szDialNumber, szDialPrefix);
    strcat(szDialNumber, pzPhoneNumber);
    /* Le indica al modem que llame y espera como respuesta CONNECT */
    if(sendexpect(szDialNumber, "CONNECT ", nTimeOut) == TRUE) {
        /* Envía un mensaje a la ventana de información para que muestre
           que se logró la conexión
        */
        LoadString(GETHINST(hTTYWnd), IDS_CONN_S_MESSAGE, szMessage,
     sizeof(szMessage));
```

```
            SendMessage(hStatLineWnd, WM_USER + 1, 0, (LPARAM) (LPSTR) szMessage);
            if (sread(buf, 4, 4) == 4) {
            }
            return(TRUE);
        }
        return(FALSE);
}

BOOL NEAR PASCAL
Dial()
{
    char szTemp[MAXLEN_TEMPSTR];
    char szMessage[MAXLEN_TEMPSTR];
    char szConfFileName[MAXLEN_TEMPSTR];
    char *flds[10];
    int  i, j, kflds;

    * Obtiene el nombre del archivo de Configuración */
    GetFileIniName(hClientWnd, szConfFileName);
    * Obtiene la lista de teléfonos */
    GetPrivateProfileString("Modem", "DialNumber", "", szTemp, sizeof(szTemp), szConfFileName);
    * Copia cada teléfono a un elemento del arreglo */
    kflds = getargs(szTemp, flds);
    /* Restablece los parámetros del modem */
    ResetModem();
    /* Inicializa el modem */
    InitModem();
    Flush();
    /* Llama hasta que logra la conexión */
    for(i = 0; i    3; i++) {
        for(j = 0; j    kflds; j++) {
            if(DialModem(flds[j]))
                return(TRUE);
            Flush();
            WinSleep(2);
        }
    }
    /* Envía un mensaje a la ventana de información para que despliegue que no
      se logró la conexión
    */
    LoadString(GETHINST(hTTYWnd), IDS_CONN_F_MESSAGE, szMessage,
     sizeof(szMessage));
    SendMessage(hStatLineWnd, WM_USER + 1, 0, (LPARAM) (LPSTR) szMessage);
    return(FALSE);
}
```

Capítulo

# 4 Transferencia de Archivos en Windows

Este capítulo describe las dificultades para escribir protocolos de comunicación en Windows y la manera en que se adaptaron los fuentes del protocolo Kermit para UNIX.

***Contenido de este capítulo***



## Antecedentes

La tarea de programar un proceso demasiado largo en el ambiente de Microsoft Windows requiere consideraciones únicas. La programación en este ambiente no es parecida a la realizada en ambientes de una sola tarea (por ejemplo DOS) o a la de ambientes multitarea con un despachador de procesos por tiempo (por ejemplo UNIX). Windows está basado en eventos (procesamiento de mensajes) y en un sistema multitarea cooperativo (las aplicaciones liberan el control del CPU voluntariamente).

Los programas en Windows no se ejecutan hasta que reciben un mensaje, una vez recibido el mensaje debe ser procesado rápidamente para liberar el control del CPU y así permitir a otros programas la ejecución.

Programar un proceso en Windows que consume mucho tiempo como una secuencia ininterrumpida de código es un mal hábito, completamente insatisfactorio porque Windows no ejecutará ningún otro programa y el usuario no podrá realizar ninguna otra tarea durante el tiempo que dure este proceso.

Afortunadamente los procesos largos pueden ser divididos en una serie de procesos más pequeños, estos últimos deben ser atendidos en el menor tiempo posible cuando el programa tiene el control del CPU.

La transferencia de archivos entre dos computadoras es un buen ejemplo de un proceso de este tipo, porque sobre líneas de comunicación convencionales puede tomar algunas horas.

Una manera de escribir un programa Kermit involucra dividir el protocolo en una secuencia de tareas controladas por un autómata de estados finitos. Se describen las modificaciones de esta técnica que cumplieron con los requerimientos impuestos por este ambiente.

Esta implementación de Kermit fue integrada al emulador de terminal descrito en el capítulo anterior. El programa es capaz de enviar y recibir archivos binarios y en formato ASCII sobre líneas de comunicación con 7 u 8 bits de datos, empleando tres métodos de corrección de error y un método de compresión de datos (run-length) para eficiencia. Algunas extensiones del protocolo como modo servidor, paquetes de atributos de archivo y paquetes largos no son incluidas.

## El Protocolo Kermit

Kermit es un protocolo de comunicación punto a punto. Después de enviar datos, Kermit se detiene y espera el acuse de recibo de los datos que envió. Si no recibió el acuse de recibo esperado (o recibió un acuse de recibo negativo), Kermit vuelve a enviar los datos.

En una sesión de Kermit, el intercambio de información es a través del encapsulamiento de bloques de datos en varios tipos de paquetes que normalmente no exceden los 100 bytes en longitud, a menos que un paquete extendido este siendo usado (ver Figura 4.1).

El campo SOH identifica el inicio de un paquete. Este campo generalmente es un Ctrl-A y es el único carácter no imprimible en el paquete. El campo LEN contiene un contador del número de caracteres en el resto del paquete. El campo NUM indica el número de secuencia del paquete. El rango de números de esta secuencia es de 0 a 63 (después de 63, se vuelve a empezar en 0 ). El campo TYPE identifica el tipo de paquete. A continuación se muestra una lista de los tipos de paquetes:

| *Tipo de paquete* | *Descripción* |
|---|---|
| S | Inicio de archivo |
| F | Nombre de archivo |
| D | Datos |
| Z | Fin de archivo |
| B | Fin de transmisión |
| Y | Acuse de recibo positivo |
| N | Acuse de recibo negativo |
| E | Error |
| T | Límite de tiempo excedido (time-out) |
| Q | Error de CRC |

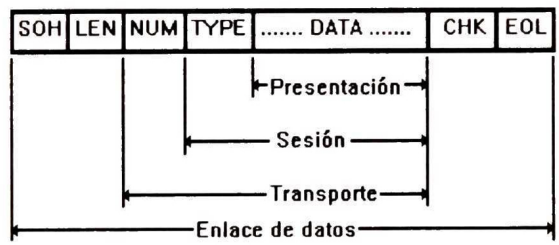
Hay otros tipos de paquetes que no son usados en esta implementación.

El campo DATA contiene datos. Debido a la restricción de que el paquete contenga solo caracteres imprimibles, para caracteres no imprimibles hay que realizar un proceso de conversión. El campo CHK (checksum) permite control de errores.

Aunque Kermit no es un protocolo con los niveles especificados por ISO, muchas versiones de Kermit son escritas basadas en este modelo para aislar funcionalidades y hacer más fácil su mantenimiento y extensiones.

*Figura 4 1*

*Formato de un paquete de Kermit*




**SOH** Inicio de encabezado (Usualmente Ctrl-A)
**LEN** Longitud en bytes hasta EOL
**NUM** Número de secuencia módulo 64
**TYPE** Tipo de paquete
**DATA** Bloque de datos
**CHK** CRC de 1, 2 ó 3 bytes
**EOL** Marca de fin de paquete

Tipos de paquetes

| | | | |
|---|---|---|---|
| S | Inicio de envío | N | Reconocimiento negativo |
| F | Nombre de archivo | E | Error |
| D | Datos | T | Time-out |
| Z | Fin de archivo | Q | Error de CRC |
| B | Fin de transmisión | | |
| Y | Reconocimiento | | |

## El Nivel de Sesión de Kermit

El nivel de sesión de Kermit es el mecanismo básico de control del protocolo. Cuando se envía o se recibe un archivo, el nivel de sesión toma acciones

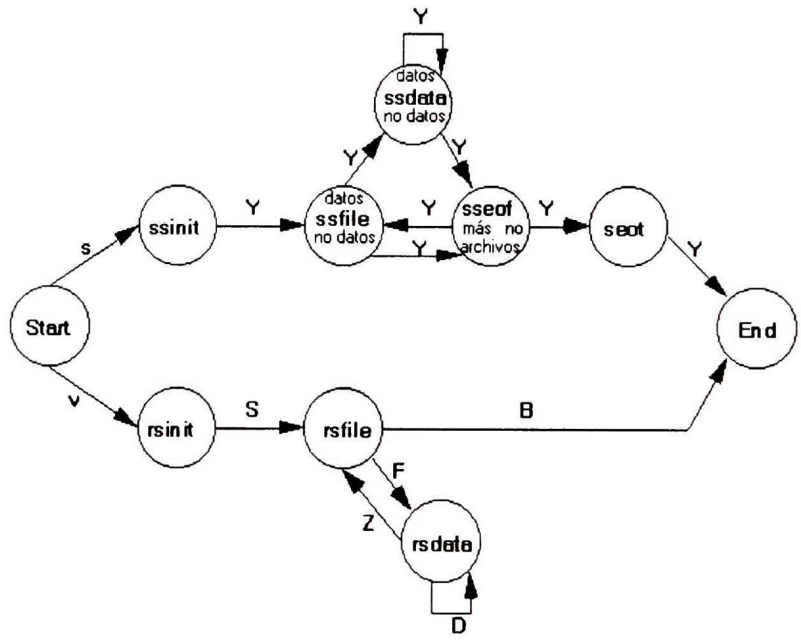
dependiendo del tipo de paquete recibido del Kermit remoto. Este nivel de sesión generalmente es modelado como un autómata de estados finitos.

Protocolos de comunicaciones como Kermit se especifican por medio de máquinas o autómatas de estados finitos. Existen herramientas como LEX y Wart, los cuales generan una tabla y un seguidor de máquinas de estados finitos. Estas herramientas toman como entrada la especificación de una máquina de estados finitos y entregan sus resultados como un programa en lenguaje C. El uso de estas herramientas está justificado por un ahorro de tiempo de programación y una generación de código sin errores. Una desventaja de las herramientas es que hay que aprender a usarlas.

En esta implementación se usó Wart. A continuación se explican algunas peculiaridades del código generado por Wart. Esta herramienta convierte la especificación del protocolo en una tabla que sirve para determinar el próximo estado del autómata. Este último entra en acción cuando es llamada la función wart. El estado actual es guardado en una variable estática, el tipo de paquete es obtenido por el llamado de la función wart a la función del nivel de transporte input. A su vez la función input llama a una función del nivel de enlace de datos, rpack, para obtener un paquete completo. El diagrama de la Figura 4.2 muestra los estados del protocolo Kermit.

*Figura 4.2*

*Estados del protocolo Kermit*




A continuación se presenta el código fuente del programa en Wart.

```
 * wnkerm.w
*

#define NOCOMM
#define NOKANJI
#define NOSOUND
#define NOATOM
#include   windows.h
#include   stdio.h
#include   string.h
#include "wnkerm.h"

#define RESUME return(0)
#define CONTINUE return(1)

static int near input(void);

 * Estados del protocolo *
%states rsinit rsfile rsdata
%states ssinit ssfile ssdata sseof sseot

 * Definición del protocolo */

%%%

 * Inicio de envío de archivos */
s {
   krm_tinit();
   Kermit.start = 'w';
}

 * Espera para enviar primer paquete */
w {
   if(Kermit.delay) {
      Kermit.start = 'w';
     CONTINUE;
   } else {
      /* transmite paquete sendinit */
     if (krm_sinit()    0) {
        krm_err(IDS_KRM_SENDINIT_ERROR);
        RESUME;
     }
      krmFlushQue();
      * Abre archivo y envía nombre de archivo */
     BEGIN ssinit;
   }
}
```

```
* Inicio de Recepción de archivos */
v {
  krm_tmit();
   * Se prepara para recibir paquete de inicio */
  BEGIN rsinit;
}

/* Recibe ACK del paquete de inicio de transmisión */
 ssinit  Y {
  krm_spar(krm_rcvpkt.data, krm_rcvpkt.len);
  /* Selecciona el tipo de CRC */
  Kermit.bctu = Kermit.bctr;
  /* Obtiene el nombre del archivo */
  Kermit.pFile = krm_getnextfile(TRUE);
  /* Envía paquete de nombre de archivo */
  if (krm_sfile()   0) {
    krm_err(IDS_KRM_SENDFILENAME_ERROR);
    RESUME;
  }
   * Se prepara para enviar primer paquete de datos */
  BEGIN ssfile;
}

/* Recibe ACK del paquete de nombre de archivo */
 ssfile  Y {
  int x;
  krm_savename();
   * Si no hay datos en el archivo */
  if((x = krm_sdata()) == 0) {
      * Envía paquete de fin de archivo */
    if(krm_seof("")   0) {
      krm_err(IDS_KRM_SENDEOF_ERROR);
      RESUME;
    }
    BEGIN sseof;
  }
  /* Hubo un error */
  else if (x   0) {
    krm_rclose(FALSE);
    krm_err(IDS_KRM_SENDDATA_ERROR);
    RESUME;
  }
  /* Se prepara para recibir paquetes de datos
  else BEGIN ssdata;
}

/* Recibe ACK del paquete de datos */
 ssdata  Y {
```

```
    * Checa si la transacción ha sido cancelada *
    krm_checkcnx();
    if (Kermit.abort) {
        * Cancela el archivo que se esta enviando *
        if(Kermit.abort == KRM_FILEABORT)
            Kermit.abort = 0;
        * Informa al servidor de la cancelación *
        if (krm_seof("D")    0) {
            krm_err(IDS_KRM_SENDEOF_ERROR);
            RESUME;
        }
        /* Se prepara para enviar paquete de fin de archivo */
        BEGIN sseof;
    }
    else {
        int x;
        * Si no hay mas datos *
        if((x = krm_sdata()) == 0) {
            * Envia paquete de fin de archivo */
            if(krm_seof("")    0) {
                krm_err(IDS_KRM_SENDEOF_ERROR);
                RESUME;
            }
            BEGIN sseof;
        }
        * Hubo un error *
        else if (x    0) {
            krm_rclose(FALSE);
            krm_err(IDS_KRM_SENDDATA_ERROR);
            RESUME;
        }
    }
}

