Universidad Autónoma de Guadalajara

INCORPORADA A LA UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

ESCUELA DE INGENIERIA

"IMPLEMENTACION DE UNA RED DE ANILLO DE INSERCION PARA MICROCOMPUTADORAS AT&T UNIX PC'S."

TESIS PROFESIONAL

QUE PARA OBTENER EL TITULO DE

INGENIERO EN COMPUTACION

P R E S E N T A

SERGIO PANIAGUA WEBB

GUADALAJARA, JALISCO, 1990.

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

CONTENIDO.

INTRODUCCION. 1

ANTECEDENTES. 2

CAPITULO 1: ESTRUCTURA GENERAL DE LA RED. 3

1.1 CARACTERISTICAS GENERALES. 3
1.2 RELACION DEL PROGRAMA CON EL MODELO OSI. 4
1.3 CONSIDERACIONES GENERALES. 4
1.3.1 Detección de errores. 4
1.3.2 Transparencia de la información. 5
1.3.3 Control de la secuencia. 5
1.3.4 Control de la línea y de "timeouts". 6
1.3.5 Paquetización de la información. 7
1.3.6 Definición del protocolo. 7
1.3.7 Excepciones y casos especiales. 8
1.3.8 Tipos de mensajes o interface de usuario. 8
1.3.9 Topología y conexión física. 9
1.4 AMBIENTE DE DESARROLLO Y TECNICAS UTILIZADAS. 10
1.5 DESCRIPCION GENERAL DE LOS PROCESOS DEL PROGRAMA. 10
1.6 DESCRIPCION GENERAL DE LOS ALGORITMOS. 11
1.7 PRINCIPALES ESTRUCTURAS DE DATOS. 12

CAPITULO 2: DEFINICION DEL PROTOCOLO. 14

2.1 ELEMENTOS DE LOS PAQUETES. 14
2.2 DESCRIPCION DETALLADA DE LOS CAMPOS DE LOS PAQUETES. 14
2.2.1 Encabezado. 14
2.2.2 Cuerpo del paquete. 17
2.2.3 Terminador. 18
2.3 DEFINICION FORMAL DEL PROTOCOLO. 21

CAPITULO 3: DEFINICION DE LA MAQUINA DE ESTADO FINITO. 22

3.1 DESCRIPCION DETALLADA DE LOS ALGORITMOS DEL PROGRAMA. 22
3.1.1 Programa principal. 22
3.1.2 Proceso de entrada. 24
3.1.3 Proceso transmisor. 27
3.1.4 Proceso receptor. 28

CAPITULO 4: IMPLEMENTACION DE PROCESOS. 31

4.1 DESCRIPCION DE LAS ESTRUCTURAS DE DATOS. 31
4.1.1 Tabla de ACKs. 31
4.1.2 Tabla de recepción. 32
4.1.3 Buffers de entrada y salida. 33
4.1.4 Método de intercomunicación inter-procesos (PIPE). 33
4.2 ENTRADA/SALIDA. 33
4.3 CONSIDERACIONES ADICIONALES. 34
4.4 INSTRUCCIONES ESPECIALES PARA UNIX PCs. 36

CAPITULO 5: INTEGRACION DEL PROGRAMA, CONEXION FISICA E INTERFACE DE USUARIO. 38

5.1 INTEGRACION DEL PROGRAMA. 38
5.2 CONEXION FISICA. 39
5.3 INTERFACE DE USUARIO. 40
5.3.1 Ventanas. 40
5.3.2 Comandos válidos. 41
5.3.3 Mensajes del programa de red. 42
5.4 CODIGO DEL PROGRAMA DE RED. 42

CONCLUSIONES Y RECOMENDACIONES. 43

APENDICE A:
LISTADO DE LOS ARCHIVOS DEL PROGRAMA CONTROLADOR DE RED. 46

ARCHIVO DEL RECEPTOR Y TRANSMISOR (RecTran.C) 47

ARCHIVO DEL PROCESO DE ENTRADA (Entrada.C) 64

ARCHIVO DE DEFINICIONES (DefRed.H) 81

ARCHIVO PARA LA UTILERIA MAKE (Makefile) 83

BIBLIOGRAFIA. 84

INTRODUCCION.

Mediante el desarrollo de el programa controlador de red, se busca obtener un sistema para la intercomunicación de computadoras personales, el cual sea económico, sencillo de usar y que sea una base para construir aplicaciones mas avanzadas.

Para lograr que sea económico, la red será manejada totalmente por "software" utilizando la electrónica para comunicaciones ya incluida en el computador. Haciendo de esta manera innecesaria la construcción o adquisición de equipo electrónico ajeno al mismo. Así, solo se necesitarán el puerto asincrono serial de los computadores, y cable telefónico que sirva como medio de comunicación.

Con el fin de que el programa sea sencillo de usar, se tendrá un interface de usuario en base a ventanas. Las cuales servirán para saber cual es el estado en que se encuentra el computador, con respecto a comunicaciones.

Lo anterior permitirá la comunicación entre diferentes elementos de la red, para la transmisión de datos libre de errores y confiable. En los cuales la velocidad no es un factor esencial, sino que lo principal es lograr la comunicación entre varias maquinas. Además de ser una herramienta útil para fines didácticos y de experimentación dentro del área de las comunicaciones de datos.

ANTECEDENTES.

A medida de que el número de computadores personales crece en una instalación, la necesidad de poder comunicarlos entre si va también en aumento. Estas necesidades de intercomunicación van desde una simple transferencia de mensajes hasta el correo electrónico. Desde la simple transferencia de archivos hasta el desarrollo de sistemas distribuidos; pasando por la necesidad de compartir periféricos tales como discos de alta capacidad o bien impresoras, plotters o scanners entre otros.

Así como lo dice TANENBAUM : "Al principio de la computación las computadoras se consideraban unidades aisladas. Sin embargo, a medida de que las necesidades de compartir información aumentaron, la necesidad de comunicarse entre ellas se hizo mas evidente. Por ello hoy muchas computadoras estan interconectadas formando redes y sistemas distribuidos"(1).

Con el propósito de resolver estas necesidades se han ideado diversas formas de interconexión, en la actualidad se ha hecho gran énfasis en las redes locales o LANs. Ya que permiten la comunicación de muchos computadores dentro de una área pequeña a altas velocidades.

Como lo establece LEFKON : "En la decada pasada, las redes de area local (LANs) han crecido en popularidad. Esto se debe a que dan la facilidad a microcomputadoras de comunicarse entre si, y con impresoras y servidores de archivos, sin la necesidad de un "mainframe" de alto costo"(2).

El desarrollo e implementación de una instalación de esta naturaleza es una tarea compleja, ya que intervienen una gran cantidad de factores a considerar. Desde equipo, medios de comunicación, protocolos, desarrollo de métodos de ruteo, aplicaciones, etc.

En vista de tal complejidad, no se pretende implementar en esta tesis, un ambiente completo de una red, sino un medio básico y sencillo para la comunicación de datos entre computadoras personales, sin incurrir en gastos extras por cuestiones de equipo de comunicación. Dicho medio básico pretende prestar las funciones basicas para la transmisión de mensajes y archivos dentro de la red, de una manera confiable. Asimismo, permite ver las actividades de comunicaciones que se están llevando a cabo dentro del computador, lo cual es útil para fines educativos, y es algo que los programas comerciales, por lo general, no permiten.

(1) Tanenbaúm, S.A. p. 43
(2) Lefkon, D. p. 147

CAPITULO 1.

ESTRUCTURA GENERAL DE LA RED.

En este capitulo se describirá a grandes rasgos las funciones, capacidades y limitaciones del programa controlador de la red. Los conceptos generales en los que se basa el mismo, así como la aplicación de algunos conceptos importantes.

1.1 CARACTERISTICAS GENERALES.

El programa busca proveer una manera básica de comunicación entre varios computadores, sobre la cual se puedan construir aplicaciones que aprovechen los servicios de comunicación que este programa provee.

Para lograr esto se basa en el modelo OSI (OPEN SYSTEMS INTERCONNECTION propuesto por la organizacion ISO: INTERNACTIONAL STANDARDS ORGANIZATION) para arquitectura de redes de computadoras. "Que propone un modelo de siete capas o niveles conceptuales cada una de las cuales realiza una función especifica, pero son interdependientes entre si para lograr la comunicación"(3).

Cuenta con la facilidad de transmisión tanto de mensajes, hasta de un tamaño máximo, como de archivos ya sean de caracteres (ASCII) o bien binarios (objetos o ejecutables) de cualquier tamaño. También permite hacer transmisiones de mensajes a todos los nodos, ("BROADCAST" en inglés) o a un grupo de nodos de la red.

Esta diseñado para funcionar en una disposición (topologia) de anillo entre varias computadoras, las cuales pueden estar agrupadas entre si. De manera que la dirección de cualquier computadora en la red consta de dos números uno para el grupo al que pertenece y otro para el numero de nodo, dentro del grupo.

Para lograr dichas funciones de transmisión de datos efectiva y libre de errores se deben realizar muchas otras actividades "dentro de las cuales se incluyen la detección de errores, mensajes de diagnostico, inserción y remoción de caracteres especiales, retransmisiones y paquetización de los datos, entre otras"(4).

Tiene un interface de usuario mediante ventanas que permite ver que es lo que esta pasando con la comunicación de datos, que errores han ocurrido así como los demás mensajes de diagnostico que el programa envia. También permite mandar paquetes con datos erróneos para probar el funcionamiento de los algoritmos.

(3) Lane, G.M. p. 91,92
(4) Lane, G.M. p. 109

1.2 RELACION DEL PROGRAMA CON EL MODELO OSI.

El programa, como muchos otros no contiene todas las capas del modelo, sino que realiza principálmente las funciones de una capa de encadenamiento de datos "DATA LINK LAYER","que es la encargada de controlar el trafico de los mensajes en el medio de transmisión, para ello provee reglas para mover datos entre los diferentes nodos. De la misma manera esta capa debe controlar el trafico de mensajes sobre la conexión física, otra función importante es el llevar a cabo el enpaquetamiento o "framing" de los datos, de manera que estos puedan ser reconocidos en otros computadores al ser recibidos.También el determinar cuando transmitir o recibir dichos paquetes de información, cuando mandar un "acknowledge" (reconocimiento positivo) por un paquete correctamente recibido o un mensaje de error cuando hubo un error en la transmisión (reconocimiento negativo) "(5).

Sin embargo, también consta de otras funciones que no se encuentran dentro de esta capa, ya que necesita de un interface de usuario para saber que es lo que actividad se necesita que se realice o se esta realizando. Así como alguna manera de realizar la conexión física entre los distintos computadores, que tampoco pertenece a la capa antes mencionada sino que mas bien sería parte de la PHYSICAL LAYER o capa física que es la capa de mas bajo nivel.

Así, fue diseñado un programa que puede servir como una base solida, para el diseño de programas mas avanzados. O bien, que hagan uso de las facilidades que presta para llevar a cabo algún tipo de aplicación, que necesite de algunos servicios de transmisión de datos confiable y libre de errores..

1.3 CONSIDERACIONES GENERALES.

A continuación se detallaran las algunas consideraciones generales para el correcto funcionamiento del programa, así como, la aplicación de estos conceptos básicos.

1.3.1 DETECCION DE ERRORES.

En todo tipo de aplicación la confiabilidad de la información es esencial, esto es especialmente valido en la transmisión de datos. Ya que estos, están expuestos a distintas situaciones que los pueden modificar dando por consiguiente información errónea. Algunos de estos factores son el ruido en la linea, una mala conexión, o bien, algún paquete de información que haya sido transmitido o recibido incompleto.

(5) Lane, G.M. p. 93

"Para evitar dichos problemas se debe de contar con algún método para la detección de errores, que pudiera detectar tanto errores en los distintos caracteres transmitidos, así como, en el paquete a ser transmitido"(6).

El programa cuenta con dicho método que será descrito en un capítulo posterior, el método es el de CRC (Cyclic Redundancy Check) mediante el cual se le agrega un par de caracteres de control a cada paquete el cual permite la detección de una gran cantidad de errores. Además si se encuentra algún error se envía un NACK (reconocimiento negativo) o bien un ACK (reconocimiento positivo) al nodo transmisor (En lo sucesivo se utilizarán las siglas ACK y NACK sin explicación).

1.3.2 TRANSPARENCIA DE LA INFORMACION.

En el caso en que solo se requiere la transmición de caracteres alfanuméricos esto no presenta un gran problema, ya que la posibilidad de que un carácter ha ser enviado se confunda con un carácter de control (Ej. carácter de fin de transmisión) no se presenta. Sin embargo cuando se transmiten archivos que son ejecutables o que contienen caracteres de control, existe la posibilidad de que alguno de estos datos a ser transmitido sea un carácter de control que el programa de red reconoce como tal, dando como resultado una transmisión truncada o con errores.

Para ello se debe considerar algún método para hacer dichos caracteres invisibles al programa de manejo de la red. Este método debe ser el mismo tanto en los nodos transmisores como receptores, para mantener la integridad de la información. Debe de asegurarse que ningún dato vaya a ser confundido con algún carácter de control ya que la transmisión sería errónea.

La manera en que se implementó en el programa se denomina "byte stuffing" ya que inserta caracteres de control en los datos a ser transmitidos cada vez que encuentra algún carácter que podría causar problemas, de la misma manera los quita cuando son recibidos.

1.3.3 CONTROL DE LA SECUENCIA.

"Este es el proceso mediante el cual se asegura que los paquetes lleguen en el orden correcto, y que la transmisión se lleve a cabo con éxito"(7).