* Recibe ACK del paquete de fin de archivo */
sseof  Y {
    * Si hay mas archivos para enviar */
    if(Kermit.pFile = krm_getnextfile(FALSE)) {
        * Envia el nombre del archivo siguiente */
        if (krm_sfile()    0) {
            krm_err(IDS_KRM_SENDFILENAME_ERROR);
            RESUME;
        }
        BEGIN ssfile;
    }
    /* En caso contrario envia paquete de fin de transmisión */
    else {
        if (krm_seot()    0) {
```

```
        krm_err(IDS_KRM_SENDEOT_ERROR);
        RESUME;
      }
      BEGIN sseot;
    }
}

* Recibe ACK del paquete de fin de transmisión */
 sseot  Y {
  krm_tend(IDS_KRM_TRANSACTION_DONE);
  RESUME;
}

/* Recibe paquete de inicio de envío */
 rsinit  S {
  BYTE data[KRM_MAXDATALEN + 1];
   * Lee datos de inicialización */
  krm_spar(krm_revpkt.data, krm_revpkt.len);
  /* Devuelve datos de inicialización en un paquete de ACK */
  krm_ack(krm_rpar(data), data);
  Kermit.bctu = Kermit.bctr;
   * Se prepara para recibir paquete de nombre de archivo */
  BEGIN rsfile;
}

/* Recibe paquete de nombre de archivo */
 rsfile  F {
   /* Trata de abrir el archivo */
  if (krm_revfil()) {
      char buf[KRM_MAXDATALEN + 1];
      int inlen, outlen;
      inlen = strlen(Kermit.pFile);
      outlen = krm_encode(buf, Kermit.pFile, sizeof(buf) - 1, &inlen);
     krm_ack(outlen, buf);
      /* Se prepara para recibir datos */
     BEGIN rsdata;
  }
  else {
      krm_err(IDS_KRM_FILE_OPEN_ERROR);
      RESUME;
  }
}

/* Recibe paquete de datos */
 rsdata  D {
  if (krm_revdata()) {
      switch(Kermit.abort) {
         case KRM_BATCHABORT:
```

```
            * Cancela los archivos que faltan */
            krm_ack(1, "Z");
            Kermit.abort = 0;
            break;
        case KRM_FILEABORT:
            * Cancela archivo actual *
            krm_ack(1, "X");
            Kermit.abort = 0;
            break;
        default:
            krm_ack(0, "");
    }
  }
  else {
    krm_rclose(TRUE);
    krm_err(IDS_KRM_FILE_WRITE_ERROR);
    RESUME;
  }
}

* Recibe paquete de fin de archivo */
rsdata  Z {
  if(krm_rclose(krm_rcvpkt.len &&
                (krm_rcvpkt.data[0] == 'D') ? TRUE : FALSE)) {
    * Reconoce el paquete */
    krm_ack(0, "");
     * Se prepara para el próximo archivo *
    BEGIN rsfile;
  }
  else {
    krm_err(IDS_KRM_FILE_CLOSE_ERROR);
    RESUME;
  }
}

* Recibe paquete de fin de transmisión */
rsfile  B {
  * Reconoce el paquete */
  krm_ack(0, "");
  krm_tend(IDS_KRM_TRANSACTION_DONE);
  RESUME;
}

* Paquete incompleto */
$ {
   * Continua hasta que el paquete este completo *
  CONTINUE;
}
```

```
* Recibe paquete de error *
E {
  krm_rclose(TRUE);
  krm_tend(KRM_ERROR_PACKET);
  RESUME;
}

/* Paquete no reconocido */
. {
  krm_rclose(TRUE);
  /* Envia error al servidor remoto */
  krm_err(IDS_KRM_UNKNOWN_PACKET);
  RESUME;
}

%%

 * input
  Proporciona el tipo de paquete a la maquina de estados
*/
static int near
input()
{
  register int type;

  if (Kermit.start) {
    type = Kermit.start;
    Kermit.start = 0;
  }
  else {
    type = krm_rpack();
    if((type != 'S') && (type != 'E')) {
      /* El paquete está completo y no es un paquete error */
      if((Kermit.seq != krm_rcvpkt.seq) || strchr("NQT", type)) {
        /* El paquete está corrompido */
        if (Kermit.retries  = KermParams.RetryLimit) {
          krmCreatePseudoPacket('E', PS_DONE, IDS_KRM_TOOMANYRETRIES);
          type = 'E';
        }
        else if ((type == 'N') &&
                    (krm_rcvpkt.seq == ((Kermit.seq + 1) & 63))) {
                      type = 'Y';
          Kermit.retries = 0;
        }
        else {
          /* Vuelve a enviar el paquete previo */
          Kermit.retries += 1;
```

```
                Kermit.totalretries += 1
                krm_resend();
                type = 'S';
              }
            }
            else Kermit.retries = 0;
          }
        }
        return (type);
}
```

En implementaciones de Kermit escritas para DOS o UNIX, una vez que wart es llamada no retorna hasta que la sesión es completada o abortada. De la misma manera input no retorna a wart hasta que un paquete formado propiamente con el número de secuencia correcto haya sido encontrado. A su vez, rpack no retorna a input hasta que un paquete completo con un checksum correcto sea obtenido o haya ocurrido un time-out.

Ninguno de estos esquemas de espera es aceptable en Windows, si wart es llamada como resultado de un mensaje y tiene el comportamiento anteriormente descrito, otras aplicaciones no se ejecutarán hasta que la transferencia de archivos se haya completado. Aún si wart retornara después de ejecutar la acción asociada con cada estado, se experimentaran largos retardos mientras rpack trata de leer un paquete completo de la línea de comunicaciones. Un paquete completo puede no llegar nunca.

Las acciones asociadas con cada estado son realizadas sin un retardo significativo porque involucran leer o escribir pequeños bloques de bytes de un archivo, codificar o decodificar estos datos, formar un paquete y escribirlos al buffer de comunicación.

La solución es violar el principio de los niveles y permitir al nivel de sesión reconocer un paquete incompleto cuya acción asociada es retornar desde wart. Este nuevo tipo de paquete y un mecanismo para construir un paquete incrementalmente, permiten al protocolo liberar el CPU a otros programas. Periódicamente la función que procesa los mensajes recibe un mensaje que indica que hay caracteres en el buffer de comunicación (WM_COMMNOTIFY), la acción correspondiente a este evento es copiar los datos del buffer de comunicación al buffer del protocolo y llamar a la función wart, la que a su vez llama a input. Si un paquete esta listo, la acción para ese estado es ejecutada y otra llamada a input es realizada. Esta vez input probablemente devuelva un paquete incompleto y wart devuelve el control a Windows. El resultado es que otras aplicaciones pueden ejecutarse.

## El Nivel de Enlace de Datos de Kermit

Después de que la función input llama a rpack para obtener el próximo paquete, rpack llama a getpacket, la que intenta, hasta donde es posible, construir un paquete completo. La función getpacket detecta el carácter de inicio de paquete (SOH) y usa el campo que indica la longitud del paquete (LEN) para determinar el momento en que el paquete está completo.

Cuando getpacket es llamada, es posible que el buffer de entrada no contenga los suficientes datos para construir un paquete completo y puede tomar varias llamadas a esta función antes de completar el paquete. Esta consideración fue tomada en cuenta en el diseño de getpacket. De nuevo se hizo a través de un autómata. La variable del estado actual se encuentra en una estructura global que contiene otros campos del paquete actual. De esta manera, getpacket y rpack no necesitan esperar a que lleguen más datos de la línea de comunicación y así la función input puede retornar inmediatamente a la función wart devolviendo un tipo de paquete incompleto.

A continuación se presenta el código fuente de las funciones krm_rpack y krm_getpacket

```
/* krm_rpack
  Construye un paquete de entrada
*/
int NEAR
krm_rpack(void)
{
  register int type;

  switch(krm_rcvpkt.state) {
    case PS_START:
      if (KermParams.Timer)
        SetTimer(Kermit.hWnd, KRM_WAITPACKET,
                 krm_sndinit.timeout * 1000, Kermit.fpTimer);
      krm_rcvpkt.state += 1;
    default:
      if (*krmBuflen)
        *krmBuflen = krm_getpacket(krmBufptr, *krmBuflen);
      if (krm_rcvpkt.state  PS_DONE) {
        type = '$';
        break;
      }
    case PS_DONE:
      krm_rcvpkt.state = PS_START;
      if (KermParams.Timer)
        KillTimer(Kermit.hWnd, KRM_WAITPACKET);
      type = krm_rcvpkt.type;
```

```
    }
    return type;
}

/* krm_getpacket
   Trata de construir hasta donde es posible un paquete completo
*/
static int NEAR
krm_getpacket(register BYTE *str, register int len)
{
    static BYTE buf[5];
    WORD chk;
    DWORD chk3;

    for ( ; len > 0; len--, str++) {
        switch(krm_rcvpkt.state) {
            case PS_SYNCH:
                if (*str == krm_rcvinit.mark) {
                    krm_rcvpkt.data_count = 0;
                    krm_rcvpkt.chk_count = 0;
                    krm_rcvpkt.data[0] = 0;
                    krm_rcvpkt.state++;
                }
                break;
            case PS_LEN:
                krm_rcvpkt.len = unchar(*str) - 2 - Kermit.bctu;
                if (krm_rcvpkt.len > KRM_MAXDATALEN)
                    krm_rcvpkt.len = KRM_MAXDATALEN;
                buf[0] = *str;
                krm_rcvpkt.state++;
                break;
            case PS_NUM:
                krm_rcvpkt.seq = unchar(*str);
                buf[1] = *str;
                krm_rcvpkt.state++;
                break;
            case PS_TYPE:
                krm_rcvpkt.type = *str;
                buf[2] = *str;
                buf[3] = 0;
                if (krm_rcvpkt.len)
                    krm_rcvpkt.state++;
                else
                    krm_rcvpkt.state = PS_CHK;
                break;

            case PS_DATA:
                if (krm_rcvpkt.data_count > krm_rcvpkt.len) {
```

```
            krm_rcvpkt.data[krm_rcvpkt.data_count++] = *str;
            break;
          }
          else {
            krm_rcvpkt.data[krm_rcvpkt.data_count] = NUL;
            krm_rcvpkt.state++;
          }
        case PS_CHK:
          if (krm_rcvpkt.chk_count    Kermit.bctu) {□
            krm_rcvpkt.rchksum[krm_rcvpkt.chk_count++] = *str;
            break;
          }
          switch(Kermit.bctu) {
            case 1:
            default:
              chk = krm_chk1(krm_chksum(krm_rcvpkt.data,
                                  krm_chksum(buf,0)));
              if (chk != (WORD)unchar(krm_rcvpkt.rchksum[0]))
                  krm_rcvpkt.type = 'Q';
                break;
            case 2:
                chk = ((WORD)unchar(krm_rcvpkt.rchksum[0])    6) |
                    (WORD)unchar(krm_rcvpkt.rchksum[1]);
                if (chk != krm_chksum(krm_rcvpkt.data,
                                  krm_chksum(buf,0)))
                  krm_rcvpkt.type = 'Q';
                break;
            case 3:
                chk3 = ((WORD)unchar(krm_rcvpkt.rchksum[0])    12) |
                        ((WORD)unchar(krm_rcvpkt.rchksum[1])    6) |
                        (WORD)unchar(krm_rcvpkt.rchksum[2]);
                if (chk3 != krm_chksum3(krm_rcvpkt.data,
                                              krm_chksum3(buf,0)))
                  krm_rcvpkt.type = 'Q';
                break;
          }
          krm_rcvpkt.state++;
          break;
        case PS_DONE:
          if (*str == krm_rcvinit.eol)
            len--;
          return len;
          break;
      }
    }
    return len;
}
```

## Detalles de Implementación

Se trató de que el módulo de Kermit fuera independiente del módulo del emulador de terminal. Sin embargo, el emulador de terminal no puede ignorar ciertas funciones de Kermit y además Kermit necesita algunas variables del emulador tales como los identificadores de la ventana y la instancia, el canal de comunicaciones y el buffer usado para leer del puerto serie. Se usó compilación condicionada para poder identificar los cambios hechos al código original. El protocolo requirió una interfaz con el usuario, así que se crearon tablas de cadenas de caracteres, menús, y algunos diálogos.

Todos los archivos que hagan referencias a variables de Kermit deben incluir el archivo WNKERM.H Este archivo consiste de prototipos de funciones, un menú, cadenas de caracteres y estructuras asociadas con diálogos o el protocolo. Todas las variables de Kermit son definidas en este archivo.

El código de Kermit espera que ciertas variables del emulador de terminal estén disponibles. En la inicialización, estas variables son copiadas a variables de Kermit. Una de estas variables es el identificador de la ventana, variable que es válida hasta que el programa termina. Otras variables pueden cambiar periódicamente, por ejemplo el identificador del puerto serie puede cambiar si el usuario selecciona un puerto diferente. El código de Kermit usa un apuntador a esta variable. Kermit debe también referenciar el buffer de comunicación y el contador de caracteres leídos, por lo tanto se usan apuntadores para este fin. El código de Kermit supone que una rutina del emulador de terminal lee caracteres del puerto serie y los deposita en este buffer y hace un llamado a la función wart.

A continuación se presenta el código que se fue incluido para que el emulador de terminal tuviera transferencia de archivos.

El archivo WNKERM.H debe ser incluido en un módulo del emulador de terminal, definiendo antes el símbolo KERMITEXTERN, para declarar globales a las variables de Kermit.

```
#if defined(KERMIT)
#define KERMITEXTERN
#include "wnkerm.h"
#endif
```

Al momento de inicialización (Puede ser cuando se procesa el mensaje WM_CREATE)

```
#if defined(KERMIT)
```

```
  if(!krmInit(hWnd, Buffer, &Buflen, &cid))
    PostMessage(hTTYWnd, WM_SYSCOMMAND, SC_CLOSE, 0L);
#endif
```

Cuando la ventana principal del emulador de terminal es creada, se hace una llamada a la función krmInit y se le pasa el identificador de la ventana, información del buffer de comunicación y la dirección del identificador del puerto de comunicaciones. Estas variables cambian dinámicamente, con excepción del identificador de la ventana, por lo que se usa apuntadores a estas variables. Esta función lee el archivo de configuración para determinar los parámetros del protocolo y de los paquetes. El menú de Kermit es agregado al final del menú del emulador de terminal.

Al momento de terminación del programa (Generalmente es cuando se procesa el mensaje WM_DESTROY)

```
#if defined(KERMIT)
  krmShutdown();
#endif
```

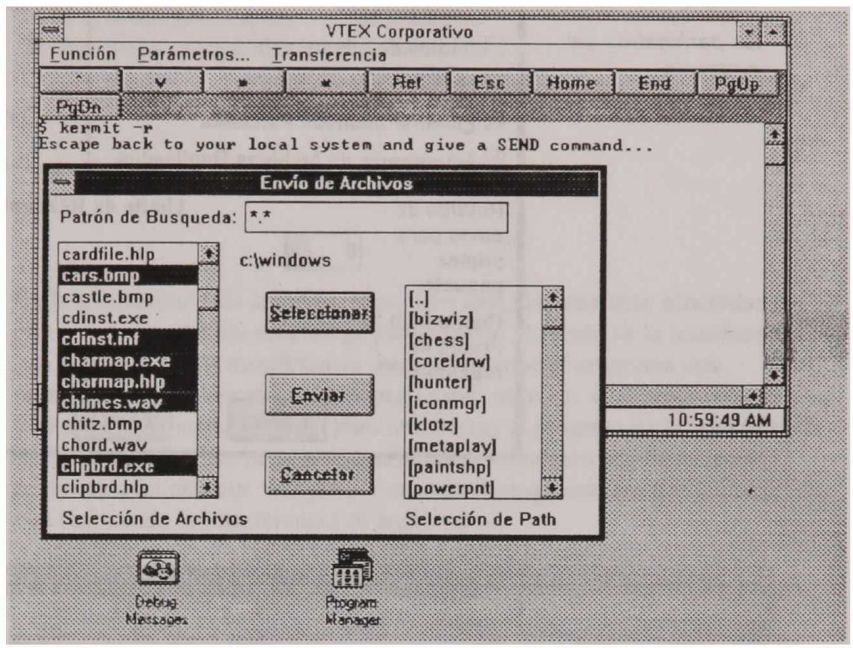
Cuando el programa termina, los recursos globales usados por Kermit deben ser liberados. La función krmShutdown realiza esta acción.

En la función que salva opciones en el archivo de configuración.

```
#ifdef KERMIT
    krmProtocolSave();    /* Salva opciones del protocolo */
    krmPacketsSave();     /* Salva opciones de los paquetes del protocolo */
#endif
```

*Figura 4 3*

*La interfaz permite seleccionar múltiples archivos y enviarlos en grupo*




En la función que procesa los mensajes WM_COMMAND

```
#if defined(KERMIT)
  if(krmWndCommand(hWnd, wParam)) break;
#endif
```

El menú de Kermit permite: enviar un grupo de archivos (ver Figura 4.3), recibir un grupo de archivos, cancelar una transferencia mediante 3 opciones diferentes y configurar parámetros y paquetes del protocolo (ver Figura 4.4 y 4.5). La función krmWndCommand regresa TRUE si una opción del menú fue reconocida. De otra manera regresa FALSE permitiendo al emulador de terminal procesar este mensaje.

*Figura 4.4*

*Interfaz para cambiar los parámetros del protocolo*

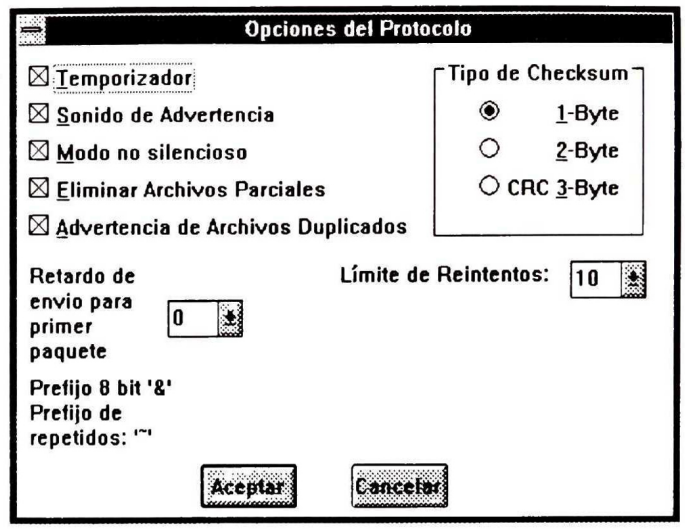
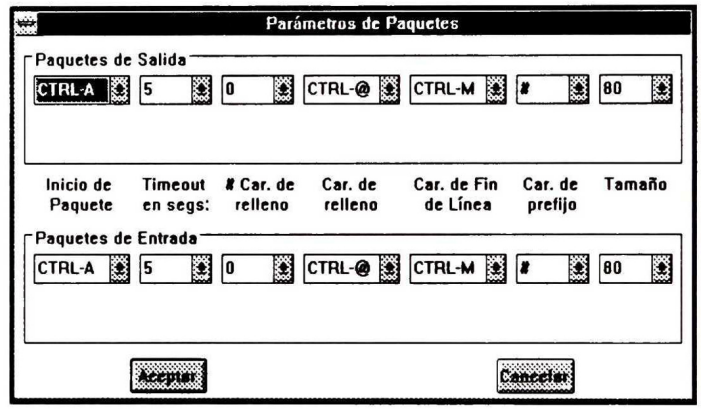



*Figura 4.5*

*Interfaz para cambiar los parámetros de los paquetes*




En la función que procesa mensajes

```
#if defined(KERMIT)
    case WM_QUERYDRAGICON:
      return MAKELONG(krmIcon, 0);
#endif
```

En la función que procesa el mensaje WM_PAINT

```
#if defined(KERMIT)
  if(IsIconic(hWnd)) {
    if(Kermit.InTransfer) krmPaint(hWnd, ps.hdc);
```

```
    else {
      RECT rect;
      GetClientRect(hWnd, &rect);
      DrawIcon(ps.hdc, (rect.right - GetSystemMetrics(SM_CXICON)) / 2,
           (rect.bottom - GetSystemMetrics(SM_CYICON)) / 2, krmIcon);
    }
  }
  else
#endif
```

En la transferencia de archivos, se ocultan todas las ventanas generadas por este programa y se usa un dialogo para mostrar el estado de la transferencia (ver Figura 4.6). La transferencia continúa aunque el programa este minimizado. El programa tiene control sobre su icono, si el programa corre en background, el icono es usado para monitorear el progreso de la transferencia (ver Figura 4.7). Este manejo es realizado en la función krmPaint como respuesta a un mensaje WM_PAINT cuando el programa esta minimizado y se está realizando la transferencia de archivos.

*Figura 4.6*

*Transferencia de archivos*

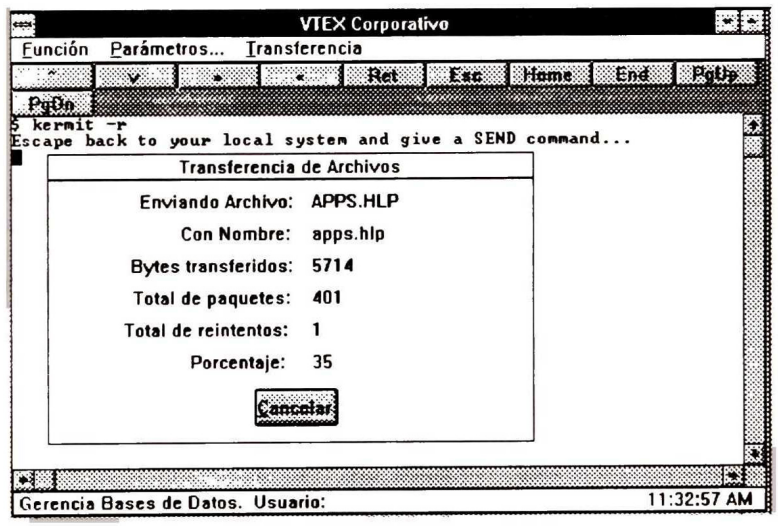
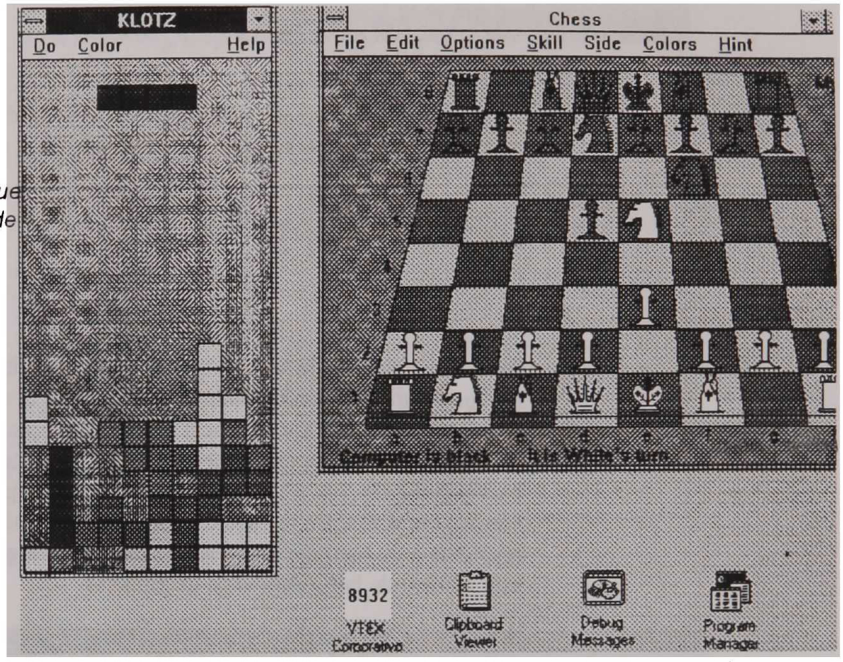


*Figura 4 7*

*La transferencia de archivos puede ser realizada en background (Ver el icono del programa que muestra el contador de paquetes)*




El mensaje WM_COMMNOTIFICATION es enviado a la aplicación por el manejador del puerto serie cuando un evento relacionado con el puerto ocurre. Hay eventos asociados al cambio de estado de algunas señales del RS232 (por ejemplo DSR, CTS, CD), eventos asociados con errores ocurridos en la comunicación (por ejemplo paridad, trama, sobre escritura), eventos relacionados con el estado de las colas de los buffers de transmisión y recepción y un evento para indicar que se recibió un carácter y se puso en la cola (EV_RXCHAR). En esta aplicación solo se maneja este último evento.

Las acciones asociadas con el mensaje WM_COMMNOTIFICATION dependen del modo de operación del programa, modo terminal o modo transferencia. En modo terminal los datos son leídos del puerto, son interpretados y son desplegados. En modo transferencia los datos son procesados por la función wart.

El método seleccionado para la operación de los dos modos fue usar subclases. Las subclases son usadas para modificar el comportamiento de una clase. Una subclase cambia la función que procesa los mensajes.

Las funciones que atienden los mensajes para la ventana de la aplicación son: ClientWndProc en modo terminal y ClientWndSubProc en modo transferencia.

Antes de iniciar la transferencia se llama a la siguiente función para cambiar el procesamiento de los mensajes

```
SetWindowLong(hClientWnd, GWL_WNDPROC, (LONG)ClientWndSubProc);
```

Y cuando termina la transferencia se vuelve a llamar para procesar los mensajes como al inicio

```
SetWindowLong(hClientWnd, GWL_WNDPROC, (LONG)ClientWndProc);
```

Esta es la función que procesa mensajes cuando el programa esta en modo transferencia

```
LRESULT FAR PASCAL
ClientWndSubProc(HWND hWnd, UINT wMsg, WPARAM wParam, LPARAM lParam)
{

  switch(wMsg) {
    case WM_COMMNOTIFY:
      SubProcessCOMMNotification(hWnd, (WORD) wParam, (LONG) lParam);
      return(0L);
    default:
      return(CallWindowProc(fpMainWndProc, hWnd, wMsg, wParam, lParam));
  }
}
```

Esta es la función que procesa el mensaje WM_COMMNOTIFY para la transferencia de archivos

```
BOOL NEAR
SubProcessCOMMNotification(HWND hWnd, WORD wParam, LONG lParam)
{
  COMSTAT ComStatus;
  static int OldBuflen;
  register int result = 0;
  register WORD room = BUFSIZE - Buflen;
  NPTTYINFO  npTTYInfo;

  if(CN_EVENT & LOWORD(lParam) != CN_EVENT)
    return(FALSE);
  if(NULL == (npTTYInfo = (NPTTYINFO) GetWindowWord(hWnd, GWW_NPTTYINFO)))
    return(FALSE);
  GetCommEventMask(COMDEV(npTTYInfo), EV_RXCHAR);
  if((Buflen    0) && (Buflen    OldBuflen))
    memmove(Buffer, Buffer + OldBuflen - Buflen, Buflen);
  if(room    0) {
```