Esto implica varias consideraciones, ya que debe haber un monitoreo continuo de los paquetes que llegan en el receptor, para mandar los ACK's necesarios con el propósito de que el transmisor pueda saber el estado de la transmisión. Este debe mantener un registro de los ACK's

(6) Lane, C.H. p. 115
(7) Lane, C.H. p. 116

recibidos para llevar a cabo retransmisiones o bien transmisiones de nuevos paquetes.

Para ello debe haber una comunicación dentro del nodo transmisor acerca de los ACK's y NACK's que llegan para poder tomar las decisiones necesarias respecto a que paquetes transmitir a continuación.

Dentro del programa esto se hizo mediante unas tablas de control tanto en el receptor como en el transmisor. Las cuales llevan la cuenta de los paquetes que se han transmitido, y/o recibido, para tomar decisiones de que se debe transmitir o recibir a continuación.

1.3.4 CONTROL DE LA LINEA Y DE "TIMEOUTS" (MAXIMO TIEMPO DE ESPERA).

"El control de la linea se refiere a cuando el nodo va a transmitir y/o a recibir información, esto esta muy relacionado con el control de "timeouts" que se refiere a la detección de cuando una transmisión a cesado por que el tiempo máximo de espera se ha cumplido"(8).

En este apartado se deben considerar cuando es posible transmitir, cuando la linea esta libre y cuando se debe de esperar, cuando algo se esta recibiendo. Se debe de considerar asimismo, que es lo que hay que transmitir a continuación. Ya sea un paquete del nodo o bien algún paquete que va dirigido hacia otro nodo, para evitar llevar a cabo un "store and forward" (Almacenar temporalmente los paquetes en el nodo y transmitirlos despues de un tiempo) sino que exista un flujo continuo de paquetes dentro de la red.

El programa maneja el control de la linea de la siguiente manera, lo que solo pasa por el nodo y debe ser retransmitido tiene prioridad para ser transmitido, para evitar demoras. Una vez que se ha transmitido se puede enviar las nuevas transmisiones del nodo

Con respecto a los "timeouts" se deben considerar tanto en el transmisor como en el receptor. En el transmisor se darian cuando este deja de transmitir por alguna razón externa, ya sea una falla o una requisición de algún usuario o aplicación. O también, cuando no se hayan recibido ACK's o NACK's del destino de nuestra transmisión ya que puede ser que dicho nodo este teniendo problemas y se debería tratar mas tarde. En el receptor se debe detectar que los paquetes llegan con cierta periodicidad ya que si en determinado momento se tarda demasiado puede ser que la transmisión haya terminado anormalmente.

Dicho control es importante, ya que cuando se inicia una transmisión y una recepción se tiene que tener algún modulo de control del estado de la misma. Por ello si existe un timeout se deben liberar esos módulos de control del estado para evitar que el programa este atendiendo transacciones que terminaron en una forma anormal y puedan degradar el funcionamiento de la red.

El control de los timeouts se lleva a cabo con las tablas de control del transmisor y receptor en las cuales guarda la hora en que se llevo a cabo la ultima transmisión o recepción. Esta hora se compara contra un tiempo preestablecido que si es sobrepasado se considera que la transmisión a terminado anormalmente y se borra la entrada de la tabla de control, para dejarla libre para otra operación.

1.3.5 PAQUETIZACION DE LA INFORMACION.

La paquetización tiene dos razones de ser, principálmente, en el caso de que una transmisión tenga errores, solo se tendrán que retransmitir pocos paquetes. Por otro lado, el permitir el envío de mensajes desde diferentes estaciones, intercalando paquetes de diferentes peticiones en el canal.

Es importante el poder dividir la información en paquetes mas pequeños ya que entre mas grande sea el paquete existen mayores posibilidades de error. De la misma manera si no se paquetiza la información algún nodo de la red podría mandar algún paquete muy grande y monopolizar la linea de manera tal que otros nodos no pudieran transmitir su información. El paquetizar implica la necesidad de tener un mayor control sobre los paquetes, sin embargo permite un mejor desempeño y control de la red.

Por otro lado, para realizar cualquier transmisión es necesario tener información de control de los paquetes para saber su numero de secuencia, tipo de información, destino, origen, etc.

Dentro del programa se separa la información en paquetes hasta de un tamaño máximo predefinido y a ese paquete se le agrega un encabezado que contiene información de control, así como un "terminador" que contiene una marca de fin de paquete así como alguna información para la detección de errores.

1.3.6 DEFINICION DEL PROTOCOLO.

El protocolo dicta las reglas para la correcta operación de un sistema de comunicación de datos. Se debe de definir el protocolo para cada una de las capas que contenga dicho sistema.

La definición del protocolo debe de considerar tanto la sintaxis del mismo como la semántica (acciones a tomar en determinada situación).

La sintaxis se puede definir mediante una gramática formal usando la forma Backus Naur"(9). Mediante esta se definirán los diferentes mensajes a transmitir, así como los elementos que componen dichos mensajes ej. componentes del encabezado, fin de datos, mensajes de ACK, NACK, mensajes de datos, etc.

La semántica define los diversos estados que se pueden dar en un programa de transmisión de datos, así como las acciones a tomar para resolver dichos estados en los que se encuentre. "Para definir estas reglas de procedimiento se puede utilizar el método de una maquina de estado finito o automata, algún lenguaje de programación de alto nivel o bien pseudo-código. En el caso de una maquina de estado finito, cada paso de un procedimiento se puede describir mediante un estado actual, un evento que causa la transición a un nuevo estado y el nuevo estado "(10). Así se representan los diversos estados y las acciones a tomar por el programa de manejo de red, haciendo mas fácil la identificación de los posibles problemas.

1.3.7 EXCEPCIONES Y CASOS ESPECIALES.

"Dentro de todo el diseño del programa, se deben de considerar las excepciones y casos especiales"(11), los cuales podrían causar errores en la transmisión de los datos ya sea por perdida de información o bien por un error en la secuencia. Así se debe considerar estos casos desde el diseño ya que así se esta garantizando la confiabilidad de la información e integridad de los datos.

En este programa se podrían considerar casos especiales, la posibilidad de que un nodo no este conectado a la red, o bien tenga problemas para transmitir o recibir. Los casos en que el encabezado de los paquetes se dañe o tenga errores, o bien que no se encuentre el fin de un paquete. También hay que considerar cuando se hace un "Broadcast" o transmisión a varios nodos. Que nodo va a extraer paquetes que estén ciclando a través de la red, así como la transmisión de un paquete hacia el mismo nodo, entre otros.

1.3.8 TIPOS DE MENSAJES E INTERFACE DE USUARIO.

La interface de usuario que presenta el programa es mediante ventanas. El usuario cuenta con dos ventanas para el manejo, prueba y visualización de mensajes del programa de red, las cuales pueden ser modificadas en tamaño y en su posición dentro de la pantalla.

(9) Lewis G.H. p.100

En una primera ventana es la que permite la comunicación de el usuario con el programa de red; ahí puede teclear los comandos para la transmisión de mensajes o archivos. Ahí también se reciben los mensajes de sintaxis errónea. En una segunda ventana aparecen los mensajes que indican el estado del programa de la red en el nodo en el que se esta trabajando. Estos mensajes son informativos, como el que indica que se recibio un mensaje, o de diagnóstico, indicando algún error.

Esta interface de usuario no forma parte de la capa de encadenamiento de datos (DATA LINK), sino que se podría considerar como un ejemplo muy sencillo de una capa superior en el modelo OSI, tal como la capa de presentación (PRESENTATION LAYER). Lo cual permite ver que el programa puede hacer una interfaco con diferentes capas.

1.3.9 TOPOLOGIA Y CONEXION FISICA.

En lo que se refiere a redes de área local (LANs) existen diferentes tipos de topologías, de acuerdo a el tipo de conexión, velocidad, equipo y aplicación en el que se vayan a implementar.

La topología para la cual este programa esta implementado como un anillo de inserción, donde los mensajes pasan de nodo en nodo para llegar a su destino. El flujo de datos dentro de un nodo de transito, uno para el cual no es el mensaje, es constante ya que mediante la inserción de mensajes se evita el "store and forward", ya que los mensajes en transito tienen prioridad. Por otro lado se evitan las colisiones como en otros tipos de redes (CSMA/CD Carrier Sense Multiple Access Collision Detection).

El programa de manejo de red esta diseñado para trabajar como base para una LAN, ya que solo soporta microcomputadores locales y no remotas, ademas de ser para un numero pequeño de nodos.

"La conexión entre diferentes nodos se realiza a través del puerto asincrono serial de los computadores (interface RS-232), utilizando cable telefónico (Twisted Pair)"(12). Por ello no tiene gran alcance ni gran velocidad. Sin embargo es suficiente para probar el correcto funcionamiento del programa, ademas que el programa se podría modificar para que soportara otro tipo de conexión.

Aunque este tipo de conexión es lenta, se puede utilizar para transmitir archivos y mensajes entre distintos computadores de una manera confiable, debido a los algoritmos de detección de errores, con una inversión mínima cuando una velocidad muy alta no es necesaria.

(12) Lane, G.M. p.83

1.4 AMBIENTE DE DESARROLLO Y TECNICAS UTILIZADAS.

El programa fue desarrollado en para correr en computadoras UNIX PCs de AT&T sobre UNIX V y fue codificado en lenguaje C, utilizando funciones y bibliotecas del mismo sistema operativo.

El lenguaje C fue seleccionado por ser nativo de UNIX lo cual permite una interface sencilla con las utilerías del mismo. Ademas es lenguaje de nivel medio, lo cual nos permite instrucciones de bajo nivel así como facilidades para estructurar un programa. El S.O. UNIX es muy útil ya que es multitarea, así como puede ser utilizado en varios tipos de computadores, lo cual le da flexibilidad. Se trato de minimizar el uso de funciones especificas de la implementaciones de AT&T del S.O. así como de 'C' sin embargo existen algunas referencias especificas que pueden ser modificados sin grandes problemas.

Las técnicas usadas para el desarrollo del mismo fueron varias:

- Notación Backus Naur: para la definición de la gramática formal, es decir los formatos de mensajes, del protocolo.

- Pseudo codigo: para definir las reglas de procedimiento del programa de red. Que mediante esta representación del programa se puede ver cual es la acción a tomar a continuación.

- Enfoque "Top Down": para facilitar la codificación del programa. Que consiste en dividir el problema en partes de lo mas general a lo especifico hasta que se divida en suficientes bloques que estos sean fáciles de codificar.

- Técnicas de Programación Estructurada: para hacer mas sencilla la codificación, prueba y mantenimiento del programa, al estar hecho el programa en base a pequeños módulos que lo hacen mas entendible y facil de mantener y/o corregir.

Así las diferentes técnicas permiten que el programa pueda ser fácilmente expandido y modificado. Lo cual cumple con el concepto de las capas del modelo OSI. Ademas que el uso de constantes en los valores que pueden ser cambiados permiten modificar fácilmente el programa.

1.5 DESCRIPCION GENERAL DE LOS PROCESOS DEL PROGRAMA.

Desde un punto de vista lógico el programa tiene tres procesos principales, ademas de la inicialización.

A continuación se describen dichos procesos:

- INICIALIZACION: Es el modulo que prepara el ambiente para que el programa pueda ser ejecutado.

- ENTRADA: Es el proceso encargado de recibir los comandos del usuario del programa, ademas lleva a cabo el control de los paquetes transmitidos, ACK's recibidos de la información que esta transmitiendo el nodo.

- RECEPTOR: Se encarga de revisar constantemente los datos que están llegando al nodo y decide que hacer con dichos paquetes, ya sea transmitirlos al siguiente nodo de la red, o bien escribirlo en algún archivo.

- TRANSMISOR: Realiza las funciones de transmitir paquetes al siguiente nodo de la red. Tiene comunicación tanto con el proceso de entrada como con el receptor ya que las funciones de transmisión de datos solo las realiza el transmisor.

Así es como el programa tiene diferentes partes las cuales tienen una función especifica, sin embargo están intercomunicadas entre ellas ya que todas necesitan obtener información de otros módulos del programa.

Para la implementación del programa se utilizaron las capacidades multitarea del UNIX, de manera que se ejecuta un proceso padre, (Inicialización y Entrada) y se crean dos procesos hijo (Transmisor y Receptor) así el programa cuenta con tres procesos corriendo al mismo tiempo. Lo cual ayuda a que se pueda recibir o transmitir datos continuamente evitando el "Store and Forward", así como el poder recibir datos del proceso de entrada "al mismo tiempo".

1.6 DESCRIPCION GENERAL DE LOS ALGORITMOS.

En este apartado se explican los algoritmos del programa, para cada uno de sus procesos así como la interacción de los mismos.

Programa Principal.

i. - Inicializar el ambiente para la ejecución del programa.
ii. - Crea el proceso Transmisor.
iii. - Crea el proceso Receptor.
iv. - Ejecuta el proceso de Entrada.
v. - Termina a los procesos hijo y termina la ejecución.

Entrada (Proceso).

i. - Lee un comando del teclado.
ii. - Revisa la sintaxis del comando.
iii. - Ensambla el paquete a ser enviado.
iv. - Abre una entrada en la tabla de ACK's.
v. - Envia el paquete al proceso transmisor.
vi. - Revisa si el receptor ha recibido algo y actualiza la tabla de bloques.
vii. - Revisa la tabla de ACK's para hacer retransmisiones, actualizarla, etc.
viii.- Si el comando leido = "down" entonces regresa al programa principal.

Transmisor (Proceso).

i. - Revisa si el proceso de entrada le ha enviado paquetes para transmitir.
ii. - Revisa si el receptor le ha enviado paquetes para transmitir.
iii. - Continua el proceso hasta que es cancelado por el proceso padre.