```
    if(GetCommError(COMDEV(npTTYInfo), (COMSTAT FAR *)&ComStatus) == 0) {
      if(ComStatus.cbInQue) {
        result = ReadComm(COMDEV(npTTYInfo), Buffer + Buflen,
                  min(ComStatus.cbInQue, room));
        OldBuflen = Buflen += abs(result);
#if defined(KERMIT)
        if(Kermit.InTransfer) wart();
#endif
      }
    }
  }
  return(TRUE);
}
```

La función SubProcessCOMMNotification prevé el caso cuando el buffer contenga datos leídos del puerto de comunicación que aún no han sido procesados. La variable global Buflen es usada para indexar el buffer. Hay que hacer notar que el buffer es lineal, por lo que los caracteres que no han sido procesados son movidos al inicio del buffer antes de llamar a la función ReadComm.

*Capítulo*

# 5 Operación sobre Red

En este capítulo se describe el modelo ISO-OSI, la familia de protocolos TCP/IP, las especificaciones desarrolladas para la interfaz de red, el protocolo telnet, la implementación de este último sobre Windows y los cambios realizados al emulador de terminal descrito en el capítulo 3.

***Contenido de este capítulo***

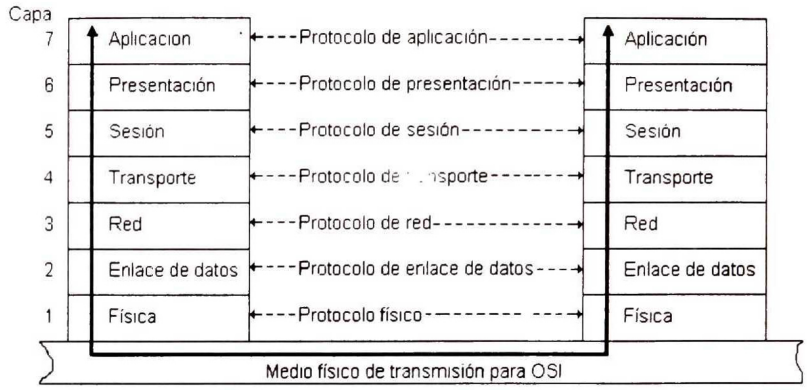


## Antecedentes

Muchas de las redes de computadoras actuales están basadas en el modelo de Interconexión de Sistemas Abiertos (OSI) desarrollado por la Organización Internacional de Normas (ISO) en 1978.

*Figura 5.1*

*El modelo de referencia OSI*




El modelo separa software y hardware en capas (Ver figura 5.1). Está basado en dos principios. El primero es comunicación par a par. Cada capa asume que se comunica con su capa similar que se encuentra en una máquina remota. Cada capa ignora a las otras capas del equipo remoto.

El segundo principio es que cada capa proporciona un servicio a la capa superior. Las capas se comunican entre ellas por medio de una interfaz y se ocultan los detalles de la implementación.

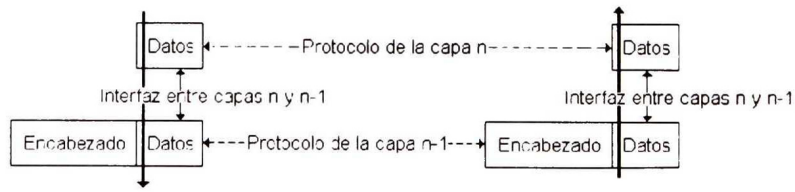
La comunicación par a par es realizada a través de la encapsulación de datos. Por ejemplo, una aplicación quiere enviar datos a una aplicación que se encuentra en una computadora remota. La aplicación emisora entrega los datos a la capa de aplicación, la cual añade un encabezado y entrega el elemento resultante o trama a la capa de presentación.

La capa de presentación transforma esta trama, generalmente le agrega datos de control y un encabezado. La trama resultante se la entrega a la capa de sesión.

Este proceso se sigue repitiendo hasta que los datos alcanzan la capa física, lugar en donde efectivamente se transmiten a la computadora donde se encuentra la aplicación receptora. En esta computadora, se van quitando uno a uno los encabezados, a medida que los datos se transmiten a las capas superiores, hasta que finalmente llegan a la aplicación receptora. Este proceso se muestra en la figura 5.2.

*Figura 5.2*

*Encapsulación de datos entre capas*



Las capas pueden ofrecer dos tipos diferentes de servicios a las capas que se encuentran sobre ellas: uno orientado a conexión y otro sin conexión. El servicio orientado a conexión establece y mantiene un circuito virtual entre el emisor y el receptor, similar a la conexión por teléfono entre dos personas.

El servicio sin conexión se modela con base en el sistema postal. Cada mensaje lleva consigo la dirección y cada uno de ellos se encamina, en forma independiente a través del sistema. Normalmente, cuando dos mensajes se envían al mismo destino, el primero que se envía será el primero en llegar. Es posible, sin embargo, que el primero sufra un retardo y llegue después al enviado en segundo lugar. Con un servicio orientado a conexión es imposible que suceda esto.

## Capas del Modelo de Referencia

La primera capa del modelo de referencia, la capa física, es responsable de la transmisión de datos sobre un medio físico de comunicación tales como: micro ondas, par trenzado o cable coaxial. La topología de la red, por ejemplo ethernet o token ring, es también parte de la capa física. El servicio proporcionado por la capa física es propenso a errores. Ejemplos de esta capa incluyen el medio de transmisión o la tarjeta de interfaz de red.

La segunda capa, la capa de enlace de datos, asegura que la transmisión de datos sobre la capa física este libre de errores, por medio de la implementación de técnicas de control de errores como: secuencias de paquetes y reconocimientos de paquetes. Esta capa consiste de dos subcapas: MAC (control de acceso al medio de transmisión) y LLC (control del enlace lógico). Ejemplos de la segunda capa son los protocolos siguientes: ethernet, IEEE 802.5, token ring y SLIP.

La capa de red determina la ruta completa de la red entre dos estaciones de comunicación, las cuales se encuentran en la misma LAN o a nivel mundial en una ambiente tipo Internet.

La función principal de la capa de transporte consiste en aceptar los datos de la capa de sesión, dividirlos, siempre que sea necesario, en unidades más pequeñas, pasarlos a la capa de red y asegurar que todos ellos lleguen correctamente al otro extremo.

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

La capa de presentación tiene que ver con la representación de los datos y con la transformación de los mismos, cuando los datos viajan entre plataformas diferentes, como sería entre una Digital VAX y una IBM System/360.

La capa de aplicación, es donde residen aplicaciones como: transferencia de archivos, correo electrónico, o emuladores de terminal, que usan los servicios que les proporcionan las capas inferiores.

## TCP/IP

TCP/IP es un conjunto de protocolos y programas usados para interconectar redes de computadoras. El modelo TCP/IP está compuesto por cuatro capas de software como se muestra en la figura 5.3.

*Figura 5.3*

*El modelo TCP/IP*

| Capa | Nombre |
|---|---|
| 4 | Aplicación |
| 3 | Transporte |
| 2 | Interconexión de redes |
| 1 | Interfaz de red |

El conjunto o la familia de protocolos TCP/IP (TCP/IP suite) describen manejo de errores, transferencia de mensajes y normas de comunicación. Las computadoras que usan TCP/IP se pueden comunicar entre si, no importando las diferencias de hardware y software, por ejemplo: una PC con sistema operativo DOS puede accesar información que se encuentra en una HP 3000 o con una IBM 3090.

La familia de protocolos TCP/IP pueden funcionar sobre líneas directas o conmutadas, redes ethernet, token ring, X.25, SNA, DECNET y algunas más.

TCP/IP proporciona diversos servicios, incluyendo correo electrónico, transferencia de archivos y conexión remota.

## El Protocolo de Interconexión de Redes (IP)

El protocolo entre redes, IP, define un sistema de entrega de datos, donde las máquinas fuente y destino no están necesariamente directamente conectadas. IP divide los datos en paquetes de cierto tamaño, los cuales son enviados a la máquina receptora a través de la red. Estos paquetes individuales de datos (a menudo llamados datagramas) son enrutados por diferentes máquinas de la redes a la red de destino y a la máquina receptora. Un grupo de datos, como podría ser un archivo, debe ser dividido en varios datagramas, los que son enviados por separado. IP es un protocolo sin conexión, datagramas individuales pueden no llegar y probablemente no lleguen en el orden en el

cual fueron enviados. El protocolo TCP proporciona la confiabilidad que le hace falta a IP.

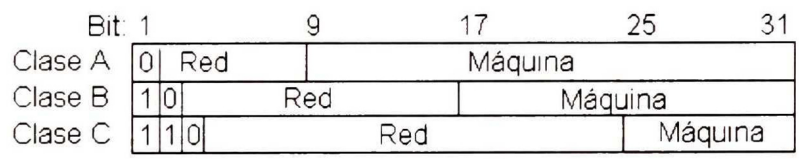
Un datagrama consiste de un encabezado de información y un área de datos. El encabezado es usado para rutear y procesar el datagrama. Los datagramas pueden ser divididos en unidades más pequeñas, dependiendo de los requerimientos físicos de la red . Por ejemplo, cuando un gateway envía un datagrama a una red que no puede procesarlo como un paquete único, el datagrama debe ser dividido en piezas lo suficientemente pequeñas para poder ser transmitido. Los encabezados de los fragmentos del datagrama contienen información necesaria para ensamblar los fragmentos. Los fragmentos no necesariamente llegan en orden; el software que implementa el protocolo IP en la máquina destino debe unir los fragmentos para obtener el datagrama original. Si alguno de los fragmentos no llega a su destino, el datagrama es descartado.

## Direccionamiento de la Redes

Dentro del encabezado de IP se encuentran la dirección fuente y la dirección de destino. Estas direcciones indican el número de red y de máquina. Como se muestra en la figura 5.4, son cuatro formatos diferentes los que en general se emplean. Los cuatro esquemas permiten hasta 128 redes con 16 millones de máquinas; 16 384 redes con hasta 64K de máquinas; 2 millones de redes, presuntamente redes tipo LAN, con hasta 256 máquinas cada una.

*Figura 5.4*

*Formatos de las direcciones IP*




Por convención, las direcciones están en una notación especial, de tal manera que la dirección es una lista de 4 números decimales separados por un punto. Por ejemplo: 100.0.0.51 y 128.10.2.1 pertenecen a la clase A y B, respectivamente.

## El Protocolo de Control de la Transmisión (TCP)

El protocolo de control de la transmisión, TCP, trabaja con IP para proporcionar una entrega confiable. Proporciona un medio que asegura que los datagramas que componen un mensaje sean ensamblados en el orden correcto y que cualquier datagrama perdido sea retransmitido hasta que sea recibido correctamente.

El principal propósito de TCP es evitar la perdida, daño, o duplicidad de paquetes que puede ocurrir con IP.

TCP proporciona confiabilidad usando códigos para detección de error, números de secuencia en los encabezados, reconocimientos positivos de paquetes recibidos y retransmisión de paquetes perdidos o dañados.

## Otros Protocolos de la Familia TCP/IP

Aunque la familia de protocolos es mencionada como TCP/IP, existen otros miembros además de TCP e IP. A continuación se listan los protocolos restantes más importantes.

| Protocolo | Propósito |
|---|---|
| Protocolo de resolución de direcciones (ARP) | Traduce direcciones IP a direcciones físicas. |
| Protocolo de control de mensajes entre redes (ICMP) | Usado para mensajes de error y verificación de datos. |
| Protocolo simple de transferencia de correo (SMTP) | Usado para transferencia de correo electrónico. |
| Protocolo de datagramas de usuario (UDP) | Capa de transporte orientada a no conexión |

## Aplicaciones TCP/IP

Existen varias aplicaciones TCP/IP que proporcionan al usuario facilidades de la red. A continuación aparece una lista parcial de estas aplicaciones:

| Aplicación | Propósito |
|---|---|
| ftp | Transferencia de archivos entre máquinas con TCP/IP; estas máquinas pueden no tener el mismo sistema operativo. |
| rcmd | Ejecución de comandos remotos entre máquinas UNIX. |
| rcp | Copia de archivos entre máquinas UNIX. |
| rlogin | Servicio de conexión remota entre máquinas UNIX. |
| rwho | Muestra una lista de usuarios conectados en las máquinas de la red. |
| telnet | Servicio conexión remota y de terminal virtual; estas máquinas pueden no tener el mismo sistema operativo. |

## Interfaz de Red

Una de las funciones de un sistema operativo es asignar recursos y dispositivos a los programas que los necesitan. Desafortunadamente, en el ambiente de red de una PC, el sistema operativo no puede comunicarse con la tarjeta de interfaz de red que conecta a la PC con la red local.

Las aplicaciones de red deben usar una capa de software intermedia para comunicarse con la tarjeta de red. Esta capa de software realiza las funciones del control de acceso del medio de transmisión de la capa de enlace de datos, por lo que es conocida como el manejador MAC. Este último oculta los detalles de las tarjetas de red del protocolo de comunicación que las usa, como se muestra en la Figura 5.5.

*Figura 5 5*

*El Manejador de la capa MAC*


Algunos protocolos de comunicación incluyen el manejador MAC, causando dos problemas: Primero, con la variedad de tarjetas de red existentes en el mercado, es difícil escribir una implementación de un protocolo que pueda funcionar con todas las tarjetas disponibles. Además, si dos tipos de protocolos de red están accesando a la misma tarjeta (por ejemplo Lan Manager y TCP/IP), existe una competencia por el uso de la tarjeta, pudiendo causar una falla al sistema operativo.

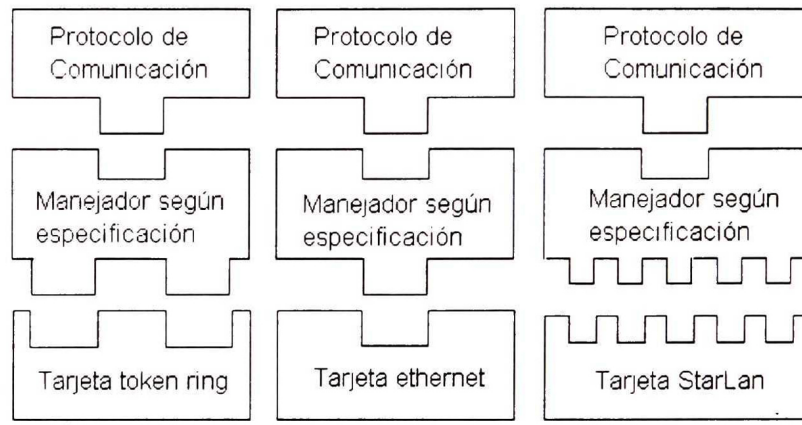
Por esta razón, se crearon especificaciones de manejadores de tarjetas de red para proporcionar una interfaz común entre la tarjeta de red y las capas superiores de los protocolos. En 1985, IBM desarrolló la primera versión de la especificación ASI para token ring. En 1987, FTP Software desarrolló la especificación de Packet Drivers para una variedad de tipos de redes. Posteriormente Microsoft y 3Com, y luego Novell, desarrollaron especificaciones para sus redes.

Un protocolo de comunicación diseñado para usar alguna de estas especificaciones puede funcionar con cualquier tarjeta de red que cuente con un

manejador para esta especificación. Si un desarrollador de software construye un producto usando una de estas especificaciones, obtiene compatibilidad con un gran número de tarjetas de red, como se muestra en la Figura 5.6

*Figura 5.6*

*Las especificaciones para manejadores de tarjetas de red proporcionan compatibilidad con la gran variedad de tarjetas de red*




Estas especificaciones también proporcionan una interfaz que permite que varios tipos de protocolos de comunicación accesen a la misma tarjeta de red al mismo tiempo (por ejemplo, es posible tener Netware y TCP/IP o dos productos que usen Packet Driver).

La especificaciones de manejadores MAC más importantes son:

- Packet Driver
- NDIS (Especificación de interfaces de manejadores de red, usada por Lan Manager)
- ASI (Interfaz para soporte de adaptadores)
- ODI (Interface abierta para enlace de datos, usada por Netware)
- DLL (Capa de enlace de datos, usada por DECnet PCSA)

## Packet Driver

La especificación Packet Driver define una interfaz compartida de software con el hardware de comunicación como: una tarjeta ethernet, una tarjeta token ring o una tarjeta serial.

Esta especificación es ampliamente aceptada en la comunidad de Internet y con los usuarios de TCP/IP para DOS. Muchos productos comerciales la soportan, entre ellos: FTP Software PC/TCP, Beame & Whiteside BW-NFS, Wollongong Pathway Access, y Sun PC-NFS.

El Packet Driver trabaja como un módulo residente en memoria. Este módulo usa una interrupción de software (como la 0x60). Cualquier capa superior que quiera accesar al Packet Driver debe generar esta interrupción.

El Packet Driver realiza todas las operaciones de entrada/salida de la red que necesita una aplicación. La aplicación no necesita conocer la tarjeta que usa, el puerto de entrada/salida de la tarjeta, interrupción, u otros detalles de hardware.

## Redes Ethernet

La red ethernet es una red de tipo bus, porque todas las interfaces se conectan a un mismo canal de comunicaciones- un cable coaxial. Todas las interfaces reciben cada transmisión. Su esquema de acceso es por un acceso múltiple por detección de portadora con detección de colisión (CSMA/CD), varias máquinas pueden accesar la red simultáneamente y cada máquina monitorea una portadora para determinar el momento en que el canal esta ocioso.

Cada interfaz (tarjeta de red) tiene una dirección de 48 bits. Las direcciones están asociadas a la interfaz.

El tamaño de las tramas ethernet varía de 64 a 1536 octetos (ver figura 5.7).

*Figura 5.7*

*Formato de la trama ethernet*

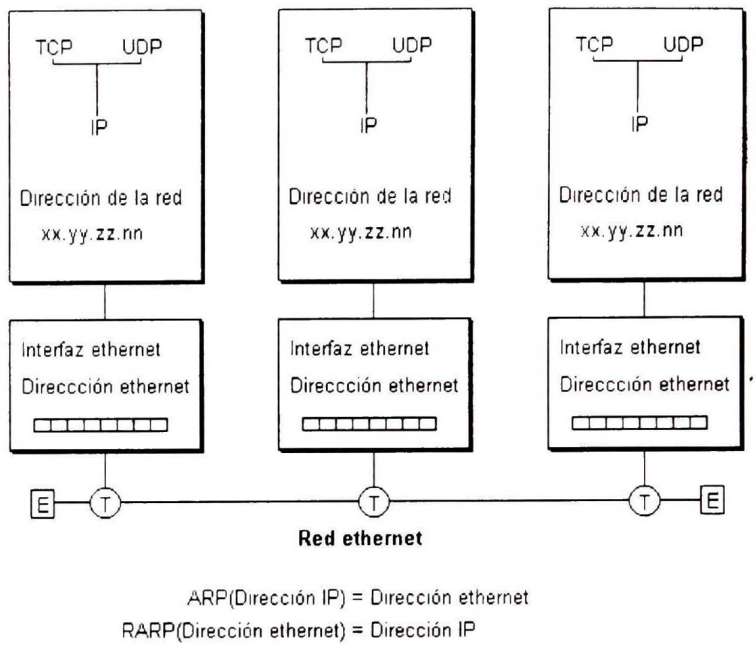
| Dirección fuente | Dirección destino | Tipo | . Datos. . | CRC |
|---|---|---|---|---|

Cada encabezado de trama incluye las direcciones fuente y destino, y un campo de 16 bits para identificar el contenido de la trama y el formato de los datos. El hardware adiciona a cada trama un CRC de 32 bits para detectar errores de transmisión. El ancho de banda para redes ethernet es de 10 megabits por segundo.

La figura 5.8 muestra la configuración de una red ethernet típica. Cada máquina tiene al menos un controlador ethernet conectado a un cable coaxial por medio de un conector T. El bus tiene en sus extremos un terminador (Indicado por la letra E en la figura).

*Figura 5.8*

*Configuración típica de una red ethernet*




## El Protocolo Telnet

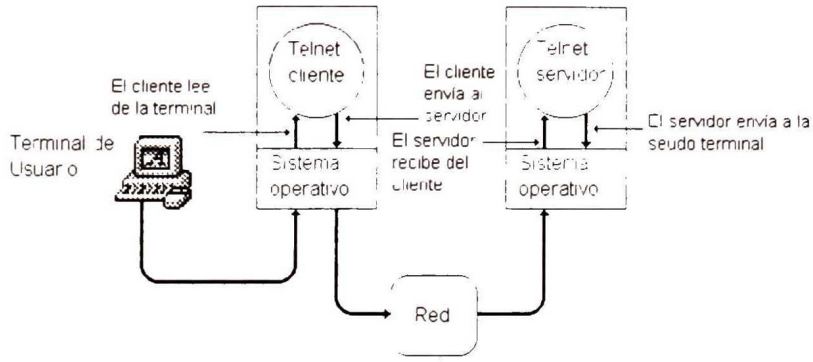
La familia TCP/IP incluye un protocolo simple para emulación de terminal llamado telnet. Telnet permite establecer una conexión TCP a un servidor, y simular que la terminal del usuario es una terminal de la máquina remota.

Telnet ofrece dos servicios básicos. Primero, define una terminal virtual de red (NVT), la cual proporciona una interfaz estándar a los sistemas remotos. Segundo, incluye un mecanismo que permite al programa cliente y al programa servidor negociar opciones, y proporciona un conjunto estándar de opciones.

La figura 5.9 ilustra la implementación de los programas telnet cliente y telnet servidor.

*Figura 5 9*

*Trayectoria de los datos de la terminal del usuario hasta el sistema operativo remoto*




El proceso telnet cliente es el programa telnet de la máquina del usuario. Cuando se ejecuta, el cliente establece una conexión TCP con el servidor que se desea comunicar. Una vez que la conexión se ha establecido, el cliente lee los caracteres que le llegan del teclado y se los envía al servidor, mientras concurrentemente acepta caracteres que el servidor le envía y los muestra en la terminal del usuario. El servidor debe aceptar las conexiones TCP de los clientes y transmitir los datos que le llegan de la conexión TCP al sistema operativo.

En la práctica el servidor es más complejo porque debe manejar múltiples conexiones. Usualmente, un proceso maestro espera una nueva conexión y crea un nuevo proceso esclavo para manejar cada conexión. La Figura 5.9 muestra el servidor telnet que maneja una conexión particular. La figura no muestra el proceso maestro que espera nuevas peticiones, ni tampoco muestra a los procesos esclavos que manejan otras conexiones.

## Comandos

La NVT de telnet define la siguiente lista de funciones de control.

| Señal | Significado |
|---|---|
| IP | Interrupción de un proceso |
| AO | Aborta la salida |
| AYT | Are You There (prueba si el servidor esta respondiendo) |
| EC | Borra el carácter previo |
| EL | Borra la línea actual |
| SYNCH | Sincroniza |
| BRK | Señal de atención |

Para enviar las funciones de control a través de la conexión TCP, telnet usa una secuencia de escape. La secuencia de escape usa un byte reservado para indicar que el próximo carácter es un código de control. El byte reservado que inicia una secuencia de escape es conocido como *interpretar como comando* (IAC). A continuación se muestra una lista de los posibles comandos y su código decimal.

| *Comando* | *Código* | *Significado* |
|---|---|---|
| IAC | 255 | Interpreta próximo byte como comando |
| DON'T | 254 | Negación a la petición para realizar opción especificada |
| DO | 253 | Aprobación para permitir opción especificada |
| WONT | 252 | Negativa para realizar opción especificada |
| WILL | 251 | Acuerdo para realizar opción especificada |
| SB | 250 | Inicio de subnegociación |
| EL | 248 | Señal para borrar línea |
| EC | 247 | Señal para borrar carácter |
| AYT | 246 | Señal "are you there" |
| AO | 245 | Señal para abortar la salida |
| IP | 244 | Señal para interrumpir el proceso |
| BRK | 243 | Señal de atención |
| NOP | 241 | No operación |
| SE | 240 | Fin de subnegociación |
| EOR | 239 | Fin de registro |

La tabla anterior muestra la correspondencia de las señales generadas por las teclas conceptuales y sus comandos. Por ejemplo, para pedir que el servidor interrumpa el programa actual, el cliente debe enviar una secuencia de dos bytes: IAC IP (255 seguido por 254). Los comandos adicionales permiten al cliente la negociación de opciones a usar y la sincronización de la comunicación.

## Opciones de Telnet

La descripción que se ha hecho de telnet omite uno de los aspectos más complejos: las opciones. Las opciones son negociables, lo que hace posible que el cliente y el servidor reconfiguren su conexión. Por ejemplo, se puede configurar la conexión para que acepte todos los caracteres ASCII (8 bits).

El rango de las opciones es amplio. Una de las opciones controla la operación de transmisión y recepción al mismo tiempo (full duplex) o una actividad a la vez (half duplex). Otra opción permite al servidor determinar el tipo de terminal del usuario.

A continuación se muestra una lista de las opciones más comunes.

| *Nombre* | *Código* | *Significado* |
|---|---|---|
| Transmisión Binaria | 0 | Cambia la transmisión a 8 bits |
| Echo | 1 | Permite mostrar los datos que se reciben |
| Estado | 5 | Petición para conocer el estado actual de telnet |
| Tipo de terminal | 24 | Intercambia información acerca del modelo de la terminal del usuario |

## Negociación de Opciones

Telnet usa una negociación simétrica de opciones que permiten a servidores y clientes reconfigurar los parámetros que controlan la conexión. Todas las implementaciones de telnet entienden un protocolo básico, razón por la que es posible que trabajen versiones más recientes o más sofisticadas con versiones anteriores o menos sofisticadas.

La petición WILL X, significa *¿estarías de acuerdo en permitirme usar la opción X?*; y la respuesta es DO X o DON'T X, que significan *estoy de acuerdo en permitirte usar la opción X* o *no estoy de acuerdo en permitirte usar la opción X*. El comando DO X pide que se empiece a usar la opción X, y las respuestas WILL X o WON'T X significan *iniciaré usando la opción X* o *no iniciaré usando la opción X*.

## Referencias del Internet acerca de Telnet

Mucha de la información escrita acerca de TCP/IP, incluyendo arquitectura, protocolos e historia, puede ser encontrada en una serie de reportes conocidos como Solicitudes Para Comentarios o RFCs (Request For Comments).

Estos RFCs se encuentran disponibles en la Internet. La Internet es una red de redes de computadoras basadas en el protocolo TCP/IP.

Al inicio de la década de los 70s la Internet estaba compuesta por pocas computadoras, por 100 en 1980, 1000 en 1984, 10000 en 1987, 100000 en 1989 y cerca de un millón al final de 1992.

A continuación se listan los RFCs más importantes acerca de telnet.

| *Número de RFC* | *Descripción* |
|---|---|
| 854 | Especificación del protocolo telnet |
| 855 | Especificaciones de las opciones de telnet |

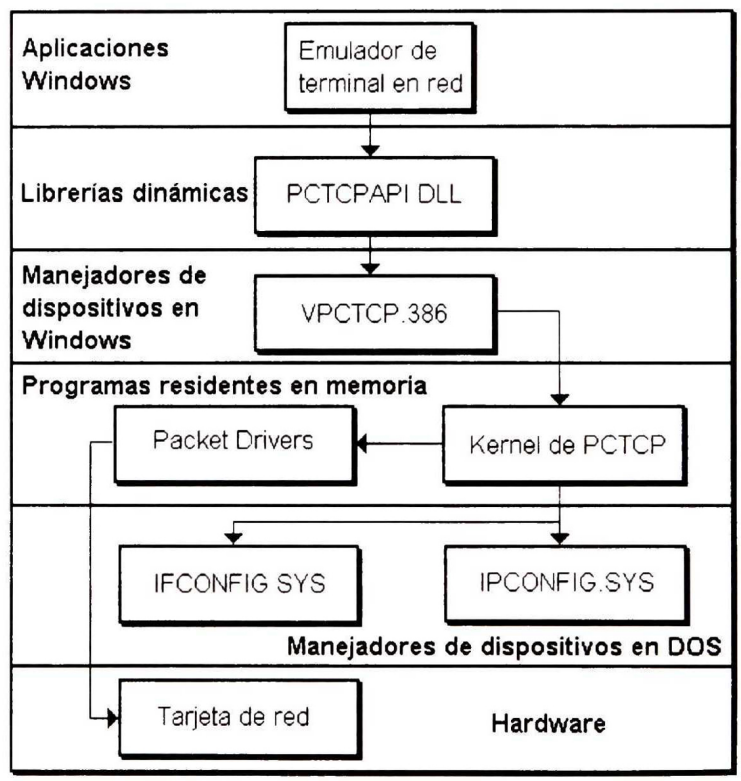
| Número de RFC | Descripción |
|---|---|
| 856 | Transmisión binaria de telnet |
| 857 | Opción de eco de telnet |
| 859 | Opción de estado de telnet |
| 861 | Opciones extendidas de telnet |
| 884 | Opción para el tipo de terminal |
| 885 | Opción de fin de registro |

## Implementación del Protocolo Telnet sobre Windows

La implementación de los servicios de comunicación sobre red, se limitó a redes que usaran el protocolo TCP/IP. Al evaluar los costos de la implementación del protocolo TCP/IP sobre Windows, se concluyó que era una actividad muy costosa en tiempo. La estrategia fue usar un producto llamado PC/TCP Development Kit de la compañía FTP Software Inc. Este producto está enfocado a DOS y ofrece un conjunto muy limitado de servicios sobre Windows, básicamente servicios de conexión y entrada/salida. La Figura 5.10 muestra la relación de todos los módulos de PC/TCP con el emulador de terminal (puede ser cualquier aplicación Windows que use los servicios de TCP/IP).

*Figura 5.10*

*Infraestructura del emulador de terminal en red*



A continuación se listan el nombre de los módulos del software de PCTCP y sus descripciones.

| Módulo | Descripción |
|---|---|
| PCTCPAPI DLL | Librería dinámica (DLL) que proporciona las funciones de entrada/salida, y la conexión |
| VPCTCP.386 | Interfaz de Windows con la implementación de TCP/IP de PCTCP |
| Packet Drivers | Interfaz con la tarjeta de red. Proporciona una interfaz genérica con la gran mayoría de las tarjetas de red |
| Kernel de PCTCP | Programa residente en memoria donde se encuentra la implementación de PCTCP del protocolo TCP/IP |
| IFCONFIG.SYS | Contiene la configuración de la tarjeta de red (número de interrupción, canal de DMA, dirección de memoria, dirección del puerto de entrada/salida) |
| IPCONFIG.SYS | Contiene la configuración del protocolo IP |

El desarrollo de este módulo fue sencillo, porque se reutilizó la gran mayoría del código del emulador de terminal para comunicación serie.

El primer paso fue sustituir las funciones de entrada/salida del API de comunicación serie de Windows por las funciones del API de PCTCP. El segundo fue reemplazar el código que maneja la conexión, es decir la inicialización del modem , el marcado y la detección del enlace. El siguiente paso fue modificar el diálogo de cambio de Parámetros. En este último se eliminó la configuración del puerto serie. Por último, se realizó una implementación básica del protocolo telnet.

La siguiente lista muestra la correspondencia entre las funciones de Windows para la comunicación serie y la funciones de PCTCP.

| API de PCTCP | API de Windows para comunicación serie |
|---|---|
| net_read | ReadComm |
| net_write | WriteComm |

A continuación se muestra las modificaciones al código y la implementación de telnet.

Se modificó la función WinMain. Ahora, además de obtener mensajes de Windows, también se monitorean las fases de conexión y se procesan mensajes relacionados con la entrada de datos de la red.