Receptor (Proceso).

i. - Lee el encabezado de los datos del puerto.
ii. - Revisa el encabezado para decidir que hacer con el paquete.
iii. - Dependiendo de lo que traiga el encabezado realiza una o varias de las siguientes acciones:
  - Retransmite el paquete.
  - Lee el paquete y escribe a la pantalla o a un archivo.
  - Envia un ACK.
  - Envia un NACK.
  - Despliega el mensaje apropiado.
  - Envia información al proceso de Entrada.
iv. - Continua leyendo hasta que es cancelado por el proceso padre.

1.7 PRINCIPALES ESTRUCTURAS DE DATOS.

A continuación se hace una descripción de lo se refiere a las principales estructuras de datos:

- La tabla de Transmisión. Que contiene la información acerca del siguiente paquete a ser transmitido, así como los paquetes transmitidos para de los cuales no se han recibido ACK's.

- La tabla de Recepción. La cual lleva un registro de los paquetes recibidos, así como cual es el siguiente paquete a recibir.

- Encabezado. Contiene la información que define a un paquete, tal como el nodo que lo origino, el destino, numero y tipo de paquete entre otra.

- Buffers. Estos son colas para el almacenamiento temporal de los datos.

CAPITULO 2.

DEFINICION DEL PROTOCOLO.

En este capitulo se expondrán las bases sobre las cuales se diseño el protocolo con el cual va a trabajar la red. Se establecerán las razones para cada uno de los campos de datos para los mensajes, caracteres de sincronía etc. Ademas de la definición del esquema para la detección de errores. En la primera parte se describen los campos de los paquetes en una forma aislada, y al final del capitulo se presenta la definición formal del protocolo completo.

2.1 ELEMENTOS DE LOS PAQUETES.

La información que transmite y recibe el programa de red se basa en paquetes de datos los cuales son armados por el nodo transmisor. Dichos paquetes contienen tres elementos principales, los cuales tienen una función especifica y son:

<Paquete> ::= <Encabezado><Cuerpo><Terminador>

i.- Encabezado: que contiene los datos que identifican al paquete asi como la información del origen y destino del mismo. También contiene información de control para el control de la red.
ii.- Cuerpo del paquete: Que son los datos ha ser transmitidos, propiamente, incluyendo algunos caracteres de control.
iii.- Terminador: son los datos que indican el final del paquete, ademas de contener los datos para la detección de errores.

2.2 DESCRIPCION DETALLADA DE LOS CAMPOS DE LOS PAQUETES.

2.2.1 ENCABEZADO.

El encabezado es una parte esencial ya que contiene la información necesaria para la transmisión y recepción correcta de los datos. A continuación se describirán los campos que lo componen.

Así tenemos la definición del encabezado:

<Encabezado> := <Destino><Origen><IdM><NumPaq><Tipo><Cuenta>

- Destino: Son dos variables de un byte. Las cuales contienen la dirección (grupo y nodo) a la cual va dirigido el paquete.

Tanto el nodo como el grupo deben estar en un rango del 0 al 255, que son los máximos números representables mediante un carácter de 8 bits. Estos campos son caracteres pero se consideran por su valor del código ASCII para determinar la dirección del destino.

El destino es leido por los receptores de los nodos para tomar las decisiones necesarias sobre que hacer con el paquete. Ya sea transmitirlo mas adelante en la red, o recibirlo y guardarlo.

<Destino> := <Grupo><Nodo>

<Grupo> := <Carácter>
<Nodo> := <Carácter>

- Origen: Son dos variables de un byte. Las cuales contienen la dirección (grupo y nodo) de la cual fue transmitido el paquete. Tiene la mismas características del destino.

<Origen> := <Grupo><Nodo>

<Grupo> := <Carácter>
<Nodo> := <Carácter>

Este campo es leido por los nodos receptores, para decidir que se hace con el paquete. ej. Si el nodo receptor es el nodo que envio el paquete, este debe sacar el paquete de la red ya que todo parece indicar que ese nodo destino no esta conectado y el paquete ha dado una vuelta completa a la red .

- IdM (Identificación de Mensaje): Este campo consta de cuatro caracteres, y contiene el numero de mensaje o archivo que se esta transmitiendo. Este campo es un numero consecutivo que no se repite en la sesión del transmisor, es utilizado para diferenciar diferentes transmisiones entre dos nodos.

Es utilizado principalmente para el caso de transmisión de archivos, ya que en este caso debe existir una manera de saber a que archivo corresponde un paquete. Esto se debe a que solo el primer paquete en la transmisión de un archivo tiene el nombre del mismo y los subsecuentes paquetes no lo tienen; sino que se identifican por el IdM asignado a el primer paquete.

<Id M> := <Carácter><Carácter><Carácter><Carácter>

Este campo es también un numero convertido a caracteres, por lo cual se debe de considerar como un entero, mas para la transmisión se convierte a caracteres.

- NumPaq (Numero de Paquete): Este campo es un numero convertido a cuatro caracteres. Este campo es utilizado para indicar el orden de los paquetes que se están enviando y recibiendo. El orden es ascendente y consecutivo.

<NumPaq> := <Carácter><Carácter><Carácter><Carácter>

Es necesario para saber si algún paquete se perdió en la transmisión, ej. si llegan los números no consecutivos. De este numero dependen varias acciones, como el escribir el paquete a un archivo, enviar un ACK si es el paquete correcto, un NACK si esta fuera de orden y descartar el paquete erróneo, entre otras.

Los paquetes son iguales en cuanto a su estructura, solo que el primer paquete o paquete cero para el caso de transmisión de archivos solo contiene el nombre del archivo. Esto es para hacer mas sencilla la transmisión ya que si no cada paquete debería tener el nombre de archivo al que pertenece, aumentando así la cantidad de información de control transmitida con cada paquete. Esto se evita teniendo el IdM para identificar los paquetes que pertenecen a un mismo archivo.

- Tipo : Este campo es un carácter, que solo puede tomar siete valores diferentes.

<Tipo> := <MACK>|<MNACK>|<MSG>|<ACKF>|<NACKF>|<FILE>|<LAST>

<MACK> := Acknowledge de un mensaje.
<MNACK> := Acknowledge Negativo de un mensaje.
<MSG> := Mensaje
<ACKF> := Acknowledge de un archivo.
<NACKF> := Acknowledge Negativo de un archivo.
<FILE> := Archivo.
<LAST> := Ultimo Paquete.

Cada uno de estos valores representa a un tipo de paquete diferente, estos tipos se pueden agrupar de diferentes maneras, en este caso se agruparon según el tipo de datos a transmitir (mensaje o archivo).

Cada uno de los tipos de transmisión cuenta con sus ACKs y NACKs, además de existir el tipo LAST que indica el ultimo paquete de la transmisión.

Dependiendo del tipo de paquete el receptor toma decisiones. Así si el mensaje es de tipo archivo va a escribir el paquete en un archivo. Si es un mensaje despliega dicho mensaje en la pantalla. Si es el ultimo paquete de un archivo lo escribe y cierra el archivo. Si es un ACK o NACK entonces manda el paquete al proceso de entrada para que tome decisiones sobre que es lo que hay que transmitir a continuación.

- Cuenta (Cuenta de saltos): Este campo es un numero del 0 al 255 convertido en carácter.

<Cuenta> := <Carácter>

La Cuenta contiene el numero de nodos por los que ha pasado el paquete. Es útil para el caso en que por alguna causa no se ha encontrado ni el nodo destino ni el transmisor, Ej. cuando se dañan dichas partes del encabezado. En dicho caso el paquete continuaría dando vueltas dentro de la red sin que ningún nodo lo retirara, lo cual causaría un aumento en trafico de la red.

Así es como cada nodo que recibe el paquete revisa este campo y si es mayor que el numero de nodos predefinido, retira el paquete de la red desplegando el mensaje adecuado. Si la Cuenta no es mayor que dicho limite y el mensaje ha de ser retransmitido entonces aumenta dicho valor en uno.

2.2.2 CUERPO DEL PAQUETE.

Esta es propiamente la información ha ser transmitida, ademas contiene caracteres de control para mantener la transparencia de los datos así como para evitar que exista que algún carácter que se confunda con el carácter de fin de transmisión.

<Cuerpo> := <Datos>...<Datos>

- Datos : Es el conjunto de caracteres que forman el cuerpo del paquete pude ser cada una de las siguientes combinaciones.

<Datos> := <Carácter>|<Quote><Quote>|<Quote><ETB>

Todos los caracteres del código ASCII, de 8 bit, se pueden incluir dentro de Datos sin embargo se hace una tratamiento especial con algunos caracteres que se tratan de una forma especial como lo son (QUOTE y ETB).

<Quote> := Carácter para enmascarar a otro carácter de control.
<ETB> := End of Transmission Block (Indica el fin de transmisión).

- ETB : Es el carácter 23 dec. (End of Transmission Block) que se utiliza para indicar que es el fin del paquete.

Este carácter se debe de enmascarar ya que si dentro de los datos ha ser transmitidos existiera esto, se consideraría que el paquete ha terminado, aunque este no fuera el caso.

Por ello si se encuentra un carácter ETB, al momento de la transmisión, se le enmascara con un carácter de Quote. Lo que ocasiona un proceso especial de la pareja de caracteres tanto en el receptor como en el transmisor.

- Quote: Es el carácter 16 dec. o el DLE (DELETE) se utiliza para enmascarar a otro carácter de control. En el programa de red solo existen dos caracteres a enmascarar el mismo Quote (DLE) y el ETB, como se mencionó anteriormente.

El enmascaramiento del ETB se vio anteriormente, mas el enmascaramiento del Quote se debe a que, si se encuentra este carácter dentro de los datos se desecha este y el siguiente carácter se toma como valido. Por otro lado si no se enmascarara un Quote se desecharía un carácter valido, lo cual ocasionaría un error en la transmisión/recepción.

2.2.3 TERMINADOR.

Esta es la parte mas pequeña de los paquetes consta solo de tres caracteres.

<Terminador> := <CRC><ETB>

- CRC : Este campo contiene un par de caracteres, utilizados para la detección de errores, los cuales son resultado de aplicar las formulas para el Cyclic Redundancy Check (CRC) a el encabezado y los datos antes de llevar a cabo el enmascaramiento.

<CRC> := <Carácter><Carácter>

- ETB : es el carácter que indica el fin del paquete, el cual no es precedido por el carácter Quote.

El método utilizado para la detección de errores en este programa es el CRC-16 (Cyclic Redundancy Check) "que consiste en obtener un par de caracteres de prueba, los cuales se le agregan al final de los datos que van a ser transmitidos. Se denomina CRC-16 ya que se obtiene con un polinomio de grado 16"(13).

Así el algoritmo para obtener el CRC se aplica en el nodo receptor y si es el mismo CRC es que la transmisión se llevo a cabo libre de errores, por el contrario si es diferente entonces hubo cuando menos un error.

La teoría para el calculo del CRC es explicada a continuación:

i) Si $M(x)$ es el mensaje a ser transmitido, entonces hay que multiplicar $M(x)$ por x^g, donde g es el grado del polinomio generador $G(x)$, dando como resultado a g 0s en las posiciones de mas bajo orden (ej. que $x^g * M(x)$ tenga ceros en los g coeficientes de mas bajo orden).
ii) Dividir $x^g * M(x)$ entre $G(x)$. Esto da como resultado un cociente único $Q(x)$ y un residuo $R(x)$:

$$\frac{x^g * M(x)}{G(x)} = Q(x) \quad MOD2+ \quad \frac{R(x)}{G(x)}$$

Nota: *MOD2+* significa suma en modulo 2.

iii) Adicionar el residuo a $x^g * M(x)$ (el mensaje), dejando así hasta g terminos con coeficientes unitarios en las posiciones de mas bajo orden. Entonces el mensaje a ser transmitido es:

$$T(x) = Q(x) \; MOD2+ \; R(x)$$

Hay que notar que tenemos:

$$\frac{x^g * M(x)}{G(x)} = Q(x) \quad MOD2+ \quad \frac{R(x)}{G(x)}$$

Así, que multiplicando ambos lados de la igualdad por G(x) se obtiene:

$$x^g * M(x) = Q(x) * G(x) \; MOD2+ \; R(x)$$

Entonces si sumamos R(x) a ambos lados da como resultado:

$$x^g * M(x) \; MOD2+ \; R(x) = Q(x) * G(x)$$

(13) Lane, G.M. p.53

ya que *MOD2+* y *MOD2-* son lo mismo, entonces el mensaje a ser transmitido es:

$$T(x) = x^g * M(x)\ MOD2+ R(x) = Q(x) * G(x)$$

y el mensaje transmitido es exactamente divisible por $G(x)$. Por lo que en el receptor, el "software" puede dividir $T(x)$ entre $G(x)$ y puede revisar que el residuo sea cero, para probar que la transmision se halla llevado sin errores.

Se utilizo este método de detección de errores ya que como dice LANE "Actualmente es uno de los mas utilizados dentro de las comunicaciones de datos."(14) Ademas de que permite detectar gran cantidad de errores, como lo sugiere la siguiente tabla:

Si g= Grado del polinomio para obtener el CRC.

g= 16 en nuestro caso.

Tipo de error	Probabilidad de Detección.
+ Error en bits únicos.	100%
+ Error en dos bits.	100%
+ Numero impar de bits en error.	100%
+ Conjunto de bits en error de longitud < $g+1$	100%
+ Conjunto de bits en error de longitud = $g+1$	$1-(1/2)^{g-1}$
+ Conjunto de bits en error de longitud > $g+1$	$1-(1/2)^{g}$

Donde la longitud de un conjunto de bits en error es: el numero de bits empezando en el primer bit en error hasta el ultimo bit en error inclusive.