```
***** while(GetMessage(&msg, NULL, 0, 0)) { ******
* Monitorea fases de la conexión y mensajes de Windows *
do {
    * Trata de conectarse *
```

```
    if(netPhase == INIT2)
        * Fase 2 de la conexión *
        InitNet2(hClientWnd);
    if(netPhase == LOGIN) {
        NPTTYINFO  npTTYInfo;

        npTTYInfo = (NPTTYINFO) GetWindowWord(hClientWnd, GWW_NPTTYINFO);
        /* Contesta el "login" y "Password" */
        Login();
        netPhase = DATA;
        /* Exito en la conexión */
        CONNECTED(npTTYInfo) = TRUE;
        SetTTYFocus(hClientWnd);
        CheckMenuItem(GetMenu(GetParent(hClientWnd)), IDM_CONNECT,
MF_CHECKED);
    }
    /* Proceso eventos relacionados a la red */
    if(readNow)
        ProcessNetNotification(hClientWnd);
    * Libera voluntariamente el control del procesador */
    Yield();
    * Búsqueda de mensajes */
    if(!PeekMessage(&msg, NULL, NULL, NULL, PM_REMOVE | PM_NOYIELD))
        continue;
    if(!TranslateAccelerator(hTTYWnd, ghAccel, &msg)) {
        TranslateMessage(&msg) ;
        DispatchMessage(&msg) ;
    }
} while(msg.message != WM_QUIT && !bExit);
```

PCTCP no define un número de mensaje para eventos relacionados con la red, pero si define una pseudo variable, net_msgType, para distinguir este tipo de mensajes. De tal manera, que fue necesario modificar la función ClientWndProc.

```
    default: {
        NPTTYINFO npTTYInfo;
        char    prbuf[128];


        /* Si no es un mensaje relacionado con la red procesa el mensaje como antes */
        if(wMsg != net_msgType)
            goto default_proc;
        * Es un mensaje relacionado con la red */
        npTTYInfo = (NPTTYINFO) GetWindowWord(hWnd, GWW_NPTTYINFO);
        switch(netPhase) {
            default:
                goto bogus_asynch;
```

```
                case INIT2:
                    * Fase de conexión *
                    if(net_msgND(wParam) != HOSTP(npTTYInfo)- h_nd)
                        goto bogus_asynch;
                    break;
                case LOGIN:
                    * Es el momento de dar el nombre del usuario y su clave */
                    SubProcessNetNotification(hWnd);
                    break;
                case DATA:
                    /* Indica que llegaron datos de la red *
                    if(net_msgND(wParam) == ND(npTTYInfo))
                        readNow = TRUE;
                    else
                        goto bogus_asynch;
                    break;
                case DESTROYED:
                    break;
            }
            return(0L);
bogus_asynch:;
            * Muestra diálogo de error */
            sprintf(prbuf, "Mensaje desconocido %u en descriptor %u en fase %d",
                net_msgEvent(wParam), net_msgND(wParam), netPhase);
            otherPlaint(prbuf, 0);
default_proc:;
            * Procesa un mensaje de Windows en el cual no se tiene interés */
            return(DefWindowProc(hWnd, wMsg, wParam, lParam));
        }
    }
```

El tiempo necesario para establecer una conexión TCP puede ser demasiado. Se implementó la conexión en pasos. Cuando el usuario escoge la opción Conectar del menú, el mensaje es enviado a ClientWndProc. Esta última invoca a OpenConnection, que a su vez llama a Inet1, la cual realiza el primer paso. En el código de InitNet1 se indica que se tiene que realizar la fase 2 de la conexión. La fase 2 se lleva a cabo en la función WinMain por medio InitNet2. A continuación se muestra el código de las funciones OpenConnection , InitNet1 e InitNet2.

```
BOOL NEAR
OpenConnection(HWND hWnd)
{

    if(NULL == (npTTYInfo = (NPTTYINFO) GetWindowWord(hWnd, GWW_NPTTYINFO)))
        return(FALSE);
    * Fase 1 de la conexión */
    if(InitNet1(hWnd)   0) {
```

```
            * Error, termina la aplicación */
            PostMessage(hTTYWnd, WM_CLOSE, NULL, 0L);
            return(FALSE);
        }
        return(TRUE);
}
int
InitNet1(HWND hWnd)
{
        int     i;
        int     errsave;
        char    *errP;
        NPTTYINFO  npTTYInfo;

        if(NULL == (npTTYInfo = (NPTTYINFO) GetWindowWord(hClientWnd,
         GWW_NPTTYINFO)))
            return(1000);
        /* Inicializa PC/TCP. */
        if (net_taskInit("winvt") == -1)
            return(perrorTypePlaint("Error: Inicializacion PC/TCP DLL.", 0));
        * Obtiene la versión de PC/TCP */
        NETVSN(npTTYInfo) = (unsigned) get_netversion();
        /* Crea un descriptor */
        if ((i = net_getdesc()) < 0) {
            errP = "net_getdesc HOST";
            goto plaintDeregNQuit;
        }
        /* Registra el evento NET_AS_RCV */
        if (net_asynchw(i, NET_AS_RCV, hClientWnd, NET_ASWM_POST) == -1L) {
            errP = "Registro de NET_AS_RECV";
            goto plaintDeregNQuit;
        }
        /* Inicia la traducción del nombre del host a una direccion IP */
        if(!(HOSTP(npTTYInfo) = host_nm_query("hp877", i))) {
            errP = "Consulta del hosts";
            goto plaintDeregNQuit;
        }
#ifdef COMMENT
        /*************** Comentario *************/
        /* Lanza un temporizador de un segundo */
        if(SetTimer(hOurWnd, 0, 1000, 0) == NULL) {
            otherPlaint("SetTimer failed", 0);
            goto deregNQuit;
        }
        *************** Comentario *************/
#endif
        /* Indica que se tiene que realizar la fase 2 de la conexión posteriormente */
        netPhase = INIT2;
```

```
        return 0;
        * Adiós *
plaintDeregNQuit:;
        perror TypePlaint(errP, 0);
deregNQuit:;
        net_taskDestroy();
        return(-1);
}
```

Se modificó la función que realiza la desconexión.

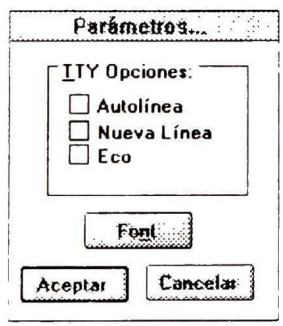
```
BOOL NEAR
CloseConnection(HWND hWnd)
{
    HMENU     hMenu;
    NPTTYINFO  npTTYInfo;

    if(NULL == (npTTYInfo = (NPTTYINFO) GetWindowWord(hWnd, GWW_NPTTYINFO)))
        return(FALSE);
    KillTTYFocus(hWnd);
    CONNECTED(npTTYInfo) = FALSE;
    hMenu = GetMenu(GetParent(hWnd));
    CheckMenuItem(hMenu, IDM_CONNECT, MF_UNCHECKED);
    * Cierra la conexión TCP *
    net_taskDestroy();
    ND(npTTYInfo) = -1;
    return(TRUE);
}
```

Se modificó el diálogo de configuración de Parámetros y el código que maneja su comportamiento. Se eliminaron todos los elementos que configuraban el puerto serie. En la Figura 5.11 se muestra el diálogo resultante.

*Figura 5.11*

*Diálogo para la configuración de terminal*




Por último, se realizó una implementación básica del protocolo telnet. La función TelnetCommand contiene el código del protocolo. Fue necesario

modificar la función que procesa los datos de entrada (WriteTTYBlock). A continuación se muestra el código de TelnetCommand y la modificación de WriteTTYBlock.

```
BOOL NEAR PASCAL
TelnetCommand(HWND hWnd, LPSTR lpBlock)
{
    NPTTYINFO npTTYInfo;

    if(NULL == (npTTYInfo = (NPTTYINFO) GetWindowWord(hWnd, GWW_NPTTYINFO)))
        return(FALSE);
    * Estados del protocolo */
    switch(TELSTATE(npTTYInfo)) {
        case TS_DATA:
            /* Posiblemente se inicie una secuencia de escape */
            if((BYTE) lpBlock[0] == IAC) {
                TELSTATE(npTTYInfo) = TS_IAC;
                break;
            }
            return(FALSE);
        case TS_IAC:
            switch((BYTE)lpBlock[0]) {
                /* Opciones de negociación */
                case WILL:
                    TELSTATE(npTTYInfo) = TS_WILL;
                    break;
                case WONT:
                    TELSTATE(npTTYInfo) = TS_WONT;
                    break;
                case DO:
                    TELSTATE(npTTYInfo) = TS_DO;
                    break;
                case DONT:
                    TELSTATE(npTTYInfo) = TS_DONT;
                    break;
                * Dos IAC significan un IAC como dato */
                case IAC:
                    TELSTATE(npTTYInfo) = TS_DATA;
                    ExtWriteTTYChar(hWnd, lpBlock);
                    break;
                default:
                    TELSTATE(npTTYInfo) = TS_DATA;
                    break;
            }
            break;
        * Procesa opciones de negociación con el servidor telnet */
        case TS_WILL:
            WillOpt(hWnd, lpBlock[0]);
```

```
                TELSTATE(npTTYInfo) = TS_DATA;
                break;
            case TS_WONT:
                WontOpt(hWnd, lpBlock[0]);
                TELSTATE(npTTYInfo) = TS_DATA;
                break;
            case TS_DO:
                DoOpt(hWnd, lpBlock[0]);
                TELSTATE(npTTYInfo) = TS_DATA;
                break;
            case TS_DONT:
                DontOpt(hWnd, lpBlock[0]);
                TELSTATE(npTTYInfo) = TS_DATA;
                break;
        }
        return(TRUE);
}

BOOL NEAR
WriteTTYBlock(HWND hWnd, LPSTR lpBlock, int nLength)
{
    int     i;
    NPTTYINFO  npTTYInfo;

    if(NULL == (npTTYInfo = (NPTTYINFO) GetWindowWord(hWnd, GWW_NPTTYINFO)))
        return(FALSE);
    for(i = 0; i    nLength; i++) {
        ********** Modificación para telnet *********/
        * Si es una secuencia de escape de telnet procesa el siguiente
          carácter
        */
        if(TelnetCommand(hWnd, lpBlock + i)) continue;
        ********** Modificación para telnet *********/
        // Ve si el carácter es un ESC
        if(lpBlock[i] == ASCII_ESC) {