Hay que notar que como algunos patrones de error son mas comunes que otros las probabilidades pueden variar. También es importante mencionar que la probabilidad de detección de conjuntos de bits en error da mas de 16 bits es mayor que 99% utilizando este tipo de detección de errores.

La tábla anterior, asi como la teoría se tomarón de DATA COMMUNICATION SOFTWARE DESIGN de Malcom G. Lane.(15)

La gran capacidad de detección de errores del CRC-16 permite que el programa de red sea muy confiable en cuanto a la validez de sus datos.

2.3 DEFINICION FORMAL DEL PROTOCOLO.

A continuación se hace la definición formal del protocolo, junto con una descripción de los símbolos utilizados en esta.

```
<Paquete>     := <Encabezado><Cuerpo><Terminador>

<Encabezado> := <Destino><Origen><IdM><NumPaq><Tipo><Cuenta>
<Destino>     := <Grupo><Nodo>
<Origen>      := <Grupo><Nodo>
<IdM>         := <Carácter><Carácter><Carácter><Carácter>
<NumPaq>      := <Carácter><Carácter><Carácter><Carácter>
<Tipo>        := <MACK>|<MNACK>|<MSG>|<ACKF>|<NACKF>|<FILE>|<LAST>
<Cuenta>      := <Carácter>
<Grupo>       := <Carácter>
<Nodo>        := <Carácter>

<Cuerpo>      := <Datos>...<Datos>
<Datos>       := <Carácter>|<Quote><Quote>|<Quote><ETB>

<Terminador> := <CRC><ETB>
<CRC>         := <Carácter><Carácter>

<Carácter>    :=.00000000b|...|11111111b      b = binario
                                              Juego Caracteres ASCII.

<MACK>        := 00010100b                      = 20 dec.
<MNACK>       := 00010110b                      = 22 dec.
<MSG>         := 00000010b                      =  2 dec.
<ACKF>        := 00001010b                      = 10 dec.
<NACKF>       := 00001011b                      = 11 dec.
<FILE>        := 00000001b                      =  1 dec.
<LAST>        := 00000011b                      =  3 dec.

<Quote>       := <DLE>                          = Delete Carácter
<DLE>         := 0010000b                       = 16 dec.
<ETB>         := 0010111b                       = 23 dec.
```

```
:=      Significa "Esta formado por:"
<a><b>  Significa "<a> seguido por <b>"
< >     Significa "Variable"
<a>|<b> Significa <a> o <b>
        Símbolos que no están definidos en <>
         son literales o constantes.
...     Significa "Desde hasta"
```

La notación anterior, se tomo de DATA COMMUNICATIONS SOFTWARE DESIGN de LANE (15).

(15) Lane, G.M. p.123

CAPITULO 3.

DEFINICION DE LA MAQUINA DE ESTADO FINITO.

En este capitulo se definirán a detalle los diversos pasos de cada uno de los procesos así como las acciones a tomar por la red dependiendo de el estado en que se encuentre. Se presentarán los algoritmos en pseudo-código explicando los diversos pasos de los componen así como las razones de estos pasos.

Estos algoritmos constituyen las reglas de procedimientos a las cuales se hacia mención en el capitulo 1. Mediante las cuales se explica el comportamiento del programa. "Esta descripción de los estados del programa junto con la definición formal de la sintaxis del protocolo serán las bases para llevar a cabo la codificación y prueba del mismo"(17).

3.1 DESCRIPCION DETALLADA DE LOS ALGORITMOS DEL PROGRAMA.

Se empezará por describir estos algoritmos en pseudo-código ya que es una manera mas intuitiva de entender el proceso que se esta llevando a cabo.

Para la descripción del programa se empezara de lo mas general a lo mas detallado siguiendo una filosofía "TOP-DOWN".

3.1.1 PROGRAMA PRINCIPAL

i) Inicialización del Ambiente.
   - Abrir ventanas.
   - Definir el Puerto (E/S).
   - Definir medios de Comunicación entre Procesos.
ii) Crear el Proceso Transmisor.
iii) Crear el Proceso Receptor
iv) Llamar al Proceso de Entrada.
v) Terminar los Procesos "Hijos" y Terminar ejecución.

(17) Lee, G H. p.127

- Inicialización del Ambiente.

Al igual que todo otro programa, este necesita crear un ambiente sobre el cual va a ejecutarse. En este caso ademas de la inicialización de variables se deben definir y abrir las ventanas para la comunicación con el usuario. Definir el puerto por el cual se va a llevar a cabo la transmisión/recepción de datos, en este caso el puerto serial del computador. También como el programa va a tener tres procesos ejecutandose simultáneamente y estos necesitan estar en comunicación constante se debe definir este medio de comunicación inter-procesos.

- Creación de Procesos Adicionales.

El programa principal crea los otros dos procesos a partir de si mismo, lo que implica que tienen variables comunes que están inicializadas al momento de la creación. Esto es necesario para que todos conozcan los medios de comunicación inter-procesos, ventanas a las que hay que escribir, dirección del nodo, etc.

- Proceso de Entrada.

Este proceso es una función llamada desde el programa principal y es la que tiene el control de todo el programa, ya que solo ella puede terminar con los procesos "hijos".

- Terminación de los procesos hijos y del programa principal.

Una vez creados los procesos hijos estos se están ejecutando continuamente, uno tras otro según el Sistema Operativo les pasa el control. Se puede decir que son procesos interminables,(Un loop infinito) ya que no tienen manera de terminarse ellos solos.

Por ello existe el programa principal que maneja la entrada de datos del usuario. Esta función de Entrada se ejecuta constantemente hasta que recibe el comando de terminar el programa, lo cual le permite regresar al programa principal que termina los procesos "hijos" libera los recursos en uso y entrega el control al S.O.

3.1.2 PROCESO DE ENTRADA.

i) Lee un comando del Teclado.
ii) Revisa si es un comando valido.
   - Revisa la Sintaxis.
   - Revisa si el Grupo y Nodo están dentro del rango valido (0-255).
iii) Ensambla el paquete ha ser enviado.
   - Arma el encabezado de acuerdo al comando.
   - Calcula el CRC en el Encabezado+Datos.
   - Adiciona el CRC al paquete.
iv) Crea una entrada en la Tabla de ACKs para el paquete.
v) Envia el paquete al proceso Transmisor.
   - Envia el encabezado (Sin enmascaramiento).
   - Envia los datos (Enmascarandolos al mismo tiempo).
   - Envia un ETB para señalar el fin del paquete.
vi) Revisa si el Proceso receptor ha enviado algún paquete.
vii) Revisa la Tabla de ACKs.
   - Actualiza la tabla de ACKs.
   - Si se recibio un NACK retransmitir.
   - Si se recibio un ACK liberar la entrada en la tabla.
   - Si existe un "timeout" retransmitir el paquete.
   - Si existen paquetes ha ser transmitidos, transmitir el siguiente.
viii) Si se leyó el comando de terminación, regresar el control al programa principal, sino leer otro comando.

- Lectura de Comandos y Revisión de Sintaxis.

En este paso el proceso de entrada revisa la ventana de entrada, para ver si el usuario no ha tecleado algún carácter, si se ha terminado de teclear (se tecleo un <CR>) entonces se pasa a la revisión de sintaxis. Si ha tecleado algún carácter entonces dicho carácter se guarda en un "buffer" temporal y el proceso de entrada continua ejecutandose, realizando funciones de revisión de tablas, retransmisión, etc.

En la revisión de sintaxis revisa que el comando sea valido. Revisa que sea para enviar un mensaje, un archivo o bien el comando para terminar la sesión. Revisa que tanto el nodo como el grupo sean un numero valido (entre 0 y 255), ademas que revisa si se quiere alguna opcion especial para dicha transmisión (para realizar pruebas).

- Ensamblo del paquete ha ser enviado.

Una vez que el comando fue aceptado como valido se contruye el paquete que va a ser enviado. Se empieza por el encabezado según el comando que se tecleo. Este se arma poniéndole las direcciones destino y origen el Identificador de Mensaje, tipo de mensaje, etc. Si es algún caso de prueba entonces se crea este mensaje con las caracteristicas pedidas.

En el caso de transmisión de archivos, el paquete solo lleva en los datos el nombre del archivo a ser transmitido. Pero el programa verifica que dicho archivo se pueda abrir (exista, no este dañado) antes de enviar dicho paquete. Si este archivo no es utilizable entonces se cancela la transmisión avisandole al usuario.

Una vez construido el encabezado se le adiciona el cuerpo del mensaje y se obtienen los caracteres del CRC los cuales se agregan al final del paquete.

- Abrir una entrada en la tabla de ACKs.

Cuando se va a enviar un archivo se debe mantener un registro de los paquetes que se han enviado, así como cuantos faltan por mandar y cual es el siguiente a ser transmitido. Esta información se almacena dentro de una tabla llamada de ACKs, la cual tiene la información que ha sido enviada, cual es el paquete ha enviar a continuación así como el Identificador de Mensaje (IdM) y el archivo asociado a esa transmisión.

Así cuando se empieza la transmisión de un archivo, se abre su entrada correspondiente en la tabla de ACKs, para llevar el control de su transmisión. Es también necesaria ya que una vez que se ejecuta el comando, la información acerca del archivo a enviar se pierde y solo queda dentro de la tabla.

- Envio del paquete al proceso transmisor.

Una vez que se le adiciono el CRC al paquete, y se creo la entrada en la tabla de ACKs, el paquete esta listo para ser enviado al proceso transmisor.

Primero se envia el encabezado sin enmascaramiento, ya que este debe tener una longitud fija. En este caso no representa un problema el carácter ETB, ya que no se considera como fin de transmisión sino como el carácter que representa.

A continuación se envia el cuerpo del paquete, aqui es cuando se lleva a cabo el enmascaramiento de los datos (ETB y DLE). Por ultimo se envia un carácter ETB (sin enmascaramiento) para señalar el fin del paquete.

- Revisa si el proceso receptor a enviado algún paquete.

En este paso revisa si el nodo a recibido un ACK o NACK de alguna transmisión a través del proceso receptor. Si existe algo entonces actualiza la tabla.

- Revisa la Tabla de ACKs.

En este paso la tabla de ACKs es revisada para ver el estado de las diferentes transmisiones y de acuerdo a eso tomar las acciones requeridas:.

- Si llego un ACKF entonces libera el espacio de la tabla que lo corresponde a ese mensaje.
- Si llego un NACKF entonces revisa si el paquete no ha sido retransmitido mas veces del limite. Si tiene oportunidades entonces lo retransmite sino deja de transmitir y libera el espacio ocupado por sus entradas.
- Si existe alguna entrada libre para un archivo en proceso de transmisión, entonces envia el siguiente paquete de dicho archivo, hasta llenar los bloques que corresponden a ese archivo. (Notas)
- Revisa el tiempo en que se envio el ultimo paquete y si es mayor al tiempo especificado de "timeout" entonces retransmite el paquete. A menos que ya se haya retransmitido anteriormente mas veces del limite fijado.

Notas:

- Cada archivo a ser transmitido tiene un espacio limitado de bloques en la tabla de ACKs, para guardar temporalmente los paquetes que ha enviado hasta que se recibe el ACK de ese paquete. Entonces este bloque es liberado y otro paquete cargado en el.
- La razón de que existan varios bloques para un archivo es la siguiente: cuando se envía un paquete se espera un ACK para ese paquete. Pero existe un retraso entre el momento en que se manda el paquete y el tiempo en que llega el ACK. Entonces es deseable permitirle al nodo transmisor ir un par de paquetes adelante de los ACKs que llegan. En otras palabras el nodo transmisor no necesita esperar el ACK del paquete N para enviar al paquete (N+1).

- Lectura del Comando de terminación.

Si el usuario da el comando de terminación la función de Entrada termina y regresa el control al programa principal el cual a su vez termina con los procesos hijos y termina.

3.1.3 PROCESO TRANSMISOR.

i) Revisa si el proceso de Entrada mandó algún paquete.
   - Lee el encabezado, enviado por el Proceso de Entrada.
   - Escribe el encabezado al puerto.
   - Lee el paquete hasta que encuentra un carácter ETB.
   - Al mismo tiempo escribe el paquete al puerto.
     (Tomando en cuenta los caracteres ETB y DLE).

ii) Revisa si existe algún paquete del Proceso Receptor.
    (Para retransmisión, ACK o NACK).
   - Lee el Encabezado enviado por el Proceso Receptor.
   - Escribe el Encabezado al puerto.
   - Lee el paquete hasta que encuentra un carácter ETB.
   - Al mismo tiempo escribe el paquete al puerto.
     (Tomando en cuenta los caracteres ETB y DLE).

iii) Continua con el mismo proceso hasta que es terminado por el proceso padre.

Este proceso es bastante sencillo ya que solo se encarga de transmitir hacia el exterior del nodo, los paquetes que tanto el proceso receptor como el transmisor le envían ya preparados.

- Revisa si existe algún paquete por transmitir.

La principal función es revisar constantemente la comunicación con el proceso receptor y de entrada, para ver si existe un paquete a transmitir.

- Transmisión de Paquetes.

Independientemente de que proceso requiera hacer la transmisión, el procedimiento para transmitirlo es el mismo.

Primero se transmite el encabezado tal y como viene, a continuación se manda el cuerpo del paquete, en este caso si se debe tener en cuenta el enmascaramiento. Aunque el enmascaramiento se realiza tanto en otros nodos como en el propio proceso de entrada se debe tomar en cuenta para poder identificar el "verdadero ETB" o fin del paquete.

3.1.4 PROCESO RECEPTOR.

i) Lee el Encabezado del puerto.
ii) Analiza el encabezado para decidir que hacer con el paquete.
    - Revisa si es un Mensaje o un Archivo.
iii) Dependiendo del encabezado existen diferentes acciones a tomar:
    - Retransmitir el paquete.
    - Leer el paquete (quitando los DLEs) y escribirlo en la pantalla.
    - Leer el paquete (quitando los DLEs) y escribirlo en un archivo.
    - Enviar un ACK.
    - Enviar un NACK.
    - Desplegar un mensaje en la pantalla.
    - Enviar información al proceso de Entrada.
    - Actualizar la información de la tabla de Recepción.
    - Alguna combinación de los anteriores.
iv) Revisa la tabla de Recepción buscando "timeouts"
v) Continuar procesando el puerto hasta que el proceso padre termine el proceso.

- Lectura de Caracteres del Puerto.

Este proceso revisa constantemente el puerto del computador y guarda en un "buffer temporal lo que esta llegando. Una vez que tiene el encabezado lo analiza para decidir que acción tomar.

- Análisis del Encabezado y Acciones a tomar.

Según los datos que tenga el encabezado se van a tomar varias acciones, ya que existen dos tipos principales de transmisiones, archivos y mensajes los cuales se deben procesar de una manera diferente.

- Procesamiento de un paquete.

A continuación están las diferentes acciones a tomar durante el proceso de un paquete. Existen algunas que son exclusivas para el procesamiento de mensajes y otras solo para archivos, esas se señalaran y explicaran aparte.

- Si el paquete es para este nodo entonces se debe leer y escribir en un archivo (1).
- Si el paquete tiene errores entonces enviar un NACK.
- Si el paquete se recibio sin errores enviar un ACK.
- Si el paquete fue mandado por nosotros, entonces se debe leer y descartar de la red. Ya que el nodo al que lo mandamos parece ser que no esta conectado ya que el paquete dio toda la vuelta a la red.
- Si el paquete no es para el nodo entonces hay que retransmitir.
- Si el paquete tiene un campo de "Cuenta" mayor que el permitido, entonces sacarlo de la red ya que probablemente su encabezado tiene errores o bien el nodo que lo mando no esta conectado en este momento.

Opciones para mensajes únicamente.

- Si el paquete es un "Broadcast" entonces se debe leer y escribir en la pantalla, ademas de retransmitirlo para los demás nodos.

Opciones para archivos únicamente.

- El primer mensaje de un archivo, contiene el nombre del archivo. Entonces al recibirlo se adiciona una entrada en la tabla de recepción y se abre el archivo donde se van a escribir los datos de los paquetes subsecuentes.

- Al recibir paquetes de tipo archivo primero se busca en la tabla de recepción para ver si es el paquete que se espera. Ya que solo pueden recibirse en orden. Para saber si llego en orden se revisa la tabla de recepción.

Aqui existen dos posibilidades :

a) Si el número de paquete es menor del que esperamos se descarta, sin mandar ninguna paquete ya que ese paquete ya se recibio correctamente. Si el número es mayor que el numero de paquete que esperamos entonces se descarta y se manda un NACKF para el paquete que se está esperando. Esto para forzar la retransmisión del siguiente paquete que es el que nos interesa que llegue a continuación.
b) Si llego en orden entonces se manda un ACKF y se escribe en el archivo correspondiente.

- Cuando se recibe el ultimo paquete correctamente, se escribe al archivo, se cierra, se libera la entrada en la tabla de recepción, y se manda un ACKF.

- Cuando se recibe un ACKF o NACKF para el nodo entonces se envia un paquete al proceso de entrada para que actualizar su tabla de ACKs.

Notas:

(1) Archivo puede ser tanto la pantalla, para el caso de los mensajes, como un archivo en disco.

- Unicamente cuando un paquete es para "nuestro" nodo es cuando se realiza el deconmascaramiento y se calcula el CRC para verificar que se recibio libre de errores.

- Si el paquete recibido tiene errores no es escrito en un archivo.

- Cada vez que se recibe algún paquete se despliega un mensaje en la ventana del usuario para informarle que es lo que esta pasando.

- Si se recibe un archivo con un nombre que ya existe entonces se escribe sobre el archivo existente.

- Si se recibe un archivo para el nodo y ese mismo nodo fue quien lo envio, entonces no se escribe sobre el archivo existente.

- Revisión de Tabla de Recepción.

La tabla de recepción es revisada periodicamente en busca de "timeouts" del transmisor. Estos son cuando el transmisor deja de enviar paquetes. Para evitar que existan entradas en la tabla de archivos que se dejaron de transmitir se compara el tiempo del ultimo paquete recibido con un limite pre-determinado y si es mayor, entonces se cierra el archivo y se libera la entrada en la tabla.

Aqui se considera que aunque un paquete haya llegado con errores se debe modificar el tiempo, ya que se sigue teniendo contacto con el nodo transmisor. Esto se hace para evitar que se termine la recepción por un "timeout".

- Terminación del proceso receptor.

El proceso receptor continua trabajando continuamente, revisando el puerto y procesando mensajes, hasta que es terminado por el proceso padre.

CAPITULO 4.

IMPLEMENTACION DE PROCESOS.

En este capitulo se describirán los detalles de la codificación de los algoritmos que constituyen el programa manejador de red. Se explicaran como se implementaron ciertos conceptos, el uso y descripción de las estructuras de datos, las llamadas al sistema, así como los procedimientos que son especificos para el UNIX V de AT&T.

4.1 DESCRIPCION DE LAS ESTRUCTURAS DE DATOS.

4.1.1 Tabla de ACKs.

Como se explicó anteriormente, la tabla de ACKs es utilizada para mantener el control de la transmisión que se lleva a cabo. Es un arreglo de un numero determinado de entradas. Cada una de las entradas contiene la siguiente información:

- Encabezado
  Es un arreglo de 10 caracteres para contener el encabezado de la transmisión a que pertenece. Este va cambiando con cada uno de los paquetes que se envia, sin embargo mucha de su información se mantiene constante, tal como el destino, fuente, etc.

- Fd (File Descriptor).
  Todos los archivos que se manejan en el programa, pantalla, puerto, o archivos de disco, se abren mediante rutinas de bajo nivel. Las cuales identifican el archivo mediante un numero entero único en el sistema, este se llama el file descriptor (Identificador de Archivo). Y es utilizado para cualquier acceso a cualquier archivo.

- Siguiente.
  Es el numero de paquete que se va a enviar a continuación.

- Ultimo.
  Es el numero del último paquete que se ha recibido con éxito en el receptor (Se ha recibido un ACK).

- Libre.
  Indica si la entrada de la tabla esta libre para ser utilizada, 1=Libre, 0=Ocupada.

- Numero de Paquetes.
  Contiene el numero de paquetes que componen al archivo que se va a transmitir. Es útil para saber cual paquete es el último y construir su encabezado.

- Nombre de archivo.
  Aqui se guarda el nombre del archivo que va a ser enviado.

- Arreglo de Paquetes.
  Para cada una de las entradas de la tabla de ACKs, se tiene un arreglo de paquetes. El cual contiene tanto información de control como los datos que componen a los siguientes paquetes que se van a enviar en seguida. A continuación se describen los campos de este arreglo de paquetes.

  - No. de Paquete. Es el identificador del paquete.
  - Libre. Indica si la entrada puede ser ocupada.
  - Estado. Indica el estado en que se encuentra dicho paquete. Ya sea ACKF, NACKF, etc. Util para saber que es lo que hay que hacer con dicho paquete. ej. retransmitir, esperar, cargar uno nuevo, etc.
  - Intento. Indica las veces que se ha transmitido el paqueto.
  - Tamaño. Indica el tamaño del paquete ha ser enviado.
  - Datos. Es un arreglo de caracteres, que contiene los datos que componen al paquete.
  - Tiempo de Envio. Guarda la hora en que se mandó el paquete. Es utilizado para el control de "timeouts".

4.1.2 TABLA DE RECEPCION.

Esta tabla es utilizada para mantener el control de los paquetes que se están recibiendo. Es un arreglo para contener los datos de control de los archivos que se están recibiendo.Contiene los siguientes datos:

- Encabezado.
  Contiene el encabezado de los paquetes que están llegando y corresponden a una determinada transmisión.

- Libre.
  Indica si dicha entrada de la tabla esta disponible.

- File Descriptor.
  Al igual que en la tabla de transmisión, esto dato indica el archivo al cual se va a escribir. Este Fd es obtenido al abrir al archivo para escritura.

- Siguiente.
  Es el numero de paquete que se esta esperando a continuación.

- Tiempo de recepción.
  Es el tiempo en que se recibio el último paquete, correspondiente a la transmisión. Es utilizado para detectar "timeouts".

4.1.3 BUFFERS DE ENTRADA Y SALIDA.

Existen diferentes buffers o estructuras de almacenamiento temporal, que se utilizan en diversas partes del programa. Son utilizadas principalmente para conservar los datos y procesarlos de alguna manera especial.

La estructura de datos utilizada es una cola circular (QUEUE) la cual trabaja bajo una filosofía FIFO (First In First OUT) que es decir que los primeros elementos que entran son los primeros que salen.

4.1.4 METODO DE INTERCOMUNICACION INTER-PROCESOS (PIPE)

Se mencionó que el programa consta de tres procesos los cuales están interrelacionados entre sí. Para que tengan información actualizada entre ellos, se necesita de un medio de comunicación inter-procesos. El método utilizado es una estructura de datos semejante a una cola ya que trabaja de una manera FIFO. Esta estructura es una facilidad de UNIX la cual sirve para comunicar diferentes procesos, la cual se genera mediante una llamada a una instrucción PIPE.

Cada PIPE tiene dos File Descriptors, uno para lectura y otro para escritura, mediante los cuales los diferentes procesos escriben y leen del PIPE.

4.2 ENTRADA/SALIDA.

- Escritura/Lectura de Pantalla, Archivos y Puertos.

En UNIX la escritura de archivos se realiza mediante un identificador denominado File Descriptor, mediante este se lee o escribe al archivo, este FD se obtiene en el momento de abrir el archivo.

Ademas en UNIX todos los periféricos, incluyendo al teclado, pantalla, puertos, son considerados como archivos así el puerto, pantalla y ventanas del interface de usuario se abren como un archivo y tienen su propio FD.

Para la lectura y escritura del programa se utilizaron rutinas de bajo nivel, instrucciones READ y WRITE, ya que estas permiten tener mayor control sobre la escritura, lectura.

El puerto usado es el puerto serial, denominado TTY000 en UNIX. El cual usa el interface RS-232-C.

En lo que se refiere a la pantalla se abrieron dos "ventanas" mediante la llamada Window al sistema.

- Modo "Raw" (modo de NO traducción de retorno, cambio de linea,etc.) en operaciones de E/S de archivos.

Se menciono que se utilizaban instrucciones de bajo nivel para la E/S de archivos. Ademas de utilizar dichas instrucciones se utilizo el modo "No Traducción" o "RAW", que permite tener control absoluto sobre las operaciones de E/S.

Este modo no reconoce caracteres especiales, ni los procesa de manera especial, esto obliga al programa a tomar acciones para el proceso de los mismos. Esto es especialmente importante durante la escritura al puerto ya que en la transmisión de archivos ejecutables existen caracteres de control los cuales no deben ser considerados como tales, sino como cualquier otro carácter.

Al procesar la entrada de datos a través del teclado, en modo "RAW" se deben tomar consideraciones especiales para el manejo de ciertos caracteres. *ej. el Retroceso o BackSpace, la tecla de Entrada o Return o bien el borrar la entrada o <CTL U>.

La función de UNIX que permite poner los archivos en modo "raw" es la instruccion IOCTL, la cual permite decirle al S.O. como vamos a procesar dicho archivo.

4.3 CONSIDERACIONES ADICIONALES.

- Creación de procesos hijos (FORK).

El programa principal es el proceso que crea los procesos hijos el receptor y transmisor. Estos los realiza mediante la llamada del sistema FORK, la cual crea un nuevo proceso, este nuevo proceso es una copia exacta del proceso padre, de cual hereda todas las caracteristicas, ambiente y variables con que cuenta.

Es importante crear los PIPEs o canales de comunicación inter-procesos antes de ejecutar la llamada a FORK, ya que debe existir primero los FD para poder realizar la comunicación.

- Terminación de procesos hijos (KILL).

El proceso padre debe tener una manera de terminar a los procesos hijos y esta es la instrucción Kill. La cual termina algún proceso ordenadamente, cerrando todo sus FD, liberando la memoria y haciendo demás acciones de limpieza.

La instrucción KILL necesita saber cual es el proceso a terminar, esto lo hace mediante un identificador de proceso o Pid (process ID) el cual se le asigna al proceso al crearlo con el FORK. Estos Pid los guarda el proceso padre para utilizarlos en el momento necesario.

- Detección de transmisión entre procesos (FSTAT).

Para saber si un PIPE tiene información esperando a ser leida se utiliza la función FSTAT, que entre otros valores nos dice que numero de caracteres existen en un archivo dado. Como los PIPEs tienen un FD se puede utilizar esta instrucción para saber si tienen información para ser leida.

También es utilizada esta instrucción para saber cuantos paquetes hay que transmitir de un archivo.

- Almacenamiento de la hora (TIME).

Dentro del control de "timeouts" el poder saber la hora es muy importante, para ello se utiliza la función TIME de "C", que nos devuelve la hora en que esta se invoco.

- Creación de ventanas (WINDOW).

En el caso de las UNIX PCs de AT&T existen funciones para la creación y manejo de ventanas. Estas ventanas se crean mediante la función Window, la cual devuelve un FD que se utiliza para lectura/escritura. Estas ventanas se abren en modo "raw" para tener un control total sobre ellas.

- Conversión de tipos de datos.

Como se ha visto a lo largo de los capitulos, los datos que maneja el programa son esencialmente caracteres. Aunque utiliza números, enteros estos deben ser convertidos a carácter para ser mandados a través de la red.