}
```

Con el esquema presentado se puede decir que los cambios fueron mínimos.

Parte
3

# Conclusiones, Bibliografía y Apéndices

Capítulo

# 6 Conclusiones

En este capítulo se discuten dos puntos importantes: las limitaciones de las herramientas y las diferentes alternativas para el desarrollo de aplicaciones Windows.

***Contenido de este capítulo***



## Limitaciones

Las herramientas desarrolladas tienen algunas limitaciones:

- Interfaz gráfica. Existe una inconsistencia en el uso de los menús y ventanas de la aplicación remota y los del emulador de terminal. La aplicación remota no responde al ratón.
- Protocolo de comunicación. El protocolo para la transferencia de archivos es ineficiente.
- Comunicación con otras aplicaciones. No se cuenta con una interfaz con otras aplicaciones como Excel, Word, etc.
- La versión para red no cuenta con un protocolo para transferencia de archivos.
- El emulador de terminal no es automático ni programable. El emulador requiere la interacción con el usuario. Si un usuario desea servicios periódicos, tiene que realizar la misma secuencia de pasos en cada sesión. Por ejemplo el usuario quiere tener el archivo FACTURA, el cual se actualiza diario en un equipo remoto. El usuario tiene que realizar la conexión y la transferencia todos los días.
- Las herramientas no tienen una interfaz con el modem.

A continuación se discute la complejidad de las soluciones para eliminar las limitaciones.

El primer punto es el más difícil de solucionar. La solución es hacer una aplicación cliente/servidor. El cliente (programa que se encuentra en la PC) se encargará del despliegue de los datos. El servidor procesará los datos.

El protocolo Kermit es ineficiente cuando procesa archivos binarios, el tamaño de los paquetes es muy pequeño, se pierden atributos de los archivos transmitidos tales como: fecha de creación, permisos.

La implementación de un protocolo más eficiente como Zmodem es complicado. Es más fácil la implementación del protocolo SuperKermit. Este último corrige todas las desventajas del protocolo básico.

La comunicación con otras aplicaciones se refiere a los siguientes puntos:

1. Otras aplicaciones puedan llamar al emulador de terminal para accesar un servicio remoto
2. El emulador de terminal pueda transferir datos a otras aplicaciones
3. El emulador de terminal accese a otras aplicaciones
4. El formato del archivo que transfiere la computadora remota sea el especificado por una aplicación Windows.

Es necesario utilizar el clipboard para copiar datos que otras aplicaciones puedan utilizar e implementar la comunicación con otras aplicaciones a través del protocolo de Windows DDE (Intercambio Dinámico de Datos).

Se necesita implementar el protocolo FTP para la transferencia de archivos en red. Esta tarea no se realizó porque no se consiguieron los RFCs de FTP.

Es necesario definir un lenguaje que permita programar al emulador de terminal. Se necesita además un calendarizador que ejecute los programas del emulador de terminal a una hora específica.

El último punto es muy sencillo de realizar, sin embargo existe el inconveniente de que se encuentran en el mercado una gran variedad de tipos de modems, razón por la cual es necesario crear un archivo en donde se almacenen las características de los tipos de modems más usados.

## Herramientas para el Desarrollo de Aplicaciones Windows

El desarrollo de aplicaciones Windows con el lenguaje C es complicado y arduo. Existen otras alternativas similares como Pascal y Visual Basic.

La popularidad de C++ ha llegado a Windows con la introducción de compiladores como Borland C++ y Microsoft C++. Estas compañías han desarrollado clases que encapsulan las funciones de Windows y simplifican el desarrollo.

Existen además otros productos que superan la funcionalidad de las clases de Borland y Microsoft Uno de ellos es Zinc Interface Library. Zinc ofrece un mismo conjunto de clases para los ambientes DOS, Windows, OS/2 PM y Motif.

Smalltalk for Windows y Actor son dos lenguajes orientados a objetos que contienen bastantes clases que reducen la curva de aprendizaje de la programación en Windows.

Existen también generadores de aplicaciones como Object Vision y Windows Maker.

En resumen, existen una amplia variedad de productos y alternativas para el desarrollo de aplicaciones Windows.

# Bibliografía

En este capítulo se proporcionará algunas sugerencias de lecturas posteriores, así como una bibliografía, para beneficio de las personas interesadas sobre la programación en Windows, protocolos para transferencia de archivos, redes de computadoras, TCP/IP y programación de aplicaciones de redes de computadoras.

***Contenido de este capítulo***



## Sugerencias de otras lecturas

Existen varias revistas que publican artículos acerca de la programación en Windows. Las más destacadas son:

- Dr Dobb's Journal.
- Microsoft System Journal.
- PC Magazine.

Estas revistas contienen no sólo artículos referentes a la programación, sino también a filosofía de Windows, casos de estudio, sugerencias, experiencias y puntos de vista de usuarios, nuevos productos para el desarrollo, revisiones y evaluaciones de aplicaciones.

A continuación se proporciona una lista de sugerencias de lecturas adicionales.

## Programación en Windows

Petzold, Programming Windows 3.1.
Es el libro básico para aprender a programar en Windows. Cubre todos los aspectos relacionados con este ambiente.

Norton, Yao, Windows 3.0 Power Programming Techniques.
Este libro contiene explicaciones detalladas acerca de los aspectos más básicos de la programación en Windows. Es altamente recomendado para programadores de nivel elemental o intermedio.

## Protocolos para Transferencia de Archivos.

Campbell, C Programmer's Guide to Serial Communications.
Este libro contiene la información para programar el puerto serie y modems Hayes. En los últimos capítulos trata de la implementación del protocolo XModem.

Anderson, Kermit Meets Modula-2.
Este artículo trata de la implementación del protocolo Kermit en el lenguaje Modula 2. Contiene una excelente guía para aprender e implementar Kermit en otras plataformas o en otros lenguajes.

## Redes de Computadoras

Tanenbaum, Redes de Ordenadores.
Contiene una amplia introducción a las redes de computadoras. Es muy completo, trata de los protocolos y algoritmos desde la capa física hasta la capa de aplicación y desde las redes locales hasta las redes de satélite.

Kochan, Wood, Unix Networking.
Este libro presenta una recopilación de artículos, escritos por expertos, de los principales sistemas de redes en Unix. Cubre aspectos referentes a la programación, detalles internos y las ventajas y características de varias redes, incluyendo:

- UUCP: la primera red, diseñada para funcionar sobre líneas telefónicas.
- TCP/IP.
- NFS: El sistema de archivos de red de Sun Microsystems.
- RFS: El sistema de archivos de red de AT&T.
- Streams: Una interfaz modular y flexible para el desarrollo de redes.
- LAN Manager/X: Un protocolo de red para OS/2, DOS y Unix.
- X Windows y News: Dos sistemas gráficos de red para Unix.

## TCP/IP

Comer, Internetworking with TCP/IP Vol I.
Es en mi punto de vista, el libro más completo de TCP/IP. Contiene la historia, explicaciones del funcionamiento de la familia de protocolos TCP/IP, del

modelo Cliente-Servidor, de la interfaz con el sistema operativo (programación con sockets), de las aplicaciones telnet, ftp, y correo electrónico.

## Programación de Aplicaciones para Redes

Stevens, Unix Network Programming.
No obstante el título, este libro es aplicable para varios ambientes. Contiene la historia de las redes de computadoras, el modelo Unix, las varias maneras de comunicación entre procesos, una breve discusión de los protocolos de redes, programación con sockets e implementación de aplicaciones como: transferencia de archivos con el protocolo TFTP, ejecución de comandos remotos (rsh).

Comer, Internetworking With TCP/IP Vol III.
Este libro está orientado a aplicaciones Cliente-Servidor con TCP/IP. Contiene una explicación profunda del modelo Cliente-Servidor, involucrando tópicos como: diseño de software, interfaces (sockets), algoritmos para el cliente y el servidor y ejemplos (la implementación del protocolo telnet para el cliente y para el servidor). Es el único libro que conozco que contiene la explicación a profundidad de los siguientes temas avanzados: XDR (representación de datos esternos), RPC (llamadas a procedimientos remotos), NFS (sistemas de archivos en red).

## Bibliografía en Orden Alfabético

ANDERSON BRIAN R.: "Kermit Meets Modula-2", Dr Dobb's Journal, No 151, p.p. 22-26, Mayo 1989.

BIGGERSTAFF TED J.: "System Software Tools", Englewood, New Jersey: Prentice Hall, 1989.

CALBAUM MIKE, PORCARO FRANK, RUEGSEGGER MARK, BACKMAN BRUCE: "Untaggling the Windows Sockets API", Dr Dobb's Journal, No. 197, p.p. 66-71, Febrero 1993.

CAMPBELL JOE, "C Programmer's Guide to Serial Communications", Indianapolis, Indiana: Howard W. Sams & Company, 1987.

COMER DOUGLAS E.: "Internetworking with TCP/IP", Vol. I, Englewood, New Jersey: Prentice Hall, 1991.

COMER DOUGLAS E., STEVENS DAVID L.: "Internetworking with TCP/IP", Vol. II, Englewood, New Jersey: Prentice Hall, 1991.

COMER DOUGLAS E., STEVENS DAVID L.: "Internetworking with TCP/IP", Vol. III, Englewood, New Jersey: Prentice Hall, 1993.

FTP: PC/TCP Development Kit Library Reference, Wakefield, MA, FTP Software Inc, 1991.

FTP: PC/TCP Interoperability, Wakefield, MA, FTP Software Inc, 1991.

FTP: PC/TCP User's Guide, Wakefield, MA, FTP Software Inc, 1991.

HALL WILLIAN S.: "A simple Terminal Program for Windows", Programmer's Journal, Vol. 7.6, p.p. 26-38, Noviembre 1989.

HALL WILLIAN S.: "Adapting Extended Process to the cooperative Multitasking of Microsoft Windows", Microsoft System Journal, Vol. 6 No. 1, p.p. 21-34, Enero de 1991.

HELLER MARTIN: "Advanced Windows Programming", New York: John Wiley & Sons Inc, 1992.

HEWLETT PACKARD: "Remote Access: User's Guide", USA, Hewlett-Packard, 1991.

KLEIN MIKE: "Windows Programmer's Guide to DLLs and Memory Mangement", Carmel Indiana: SAMS, 1992.

KOCHAN STEPHEN G., WOOD PATRICK: "Unix Networking", Carmel, Indiana: Hayden Books, 1990.

LEAVENS ALEX: "Building Free-Standing Control Buttons", Windows/Dos, Vol. 3 No. 2, p.p. 53-62, Febrero 1992.

MEFFORD MICHAEL J.: "ANSI.SYS Without the hassle", PC Magazine, Vol. 8 No. 2, p.p. 229-254, Enero 1989.

MICROSOFT: "Microsoft Windows Software Development Kit version 3.1 Volume 1: Overview", USA, Microsoft Corporation, 1992.

MICROSOFT: "Microsoft Windows Software Development Kit version 3.1 Volume 2: Functions", USA, Microsoft Corporation, 1992.

MICROSOFT: "Microsoft Windows Software Development Kit version 3.1 Volume 3: Messages, Structures and Macros", USA, Microsoft Corporation, 1992.

MICROSOFT: "Microsoft Windows Software Development Kit version 3.1 Volume 4: Resources ", USA, Microsoft Corporation, 1992.

MICROSOFT: "Microsoft Windows version 3.1: Programming Tools", USA, Microsoft Corporation, 1992.

MICROSOFT: "Microsoft Windows version 3.1: User's Guide", Redmond, Washington, Microsoft Press, 1992.

MOTOROLA: "3260/3265 Series Modem Reference and Applications Guide", Mansfield, Massachusetts, Motorola Codex Corporation, 1992.

NORTON PETER, YAO PAUL: "Windows 3.0 Power Programming Techniques", New York: Bantam books, 1990.

PATCH RAY, SINHA ALOK: "Developing Netware-aware Programs Using Windows 3.1 and NetBIOS", Microsoft System Journal, Vol 7 No. 4, p.p. 57-75, Julio 1992.

PETZOLD CHARLES: "Programming Windows 3.1", Redmond, Washington: Microsoft Press, 1992.

RICHTER JEFFREY M.: "Windows 3: A Developer's Guide", Redwood City, CA: M&T Books, 1991.

RIEKEN BILL, WEIMAN LYLE: "Adventures in Unix Network Applications Programming", New York: John Wiley & Sons Inc, 1992.

SAX MIKE: "The Windows Communications API", Dr Dobb's Journal, p.p. 40-44, Mayo 1992.

SINHA ALOK, PATCH RAYMOND: "An Introduction to Network Programming Using the NetBIOS Interface", Microsoft System Journal, Vol. 7 No. 4, p.p. 61-81, Marzo 1992.

SMITH DONALD W.: "Finite State Machines for XModem", Dr Dobb's Journal, No. 156, p.p. 45-50, Octubre 1989.

STEVENS W. RICHARD: "Unix Network Programming", Englewood Cliffs, New Jersey: Prentice Hall, 1990.

TANENBAUM ANDREW S.: "Redes de Ordenadores", México: Prentice Hall Hispanoamericana, 1991.

WILTON RICHARD: "Windows 3.0: Developer's Workshop", Redmond Washington: Microsoft Press, 1991.

*Apéndice*

# A El Programa Wart

El programa Wart implementa un pequeño subconjunto del analizador léxico de Unix llamado lex. Wart puede ser distribuido sin el requerimiento de una licencia de Unix. Wart fue escrito por Jeff Damens en la Universidad de Columbia en el Centro de Actividades de Computo. El propósito de Wart es facilitar el desarrollo del protocolo Kermit para Unix.

Wart produce tablas de estados. Permite definir un conjunto de estados, una función para obtener la entrada de datos y una tabla de transiciones de estados. Wart genera un programa en lenguaje C que realiza ciertas acciones y cambios de estado basado en el estado actual y el dato de entrada.

El programa wart acepta un programa en C que inicia con "%" o con secciones delimitadas por "%%". La directiva "%states" declara los estados del programa. La sección que se encuentra entre los delimitadores "%%", es la tabla de estados. Los elementos de la tabla de estados tienen el siguiente formato:

<state>X { action }

el cual es leído como "*Si el estado actual es state y recibe el dato X, entonces realiza la acción { action }*"

El campo opcional <state> indica el estado o estados en el cual debe encontrarse el programa para realizar la acción indicada. Si el estado no es especificado, significa que la acción debe ser realizada sin importar el estado actual. Si más de un estado es especificado, la acción será ejecutada en cualquiera de los estados. Los estados en la lista son separados por comas.

El dato de entrada consiste de un carácter. Cuando el programa se encuentra en el estado indicado, si el dato es el carácter especificado, la acción asociada es realizada. El carácter '.' es usado para representar cualquier carácter. Wart no proporciona expresiones regulares. El carácter de entrada es obtenido de la función input, la cual es necesario definir. Este carácter puede ser un carácter alfanumérico o alguno de los siguientes:

% - $ @

La acción es una secuencia delimitada por llaves, de estatutos del lenguaje C.

La macro BEGIN es definida para simplificar la siguiente asignación:

state =

La función wart es generada por el programa Wart, basándose en la declaración de estados y en la tabla de transición. wart contiene un ciclo, donde invoca a la función input, y el resultado que esta última le devuelve es usado como índice para el estatuto switch.

La manera de invocar a Wart es:

| | |
|---|---|
| wart | (La entrada es stdin, la salida es stdout) |
| wart fn1 | (La entrada es el archivo fn1, la salida es stdout) |
| wart fn1 fn2 | (La entrada es fn1, la salida es fn2. Ejemplo wart a.w a.c). |

La extensión sugerida para los programas wart es '.w'.

El programa siguiente demuestra algunas de las capacidades y limitaciones de Wart. El programa acepta de la línea de comandos un número binario, precedido opcionalmente por un signo menos y posiblemente seguido de una parte fraccional. El programa imprime el número decimal correspondiente.

## *BINTODEC.W*