Así los principales tipos de conversiones de datos son de entero a carácter o viceversa. Para ello se realizaron diferentes funciones a lo largo del programa, para hacerlo mas sencillo. Estas conversiones utilizan únicamente funciones estandard de "C" lo cual las hace portables.

- Constantes utilizadas.

En el programa se utilizan una serie de constantes globales para todos los procesos. Algunas definen caracteres especiales, otras los diferentes tipos de mensajes así como tamaños de paquetes, buffers, tablas, tiempos para "timeouts", datos para el calculo del CRC-16, etc.

Estas constantes están definidas y comentadas dentro del listado del programa. El uso de constantes es sumamente útil, ya que solo se deben de definir una vez y si se quiere cambiar algún valor determinado, solo se debe cambiar el valor de la constante.

4.4 INSTRUCCIONES ESPECIALES PARA UNIX PCs.

El saber que instrucciones o funciones son especificas de las UNIX PCs son necesarias en el momento en que se quisiera correr el programa en otro tipo de maquina.

El programa se trato de hacer lo mas independiente posible de UNIX PCs, sin embargo debido a las grandes facilidades que presentan sus rutinas para el manejo de ventanas, se hizo uso de ellas.

Estas rutinas de ventanas son exclusivas de UNIX PCs, aunque con el aumento de popularidad de interfaces gráficos, otros UNIX deben de contar con funciones equivalentes, que serían las que se deberían de modificar.

En lo general se utilizaron funciones estandard de UNIX V de AT&T y bibliotecas de "C". Por lo que se podría decir que el programa es bastante portable.

Los archivos de encabezado o inclusión, en los que se encuentrán las definiciones de prototipos de funciones y biblioteca, utilizadas en el programa son las siguientes:

<fcntl.h> Constantes para el manejo de Archivos.
<errno.h> Funciones de error.
<termio.h> Para el manejo de E/S de la Terminal.
<stdio.h> Funciones estandard de E/S.

<window.h>	Funciones para el manejo de ventanas.
<types.h>	Contiene definiciones de variables usadas por "C".
<stat.h>	Para el manejo de las características de archivos.
<string.h>	Funciones para el manejo de cadenas de caractores.
<rec.h>	Definiciones de Constantes del programa del manejo de red.

Así de esta manera si se cuentan con estos archivos de inclusión, se podría modificar el programa y correr en otras maquinas.

CAPITULO 5.

INTEGRACION DEL PROGRAMA, CONEXION FISICA E INTERFACE DE USUARIO.

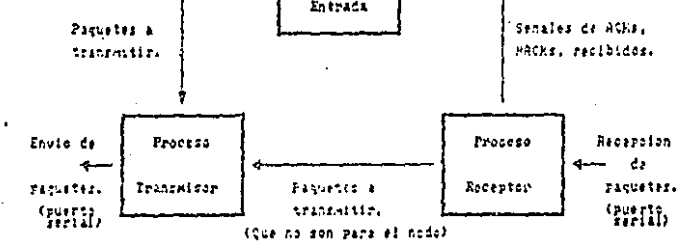
En este capitulo se vera como se integraron los diferentes procesos, dentro del mismo programa. Como se lleva a cabo la conexión fisica de los computadores, asi como la interface de usuario, que comandos existen, opciones etc.

Por ultimo se tendrá el listado de los archivos que contienen los diferentes programas y definiciones. Asi como sus nombres e interacción entre ellos.

5.1 INTEGRACION DEL PROGRAMA.

El programa de red consta de los tres procesos principales , Receptor, transmisor y Entrada, los cuales están interconectados mediante PIPEs. Los comandos son alimentados al proceso de Entrada, a través del teclado. Los paquetes de salida del nodo salen a partir del proceso transmisor y los paquetes que se reciben llegan a través del proceso receptor. Asi existe solo un lugar de entrada y de salida para cada tipo de datos.

A continuación se muestra un diagrama de la comunicación entre procesos y con el exterior del programa.

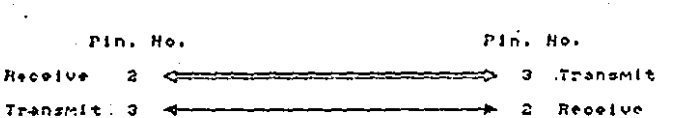


El control de cada proceso se lo da el S.O. ya que cada uno de ellos es independiente, sin embargo cada uno tiene la misma prioridad de ejecución. Aquí podría existir el problema de que se perdieran datos que están llegando, sin embargo el puerto serial cuenta con un "buffer" para almacenar pequeñas cantidades de datos. En lo que se refiere a datos escritos al teclado, la posibilidad es eliminada ya que el usuario al momento de teclear no es tan rápido como para que esto suceda.

5.2 CONEXION FISICA.

La conexión necesaria para lograr la comunicación de los computadores es muy sencilla, es una conexión tipo NULL MODEM, "que consiste en conectar el "pin" transmisor de un computador con el receptor del otro y la señal de tierra de los computadores"(18). Así se conectan todos los computadores que pertenezcan a la red. Si se desea una manera mas fácil de hacer dicho tipo de conexiones, se pueden realizar cajas de "conmutación" para hacer mas fácil la entrada y/o salida de nuevos computadores a la red.

Conexión de Null Modem para comunicaciones asincronas serialos, con un conector tipo DB-25.



"El tipo de interface es el estandard RS-232-C aprobado por la EIA (Electronic Industry Asociation). El cual se usa para transmisiones en banda base, en cortas distancias, de 1000 a 2000 pies, dependiendo de la velocidad y de las fuentes de ruido cercanas"(13).

Para una red grande o con necesidades de alta velocidad este tipo de conexión no es la mas adecuada, sin embargo para propósitos de pruebas, o transmisiones pequeñas es suficiente.

(18)(13) Lano, G.H. p. 83, p.82

5.3 INTERFACE DE USUARIO.

En esta sección se explicara la interface de usuario y sus principales caracteristicas.

Hay que recordar que esta interface de usuario no forma parte de la capa de encadenamiento de datos (DATA LINK), sino que se podria considerar como un ejemplo muy sencillo de una capa superior en el modelo OSI, tal como la capa de presentación (PRESENTATION LAYER). Lo cual permite ver que el programa puede hacer una interface con diferentes capas.

Aunque esta interface no forme parte de la capa de encadenamiento, es necesaria para tener alguna manera de comunicación con el programa controlador de red. Ademas de que provee una manera sencilla de usar para la transmisión de mensajes y archivos, asi como una herramienta indispensable durante la etapa de desarrollo y depuración de la aplicación. Para la depuración cuenta con algunas facilidades, ademas de las evidentes, poder visualizar los mensajes que llegan y salen del nodo, poder desplegar las variables, así como el poder enviar mensajes especiales con errores, para probar condiciones especiales y de error.

5.3.1 VENTANAS.

El programa cuenta con dos ventanas, una para entrada y otra de salida. Al inicio estas cubren toda la pantalla pero pueden ser modificadas tanto en tamaño como en su posición dentro de la pantalla. La ventana de entrada acepta los comandos y mensajes del usuario, ademas de desplegar mensajes de error del proceso de Entrada. La ventana de salida despliega mensajes que el programa de red manda al usuario, estos mensajes son principalmente informativos, explican lo que esta pasando dentro del programa en el nodo.

5.3.2 COMANDOS VALIDOS.

El programa acepta tres comandos validos, uno para mandar archivos, otro para mandar mensajes y uno ultimo para terminar el programa. Estos tienen diferentes variantes a continuación se muestran junto con su sintaxis.

send X Y [-s]|[-e] Mensaje
- Para mandar mensajes hasta de 250 caracteres.
- X Y : Indican la dirección a la cual va dirigido el mensaje.
  X Grupo.
  Y Nodo.
  Pueden tomar los valores de 0 a 255.
  Siendo el 0 0 un "Broadcast" para toda la red.
  El X 0 es un "Broadcast" para el grupo X.

sendf X Y NombreDeArchivo
- Para mandar un archivo al nodo X Y.
- X Y : Tienen las mismas caracteristicas que para mensajes. Solo que NO se permiten hacer "Broadcasts".

down
- Es el comando para terminar el programa.

- El comando para enviar mensajes, tiene dos opciones diferentes, que permite realizar pruebas. Estas opciones son:

[-e] Error en el CRC el cual añade un CRC invalido al paquete a ser enviado, lo cual es equivalente a que el paquete hubiera tenido problemas en los datos durante la transmisión.

[-s] Error en el nodo transmisor. Le pone un valor inexistente a la dirección del fuente. Para semejar que hubo un problema de un nodo que se desconecto.

5.3.3 MENSAJES DEL PROGRAMA DE RED.

EL programa de red manda mensajes constantemente al usuario, indicando que es lo que esta pasando en ese momento. Existen tanto mensajes informativos como de error.

Dentro de los mensajes de error dice si un archivo no se pudo abrir, si llego un paquete fuera de orden, si hubo un error de sintaxis, si hubo errores en la transmisión, etc. En lo que se refiere a mensajes informativos dice al usuario si se esta retransmitiendo un paquete, si llego un ACK, NACK o un mensaje diciendo cual es su procedencia, etc.

5.4 CODIGO DEL PROGRAMA DE RED.

El programa fuente consta de 4 archivos:

Entrada.C	Contiene el programa principal así como las funciónes del proceso de Entrada.
RecTran.C	Contiene el programa de Receptor así como el Transmisor.
DefRed.H	Contiene las definiciones de las constantes usadas por todos los procesos.
Makefile	Es el Archivo para ucar la utileria Make de UNIX que guarda las relaciones entre los archivos. Esta utileria decide que modulos hay que compilar y que otros no.

El listado de dichos archivos se muestra en el apéndice A.

CONCLUSIONES Y RECOMENDACIONES.

En General se puede decir que el programa manejador de red cumplió con su objetivo, ya que permite la comunicación confiable y sin errores entre varios computadores. Tiene una interface de usuario sencilla de usar, ya que trabaja con ventanas, la cual permite visualizar lo que pasa con las comunicaciones que pasan dentro del computador.

Asimismo se diseño con una filosofía modular, la cual le permite se adecuase para otras aplicaciones, o bien ser utilizado como base de un nuevo programa.

Desde otro punto de vista, el programa tiene limitaciones, como lo son la velocidad, ya que trabaja únicamente con la velocidad del puerto serial y algunas de las subrutinas se podrían llevar a cabo mediante equipo electrónico. Esto es el costo de no usar equipo electronico adicional al ya incluido en el computador.

También sus facilidades son limitadas, sin embargo son una base solida sobre la cual se podrían construir otras mas sofisticadas. Es una base solida ya que provee gran cantidad de casos dentro de la comunicación de datos y casos especiales, los cuales fueron pensados desde el diseño.

Para llegar a las conclusiones anteriores, se realizaron diferentes pruebas de las cuales se puede resumir lo siguiente:

- Los errores de transmisión se detectaron todas las veces que se generaron. Tanto errores en los datos del paquete, mediante el CRC-16, como errores en la secuencia de los paquetes (perdida de orden o de un paquete especifico), lo cual demostró que la transmisión era confiable.

- El proceso de transmisión no sufrio detrimento en velocidad, a medida que se conectaron mas maquinas, hasta 4. Lo anterior no se puede generalizar para un mayor número de maquinas, ya que la velocidad limitada del puerto serial significaría un cuello de botella, aunque el anillo de inserción tiende a minimizar esto. Lo que significa que si se tuviera equipo electrónico especializado mas veloz, para la transmisión, entonces el aumento de nodos en la red afectaría aún menos en la velocidad.

- La velocidad y la distancia entre nodos son inversamente proporcionales, ya que a medida que aumenta la velocidad la distancia a la cual las transmisiones pueden ser confiables se reduce. Estos dos factores se ven afectados también por las fuentes de ruido externas que existan. Así los máximos nominales de velocidad y distancia son 9.6 Kbps a 1000 pies de distancia. Sin embargo la velocidad efectiva de datos se puede reducir hasta en un 40% por el tiempo necesario para el control de los paquetes, empaquetamiento, retransmisiones, fuentes de ruido, etc. Por ello se recomienda que si existen fuentes de ruido cercanas, o las distancias son mayores, el bajar la velocidad del puerto serial, para tener menos retransmisiones y mayor salida efectiva.

RECOMENDACIONES.

A continuación se describen algunas recomendaciones para la modificación y aprovechamiento del programa en otras aplicaciones, así como mejoras del mismo.

- Para utilizar el programa como una base (DATA LINK LAYER) para otras aplicaciones de comunicaciones. La modificación principal sería en lo que respecta a la comunicación entre capas (ISO) ya que en lugar de escribir a la capa de presentación (ventanas en este caso) se debería tener comunicación con otra capa superior mediante los mensajes o códigos que esta entendiera. Ej. en vez de desplegar mensajes informativos para el usuario, se deberían mandar códigos de retorno indicando errores, éxitos o el estado de una transmisión para que la capa superior pudiera tomar las acciones necesarias.

- Para mejorar la velocidad se debería tener un equipo electrónico especializado para la transmisión. El cual eliminaría las restricciones impuestas por el puerto asincrono serial. Además se podría implementar en ese equipo la detección de errores, así como alguna lógica que permitiera reconocer cuales paquetes deben ser retransmitidos (por que no son para el nodo en cuestión) y cuales son para el nodo, sin que el "software" necesite decidirlo, así como la facilidad de sacar los paquetes que tengan una cuenta mayor que la permitida. Esto daría un programa controlador de red Híbrido ya que tendría funciones tanto en "hardware" como en "software".

- Si se quisiera modificar el programa para que corriera en un S.O. que no fuera multitarea, se tienen que hacer gran cantidad de cambios en la implementación, sin embargo el protocolo sigue siendo valido. Algunos de los puntos a considerar serían: A) El uso de interrupciones (para evitar la perdida de información cuando esta se recibiera). B) El uso de colas circulares para almacenamiento de la información que va a ser transmitida, recibida y procesada. C) El remplazo de funciones especificas de UNIX por otras validas en el S.O. en el que se va a implementar el programa. D) Un cuidadoso rediseño del orden en que se llevan a cabo las diferentes actividades del programa. E) Otras consideraciones que imponga el S.O. al que se va a migrar.

- La opción de utilizar el programa en otra topología diferente al anillo de inserción no esta contemplada, ya que el diseño se llevo a cabo basandose en una filosofía para dicha topología.

- En el caso de querer experimentar con otro método para la detección de errores, las modificaciones dependen del método que se desee implementar. Ya que si no necesita de caracteres adicionales al final de los paquetes las modificaciones irían hasta la definición del protocolo. Sin embargo con un análisis cuidadoso del programa así como del método deseado, se podría implementar ya que el programa es bastante modular.

- Si se deseara utilizar este proyecto para la enseñanza de teoría de redes, se debería llevar a cabo a través de varios pasos que se vayan incrementando en dificultad:

i) Utilización del modo "Raw".
ii) Transmisión básica de mensajes de un computador a otro.
iii) Uso de ACKs, NACKs y CRC-15.
iv) Transmisión de archivos pequeños y retransmisiones.
v) Empaquetamiento para la transmisión de archivos.

Esta última información se basó en CS457 DATA COMMUNICATIONS Course Description de Erik Wettersten, Colorado State University, Fort Collins Co. E.E.U.U. Otoño 1988.

APENDICE A.

LISTADO DE LOS ARCHIVOS DEL PROGRAMA CONTROLADOR DE RED.