```
#include  stdio.h

int state, s = 1, m = 0, d;
float f;
char *b;

/* Declara los estados */
%states sign mantissa fraction

%%                                          /* Inicia la tabla de estados */
  sign     { s = -1; BEGIN mantissa; }      /* Signo */
  sign  0  { m = 0;  BEGIN mantissa; }      /* Es un dígito; Inicia mantisa */
  sign  1  { m = 1;  BEGIN mantissa; }
  sign     { fatal("Entrada errónea"); }    /* Detecta formato erróneo */
  mantissa 0  { m *= 2; }                   /* Acumula mantisa */
  mantissa  1  { m = 2 * m + 1; }
  mantissa  $  { printf("%d\n", s * m); return; }
  mantissa  .  { f = 0.0; d = 1; BEGIN fraction; }  /* Inicia fracción*/
  fraction  0  { d *= 2; }                  /* Acumula fracción */
  fraction  1  { d *= 2; f += 1.0 / d; }
  fraction  $  { printf("%f\n", s * (m + f) ); return; }
  fraction     { fatal("Entrada errónea"); }
%%
```

```
input() {                                   * Define la función de entrada */
   int x;
   return(((x = *b++) == '\0') ? '$' : x );
}

fatal(s) char *s; {                                    /* Error */
   fprintf(stderr, "Error - %s\n",s);
   exit(1);
}

main(argc,argv) int argc; char **argv; {
   if (argc-- < 1) exit(1);
   b = *++argv;
   state = sign;                  * Inicializa el estado */
   wart();
   exit(0);
}
```

## BINTODEC.C

```
 * WARNING -- This C source program generated by Wart preprocessor. */
 * Do not edit this file; edit the Wart-format source file instead, */
 * and then run it through Wart to produce a new C source file.    */

 * Wart Version Info: */
char *wartv = "Wart Version 1A(006) Jan 1989";

#include <stdio.h>

int state, s = 1, m = 0, d;
float f;
char *b;

 * Declara los estados */
#define sign 1
#define mantissa 2
#define fraction 3


#define BEGIN state =

int state = 0;

wart()
{
```

```
  int c,actno;
  extern short tbl[];
  while (1) {
     c = mput();
     if ((actno = tbl[c + state*128]) != -1)
        switch(actno) {
case 1:
  { s = -1; BEGIN mantissa; }
  break;
case 2:
  { m = 0; BEGIN mantissa; }
  break;
case 3:
  { m = 1; BEGIN mantissa; }
  break;
case 4:
  { fatal("Entrada errónea"); }
  break;
case 5:
  { m *= 2; }
  break;
case 6:
  { m = 2 * m + 1; }
  break;
case 7:
  { printf("%d\n", s * m); return; }
  break;
case 8
  { f = 0.0; d = 1; BEGIN fraction; }
  break;
case 9:
  { d *= 2; }
  break;
case 10:
  { d *= 2; f += 1.0 / d; }
  break;
case 11:
  { printf("%f\n", s * (m + f) ); return; }
  break;
case 12:
  { fatal("Entrada errónea"); }
  break;

        }
  }
}

short tbl[] = {
```

```
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,
 4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,
 4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  1,  4,  4,
 2,  3,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,
 4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,
 4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,
 4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,
 4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,  4,
-1,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,
 8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,
 8,  8,  8,  8,  7,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,
 5,  6,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,
 8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,
 8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,
 8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,
 8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,  8,
 0, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12,
12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12,
12, 12, 12, 12, 11, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12,
 9, 10, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12,
12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12,
12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12,
12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12,
12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12,
};


input() {                    /* Define la función de entrada */
  int x;
  return(((x = *b++) == '\0') ? '$' : x );
}

fatal(s) char *s; {          /* Error */
  fprintf(stderr, "Error - %s\n",s);
  exit(1);
}

main(argc,argv) int argc; char **argv; {
  if (argc    1) exit(1);
  b = *++argv;
```

```
    state = sign;                    * Inicializa el estado *
    wart();
    exit(0);
}
```

Apéndice

# B Ejemplo de una Aplicación

Este apéndice describe brevemente una aplicación que permite accesar desde una computadora personal información que se encuentra en una base de datos de una minicomputadora.

## Sistema de Información Corporativo (VTex)

El sistema VTex proporciona un acceso a las bases de datos corporativas de TELMEX, un correo electrónico y una posible gama de servicios adicionales; todo a través de la red telefónica y de la red universal de TELMEX.

VTex incluye un programa de comunicaciones que opera sobre Windows en el sitio del usuario (IBM-PC o compatibles) y el Nodo de Acceso VTex (HP-9000/877) con capacidad para atender a varios usuarios a la vez.

Este sistema tiene como características:

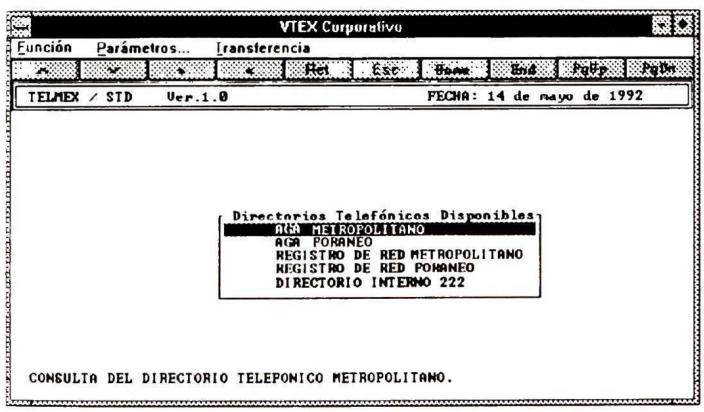
- El poder integrar cualquier base de datos sin modificar el código.
- El poder configurar la información a mostrar.

Estas dos facilidades permiten usar un mismo sistema para diferentes usuarios.

La figura B.1 muestra el menú de directorios (bases de datos) disponibles.

*Figura B.1*

*Menú de directorios disponible*

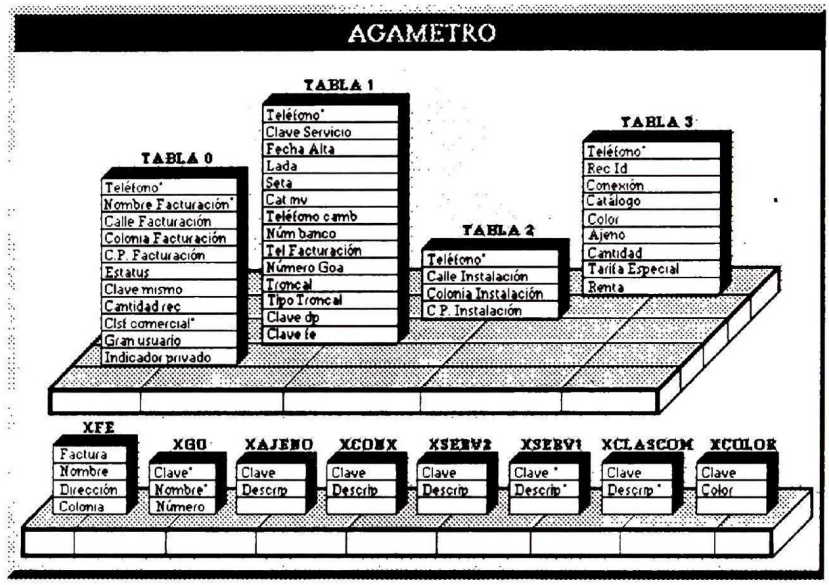


Para explicar el uso del sistema se tomará como ejemplo la primer opción del menú de directorios: El AGA METROPOLITANO.

El AGA METROPOLITANO es el Archivo General de Abonados. Este archivo contiene los datos esenciales para efectuar la facturación y algunos procesos estadísticos tales como: ocupación de la red, cantidad de líneas de aparatos por tipo de servicios, entre otros. Este archivo es en realidad una base de datos relacional con varias tablas como se aprecia en la figura B.2.

*Figura B.2*

*Estructura del Registro General de Abonados*




El AGA METROPOLITANO contiene cerca de 4.5 millones de abonados los cuales ocupan un poco más de 2.5 GigaBytes.

Dicho archivo fue creado para el control de los abonados y para abastecer de datos a varios procesos y subprocesos que se manejan en las diferentes áreas de TELMEX. Los datos que contiene son: número telefónico, nombre del abonado, domicilio, tipo de aparato, clase de servicio, entre otros campos que describen campos administrativos y técnicos del abonado.

VTex es un sistema de consultas. Generalmente las consultas se basan en las llaves de las tablas de la base de datos. Se pueden combinar información de varias tablas y de varias bases de datos. La figura B.3 muestra la pantalla de consulta de información y la figura B.4 muestra el resultado de la búsqueda.

*Figura B 3*

*Pantalla para consulta de registros del AGA METROPOLITANO*

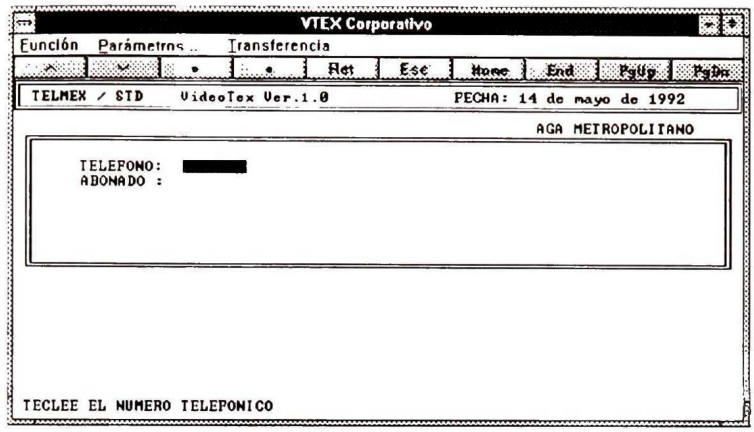
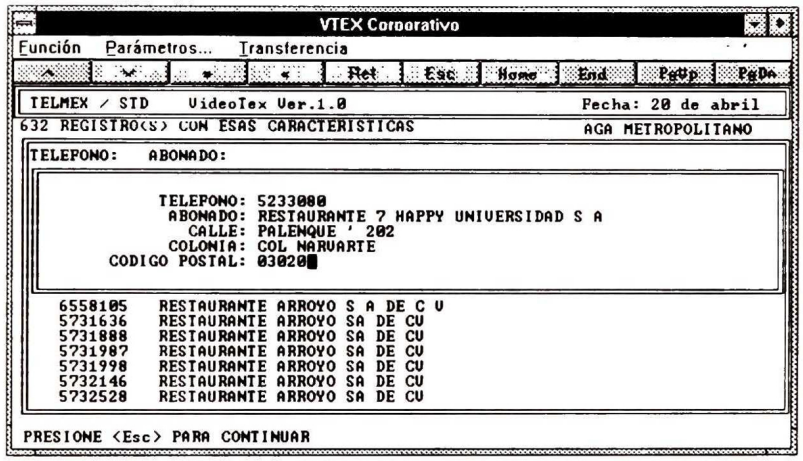



*Figura B.4*

*Pantalla que muestra el resultado de la búsqueda*




VTex tiene la opción de transferir toda la información que se ha consultado a un archivo local de la PC llamado BITACORA.TXT. Para realizar esta opción se utilizó la transferencia de archivos del emulador de terminal y el programa kermit de la HP 9000.

El archivo BITACORA.TXT se puede leer con el programa NOTEPAD.EXE de Windows.

Las figuras B.5 y B.6 muestran la transferencia de archivos y el uso de la información por otra aplicación Windows.

*Figura B.5*

*Transferencia del archivo BITACORA.TXT entre la HP 9000 y la PC*

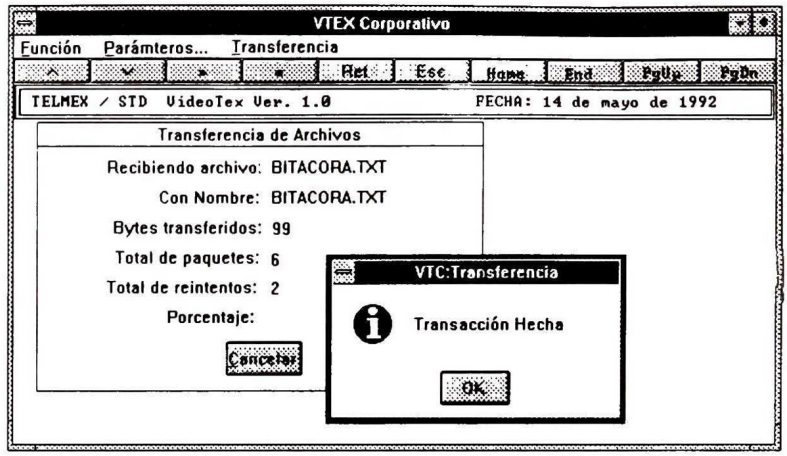



*Figura B.6*

*El archivo BITACORA.TXT es acceso por la aplicación NOTEPAD*

CENTRO DE INVESTIGACION Y DE ESTUDIOS AVANZADOS DEL I.P.N.

**EL JURADO DESIGNADO POR LA SECCION DE COMPUTACION DEL DEPARTAMENTO DE INGENIERIA ELECTRICA, APROBO EL DIA** 14 **DEL MES DE** JUNIO **DEL AÑO DE** 1993 **, EL TRABAJO DE TESIS.**

Herramientas de comunicaciones para Microsoft Windows

**DESARROLLADO POR EL ALUMNO:**

David Austreberto Solís Pacheco

Dr. Sergio V. Chapa Vergara
Jefe de la Sección

M en C. Carlota M. Tamayo I.
Coordinadora Académica

Dr. Manuel E. Guzmán Rentería
Investigador Titular del Centro de Tecnología de Semiconductores de la Unidad Guadalajara

Ave. Instituto Politécnico Nacional 2508 Col. San Pedro Zacatenco México, D.F. C.P. 07300
Tels: 752-0677 y 754-0200 Telex: 1772826 PPTME Fax: 752-0590 y 586-6290