ARCHIVO DEL RECEPTOR Y TRANSMISOR (RecTran.C)

```
/* PROGRAMA : El programa controlador de red esta compuesto de tres
** archivos fuentes principales:  Entrada.C, RecTran.C, DefRed.H.
**
** PROPOSITO: Transferir datos de una PC a cualquier otra PC conecta-
** da a la red. El programa puede transmitir tanto mensajes como ar-
** chivos, los últimos divididos en varios paquetes.
**
** En el caso de archivos, el paquete num0 contiene únicamente el
** nombre del archivo, y los paquetes subsecuentes contienen los dat-
** os del archivo. El último paquete es de tipo EOF, para indicarle
** al receptor que el archivo se ha recibido completamente.
** El programa realiza retransmisiones de paquetes, solo cuando el
** paquete fue recibido con errores o no se ha recibido un ACK de
** dicho paquete. También existen retransmisiones cuando ocurre un
** timeout.
** De la misma manera existe un timeout del receptor, ya que no exis-
** te manera de que el receptor sepa que el transmisor a terminado la
** transmisión debido a demasiados errores.
** Para mensajes se puede realizar un Broadcast o transmisión a todas
** las PCs de la red o bien a todas las PCs de un grupo.
**
** Un ACK se envía cuando un paquete fue recibido correctamente, y un
** NACK se transmite en el caso contrario.
** Cada paquete lleva una cuenta de las veces que ha pasado por algún
** nodo. De manera que cada PC revisa ese numero y si es mayor que un
** limite, El paquete es sacado de la red.
**
** Cuando un archivo se empieza a transmitir, se guarda información
** acerca de el tiempo y el numero de retransmisiones , para mantener
** un control de los paquetes que llegaron correctamente, y cuales se
** necesitan retransmitir.
**
** De la misma manera cuando un archivo se recibe por primera vez, se
** abre una entrada en el receptor, para que este pueda determinar
** cual es el siguiente paquete a recibir, timeouts y a que archivo
** hay que escribir el paquete.
**
** UTILIZA: fcntl.h, errno.h, termio.h, stdio.h, sys/window.h,
**          sys/types.h, sys/stat.h y string.h;
** FUNCIONES: main, initially, xmit, recv, , echo_msg, fill_inbuf,
** send_pkt, open_window, conv_char,KB, check_KB, checksyntax,
** flush_outbuf, error_display, syntax_check, get_PC_id, init_queue,
** st_queue, rt_queue, setport, kill_close_all, calc_crc, conv_mid.
**
*/
#include <fcntl.h>
#include <errno.h>
#include <termio.h>
#include <stdio.h>
#include <sys/window.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <string.h>
#include </usr/defred.h>
```

```
main()
/*
** PROPOSITO: Abre el puerto remoto, pide la identificación de la PC
** el limite de saltos.
** Abre dos ventanas, crea dos pipes, los procesos receptor y trans-
** misor.
** Lee del teclado y pasa los datos leidos al proceso transmisor.
** Cuando termina cierra los puertos, ventanas, y termina los proce-
** sos hijos.
**
** UTILIZA: open, pipe, fork y exit;
** LLAMA: initially, recv, xmit, KB y kill_close_all.
** LLAMADO POR: ninguno, es la función principal.
*/
{
   int remote, recv_xmit[2], KB_xmit[2], pid_port[2], recv_KB[2];
   int PC_id[3], window[2];

   /* Abre el puerto remoto (serial RS-232). guarda el file
      descriptor */

   if ((remote=open("/dev/tty000", O_RDWR | O_NDELAY)) < 0){
      fprintf(stderr, "error al abrir el puerto remoto  %d error:
                       %d\n", errno);
      sleep(10);
      exit(1);
   }
   initially(remote, PC_id, window);  /* inicializa variables y
                                         ambiente */
   pipe(KB_xmit);                     /* abre pipes para comunicación
                                         inter-procesos */
   pipe(recv_xmit);
   pipe(recv_KB);

   /* crea el proceso transmisor. */

   if (!(pid_port[0]=fork())){
      xmit(KB_xmit[0], recv_xmit[0], remote, window[0]);
      exit();
   }
   /* crea el proceso receptor.  */

   if (!(pid_port[1]=fork())){
      recv(recv_xmit[1], remote, PC_id, window[1],recv_KB[1]);
      exit();
   }
   /* Llama al proceso del teclado, cuando termina, acaba los
       procesos hijos y limpia el ambiente, cerrando los Fd */

   KB(KB_xmit[1], recv_KB[0], PC_id, window[0]);
   kill_close_all(remote, recv_xmit, KB_xmit,recv_KB,pid_port,
                  window);
}
```

```
initially(remote, PC_id, window)
int remote, PC_id[3], window[2];
/*
**
** PROPOSITO: Obtiene la dirección de la PC's y el limite de saltos,
** pone el puerto en modo "Raw", abre dos ventanas, una para el
** teclado (o Entrada) y otra para el proceso receptor.
** LLAMA: get_PC_id, setport, open_window.
** LLAMADO POR: main.
**
*/
{
   int window_size[4];

   cr[0]=NL;
   cr[1]=CR;
   get_PC_id(PC_id);
   setPort(remote);
   window_size[0]=0;
   window_size[1]=170;
   window_size[2]=720;
   window_size[3]=90;
   window[1]=open_window(window_size);
   window_size[0]=0;
   window_size[1]=0;
   window_size[2]=720;
   window_size[3]=130;
   window[0]=open_window(window_size);
}

xmit(KB_xmit, recv_xmit, remote, window)
int KB_xmit, recv_xmit, remote, window;
/*
** PROPOSITO: Se mantiene leyendo de los pipes para transmitir ya sea
** paquetes de esta PC hacia otra o bien paquetes transmitidos por
** otra PC (retransmisión).
**
** Lleva a cabo el enmascaramiento o byte stuffing de caracteres.
** Lee paquetes del teclado y del proceso receptor, para escribirlos
** al puerto remoto.
**
** UTILIZA: fstat.
** LLAMA: ninguno otro.
** LLAMADO POR: main.
**
*/
```

```
{
   struct stat buf;
   unsigned char bufchar;
   int end, i;

   end=0;
   while (!end){
      fstat(KB_xmit, &buf);    /* Revisa si hay paquetes por mandar */
      if (buf.st_size >= HEADER_SIZE){
         for (i=0; i<HEADER_SIZE; i++){ /* transmite el encabezado */
            read(KB_xmit, &bufchar, 1);
            write(remote, &bufchar, 1);
         }
         read(KB_xmit, &bufchar, 1);
         while (bufchar != ETB){          /* transmite el mensaje, CRC
                                             y el ETB */
            write(remote, &bufchar, 1);
            if (bufchar == DLE){          /* hace el byte stuffing */
               read(KB_xmit, &bufchar, 1);
               write(remote, &bufchar, 1);
            }
            read(KB_xmit, &bufchar, 1);
         }
         write(remote, &bufchar, 1);
      }
      fstat(recv_xmit, &buf);     /* Revisa si hay un paquete para
                                     retransmisión. */
      if (buf.st_size >= HEADER_SIZE){
         for (i=0; i<HEADER_SIZE; i++){ /* transmite el encabezado */
            read(recv_xmit, &bufchar, 1);
            write(remote, &bufchar, 1);
         }
         read(recv_xmit, &bufchar, 1);
         while (bufchar != ETB){ /* transmite el mensaje, CRC y ETB*/
            write(remote, &bufchar, 1);
            if (bufchar == DLE){          /* hace el byte stuffing */
               read(recv_xmit, &bufchar, 1);
               write(remote, &bufchar, 1);
            }
            read(recv_xmit, &bufchar, 1);
         }
         write(remote, &bufchar, 1);
      }
   }
}


recv(recv_xmit, remote, PC_id, window, recv_KB)
int recv_xmit, remote, PC_id[3], window, recv_KB;
/*
**
** PROPOSITO: Lee paquetes del puerto. Lee el encabezado y decide si
** es un mensaje o un paquete de archivo. Entonces decide que función
** hay que llamar, ya sea la que procesa archivos o mensajes.
** También revisa la tabla de recepción por si hay un timeout.
** LLAMA: recv_msg, recv_file, check_t_c.
** LLAMADO POR: main.
**
*/
{
   unsigned char header[HEADER_SIZE];
   int      end,i,finish;
```

```
    end=0;
    for(i=0;i<RC_TBL_SIZE;i++)
       r_tbl[i].free=1;
    while (!end){
       finish=0;
       while(!finish){
          if((read(remote, &header[0],1))>0){
             finish=1;
          }
          check_t_o(window);
       }
       for (i=1; i<HEADER_SIZE; i++){
          finish=0;
          while(!finish){
             if((read(remote, &header[i],1))>0){
                finish=1;
             }
             check_t_o(window);
          }
       }
       if((header[12] == MSG) || (header[12] == ACK) ||
          (header[12] == NNACK)){
          recv_msg(recv_xmit,remote,PC_id,window,header);
       }
       else{
          recv_file(recv_xmit,remote,PC_id,window,header,recv_KB);
       }
    }
}

recv_msg(recv_xmit,remote,PC_id,window,header)
int recv_xmit,remote,PC_id[3],window;
unsigned char header[HEADER_SIZE];
/*
**
** PROPOSITO: Procesa los mensajes recibidos, revisa para que nodo es
** el paquete. Si es para 'este' nodo lo despliega en la pantalla, en
** caso contrario o es retransmitido a la red. Existen otros casos
** especiales como:
** Si el paquete se recibio con errores se manda un NACK, en caso
** contrario envia un ACK. Se revisa la cuenta, si es mayor que el
** limite, entonces hay que remover el paquete de la red.
** También se maneja el Broadcast de grupo y general.
** LLAMA: fill_inbuf, calc_crc, conv_mid, echo_msg, send_pkt,
** disp_ACK_NACK, disp_spec, re_xmitt.
** LLAMADO POR: recv.
*/
{
    unsigned char bufchar;
    char str[80];
    int i, x, echo0, echo1, xmit;
    unsigned int crc;

    str[0]=NULL;
    echo0=0;
    echo1=0;
    xmit=0;

    if (header[0] == PC_id[0] && header[1] == PC_id[1]){
       /* El paquete es para 'esta' PC. */
       echo0=1;
    }
    else{
```

```
    if ((header[0] == PC_id[0] || header[0]== 0) && header[1] == 0){
      echo0=1;          /* paquete para 'esta' PC y Broadcast */
      xmit=1;
    }
    else{
       if (header[2] == PC_id[0] && header[3] == PC_id[1]){
          /* La PC no estaba conectada. */
          strcpy(str, "no conectada, paquete descartado No: \0");
          echo1=1;
       }
       else{      /* El paquete no es para esta PC, retransmitirlo */
         if (header[13] < PC_id[2])     /* Revisar cuenta */
           xmit=1;
         else{
           strcpy(str, "cuenta mayor que el limite, mensaje
                        descartado No: \0");
           echo1=1;
         }
       }
    }
    if (echo0 && xmit){
       /* paquete de broadcast, pero esta PC lo envió, entonces no
          retransmitir */
       if (header[2] == PC_id[0] && header[3] == PC_id[1])
         xmit=0;
       if (header[13] >= PC_id[2])
         xmit=0;
    }
}
if (echo0 && xmit){
   /* para esta PC y broadcast. transmitir el encabezado */
   header[13]++;
   for (i=0; i<HEADER_SIZE; i++){
      write(recv_xmit, &header[i], 1);
   }
}
if (echo0 || echo1){                    /* Llenar in_buffer */
   fill_inbuf(xmit, recv_xmit, remote, window);
}
if (echo0){                         /* paquete para esta PC */
   init_queue(&aux_buf);
   for(i=0;i<(HEADER_SIZE-1);i++){
     st_queue(&aux_buf,header[i],BUF_SIZE);
   }
   bufchar=0;
   /* cuenta = 0 para calcular el CRC del paquete entero. */
   st_queue(&aux_buf,bufchar,BUF_SIZE);
   while(rt_queue(&in_buf,&bufchar)!=-1){
     st_queue(&aux_buf,bufchar,BUF_SIZE);
   }
   crc=calc_crc(aux_buf.q,aux_buf.st);
   if (crc == 0 && header[13] < PC_id[2]){
      /* recepción correcta y cuenta dentro del limite */
      echo_msg(window, header);
      if (header[12] == MACK){
         strcpy(str, "ACK para el mensaje No = \0");
         disp_ACK_NACK(str,&header[4],window);
         write(window,&cr[0],2);
      }
      else{
         if (header[12] == MNACK){
            strcpy(str, "NACK para el mensaje No = \0");
            disp_ACK_NACK(str,&header[4],window);
            write(window,&cr[0],2);
         }
      }
```

```
            if (header[12] == MSG){    /* paquete tipo mensaje normal */
               in_buf.rt=0;
               in_buf.st=2;
               while (rt_queue(&in_buf, &bufchar) != -1)
                  write(window, &bufchar, 1);
               if (!(header[2]==PC_id[0] && header[3]==PC_id[1]))
                  send_pkt(header, recv_xmit, window, PC_id, MACK);
            }
         }
         if (crc != 0 && header[13] < PC_id[2]){
            /* mensaje con errores, CRC equivocado */
            send_pkt(header, recv_xmit, window, PC_id, MNACK);
            echo_msg(window, header);
            strcpy(str, " mensaje No = \0");
            disp_ACK_NACK(str,&header[4],window);
            strcpy(str, "recibido con errores.\n\r\0");
            write(window, str, strlen(str));
         }
         if (crc == 0 && header[13] >= PC_id[2]){
            strcpy(str, "cuenta > limite, mensaje descartado No: \0");
            echo1=1;
         }
         echo0=0;
         xmit=0;
      }
      if (echo1){                          /* casos especiales */
         echo_msg(window, header);
         disp_spec(str,window);
         echo1=0;
      }
      if (xmit){
         /* Solo cuando se retransmite, no en el caso del broadcast */
         re_xmit(header,recv_xmit,remote);
         xmit=0;
      }
   }


recv_file(recv_xmit,remote,PC_id,window,header,recv_KB)
int recv_xmit,remote,PC_id[3],window;
unsigned char header[HEADER_SIZE];
int recv_KB;
/*
** PROPOSITO: Procesa los paquetes tipo archivo recibidos, revisa
** para que nodo es el paquete. Si es para 'este' nodo lo escribe al ** archivo
correspondiente, en caso contrario o es retransmitido a la
** red. Existen otros casos especiales como:
** Si el paquete se recibio con errores se manda un NACK, en caso
** contrario envia un ACK. Se revisa la cuenta, si es mayor que el
** limite, entonces hay que remover el paquete de la red.
** Ademas de enviar ACKs y NACKs a la red envia ACKs al proceso de
** teclado, para que pueda liberar y actualizar la tabla de ACKs.
** No existe la opción de Broadcast de archivos así que no se
** considera, en el código.
** Los archivos se transmiten en paquetes, el paquete numero cero
** siempre contiene el nombre de archivo. Una vez que se ha recibido
** dicho paquete un lugar en la tabla de ACKs se abre para llevar el
** control de los paquetes que llegan, timeouts y cual es el siguien-
** te paquete ha ser escrito. Los paquetes deben llegar en orden, en
** caso contrario se manda un NACK y se descarta el paquete.
** Todos los paquetes de un archivo contienen el mismo IdM identifi-
** cador de mensaje. El último paquete es diferente en el sentido de
** que tiene el tipo LAST.
** Se revisa si existen timeouts, ya que si la transmisión falla no
** existe manera, para el receptor, de saber que el transmisor a de-
** jado de transmitir.
```

```
** Cuando un archivo se recibe sin errores entonces se escribe a
** disco. si llego con errores se envia un NACK.
** Al recibir un ACK o un NACK, se envia un mensaje al proceso de
** entrada, para que actualize la tabla de ACKs.
** LLAMA: fill_inbuf, calc_crc, conv_mid, echo_msg, send_pkt,
** disp_ACK_NACK, disp_spac, re_xmitt.
** LLAMADO POR: recv.
*/
{
   unsigned char ch,bufchar, fname[FNAME_SIZE], buf[FNAME_SIZE], c;
   char str[80];
   int count,i, x, echo0, echo1, xmit, fd, free, ind;
   unsigned int crc, pkt_no;

   fname[0]=NULL;
   str[0]=NULL;
   echo0=0;
   echo1=0;
   xmit=0;

   if (header[0] == PC_id[0] && header[1] == PC_id[1]){
      /* El paquete es para 'esta' PC */
      echo0=1;
   }
   else{
       if (header[2] == PC_id[0] && header[3] == PC_id[1]){
          /* LA PC no esta conectada */
          strcpy(str, "no esta conectada, \0");
          echo1=1;
       }
       else{ /* paquete no es para esta PC, entonces retransmitir */
          if (header[13] < PC_id[2])    /* Revisar cuenta */
             xmit=1;
          else{
             strcpy(str,"cuenta mayor que el limite, \0");
             echo1=1;
          }
       }
   }

   if (echo0 || echo1){   /* Llenar el in_buffer */
      /* Lee el nombre de archivo del puerto, quitando DLEs. */
      pkt_no=(unsigned int) W_to_I(&header[8]);
      if(header[12]==FILE && pkt_no==0){
         count=0;
         while(!(read(remote, &fname[count],1)));
         while (fname[count] != NULL){
            /* lee nombre de archivo quitando DLEs */
            count++;
            if (fname[count-1] == DLE){ /* quita DLEs */
               while(!(read(remote, &fname[count-1],1)));
            }
            while(!(read(remote, &fname[count],1)));
         }
      }
      fill_inbuf(0, recv_xmit, remote, window);
   }
   if (echo0){                              /* paquete para esta PC */
      /* guarda el encabezado en aux_buff */
      init_queue(&aux_buf);
      for(i=0;i<(HEADER_SIZE-1);i++){
         st_queue(&aux_buf,header[i],BUF_SIZE);
      }
      bufchar=0;  /* Cuenta = 0 para calcular CRC del paquete */
      st_queue(&aux_buf,bufchar,BUF_SIZE);
```

```
/* Si el paquete es de un archivo, guardar el nombre de archivo
   en aux_buf */
if(header[12]==FILE && pkt_no==0) {
   for(i=0;i<=count;i++)
      st_queue(&aux_buf,fname[i],BUF_SIZE);
}
/* guarda los datos en aux_buf */
while(rt_queue(&in_buf,&bufchar)!=-1){
  st_queue(&aux_buf,bufchar,BUF_SIZE);
}
crc=calc_crc(aux_buf.q,aux_buf.st);
/* calcula el CRC del paquete */
if (crc == 0){                          /* recibido sin errores. */
   echo_msg(window, header);
   if (header[12] == ACKF){
      strcpy(str, "ACK de Archivo, paquete No = \0");
      disp_pkt(str,header,window);
      write(recv_KB,&header[4],9);
      /* envia ACK al proceso de entrada. */
   }
   else{
      if (header[12] == NACKF){
         strcpy(str, "NACK de Archivo, paquete No = \0");
         disp_pkt(str,header,window);
         write(recv_KB,&header[4],9);
         /* envia NACK al proceso de entrada. */
      }
   }
   if (header[12] == FILE || header[12] == LAST){
      /* Paquete tipo normal. */
      /* restaura los apuntadores de in_buff para utilizar */
      /* la información de nuevo. */
      in_buf.rt=0;
      in_buf.st=2;
      if(header[2]==PC_id[0] && header[3]==PC_id[1]){
         strcpy(str," paquete enviado por nosotros, no sobre-
                    escribir. No = \0");
         disp_pkt(str,header,window);
         if(pkt_no==0)
           write(recv_KB,&header[4],9);
           /* envia mensaje tipo FILE al proceso de Entrada */
      }
      else{
         if(pkt_no == 0){
            /* abre el archivo y escribe datos. */
            free=find_space();
            if(free>=0 && (look_entry(header)<0)){
               strcpy(buf,fname);
               strcpy(str,"Archivo >> \0");
               write(window,str,strlen(str));
               write(window,buf,strlen(buf));
               if(open_file(&fd,buf)<0){
                strcpy(str,"error, al abrir archivo. No = \0");
               }
               else{
                open_tbl_entry(header,fd,free);
                send_pkt(header,recv_xmit,window,PC_id,ACKF);
                strcpy(str," << abierto No = \0");
               }
            }
            else{
               if(free<0)
                 strcpy(str, "Tabla de Recepción llena,
                              archivo no abierto, No = \0");
               else
```

```
                strcpy(str,"Archivo abierto anteriormente.
                            No = \0");
            }
            disp_pkt(str,header,window);
        }
        else{
            ind=look_entry(header);
            if(ind<0){
              strcpy(str," Entrada de Archivo no encontrada en
                            tabla, No = \0");
              disp_pkt(str,header,window);
            }
            else{
              if(pkt_no==r_tbl[ind].next){
                  send_pkt(header, recv_xmit, window, PC_id,
                            ACKF);
                  while (rt_queue(&in_buf, &bufchar) != -1){
                    write(r_tbl[ind].fd, &bufchar, 1);
                  }
                  r_tbl[ind].next++;
                  time(&r_tbl[ind].recv_time);
                  strcpy(str,"paquete escrito al archivo, No
                            = \0");
                  disp_pkt(str,header,window);
                  if(header[12]==LAST){
                    r_tbl[ind].free=1;
                    if(close(r_tbl[ind].fd)>=0){
                      strcpy(str," archivo cerrado, \0");
                      write(window,str,strlen(str));
                    }
                    else{
                      strcpy(str," error al cerrar archivo
                                  \0");
                      write(window,str,strlen(str));
                    }
                    strcpy(str," archivo recibido completo, No
                                = \0");
                    disp_pkt(str,header,window);
                  }
              }
              else{
                if(pkt_no < r_tbl[ind].next){
                  strcpy(str,"No de paquete menor que el
                              esperado, No = \0");
                  disp_pkt(str,header,window);
                }
                else{
                  strcpy(str," paquete fuera de orden, No =
                              \0");
                  disp_pkt(str,header,window);
                  header[11]=(unsigned char)(r_tbl[ind].next);
                  header[10]=(unsigned char)
                              (r_tbl[ind].next >> 8);
                  header[9] =(unsigned char)
                              (r_tbl[ind].next >> 16);
                  header[8] =(unsigned char)
                              (r_tbl[ind].next >> 24);
                  send_pkt(header, recv_xmit, window, PC_id,
                            NACKF);
                }
                time(&r_tbl[ind].recv_time);
              }
            }
        }
    }
}
```

```
        }
        if (crc != 0){          /* Paquete recibido con errores */
                                /* envia un NACK */
           /* actualiza el tiempo de recepción en la tabla.*/
           echo_msg(window, header);
           strcpy(str, " paquete recibido con errores No = \0");
           disp_pkt(str,header,window);
           ind=look_entry(header);
           if(ind>=0){
             time(&r_tbl[ind].recv_time);
             send_pkt(header, recv_xmit, window, PC_id, NACKF);
           }
        }
        echo0=0;
        xmit=0;
     }
     if (echo1){                /* casos especiales */
       /* despliega mensajes para los casos de cuenta y no conectado.*/
        echo_msg(window, header);
        write(window,str,strlen(str));
        strcpy(str," paquete descartado No = \0");
        disp_pkt(str,header,window);
        if(header[2]==PC_id[0] && header[3]==PC_id[1] &&
           (header[12]==FILE || header[12]==LAST   )){
           ch=FILE;
           write(recv_KB,&header[4],8);
           write(recv_KB,&ch,1);
        }
        echo1=0;
     }
     if (xmit){  /* Solo se ejecuta durante la retransmisión. */
        re_xmitt(header,recv_xmit,remoto);
        xmit=0;
     }
}


disp_pkt(str,header,window)
char str[80];
unsigned char header[HEADER_SIZE];
int  window;
/*
**
** PROPOSITO : Despliega un mensaje, indicándole el mensaje y el
** numero de paquete.
** LLAMADO POR : recv_file.
**
*/
{  disp_ACK_NACK(str,&header[4],window);
   strcpy(str,  paquete No = \0");
   disp_ACK_NACK(str,&header[8],window);
   write(window,&cr[0],2);
}

find_space()
/*
**
** PROPOSITO : Encuentra un lugar libre en la tabla de recepción.
** Regresa la posición de dicho lugar.
** LLAMADO POR : recv_file;
**
**
*/
{   int i;
```

```
    for(i=0;i<RC_TBL_SIZE;i++){
       if(r_tbl[i].free==1)
          return (i);
    }
    return (-1);
}


open_tbl_entry(header,fd,ind)
unsigned char header[HEADER_SIZE];
int      fd;
int    ind;
/*
**
** PROPOSITO : Abre una entrada en la tabla de recepción, para llevar
** el control de los diferentes archivos que se están recibiendo.
** LLAMADO POR : recv_file.
**
*/
{    int j;

     r_tbl[ind].free=0;
     for(j=0;j<HEADER_SIZE;j++)
        r_tbl[ind].header[j]=header[j];
     r_tbl[ind].fd=fd;
     r_tbl[ind].next=1;
     time(&r_tbl[ind].recv_time);
}

check_t_o(windond].header[j]=header[j];
     r_tbl[ind].fd=fd;
     r_tbl[ind].next=1;
     time(&r_tbl[ind].recv_time);
}


check_t_o(window)
int   window;
/*
**
** PROPOSITO : Revisa la tabla de recepción para ver si hay timeouts.
** Si existe alguno entonces libera la entrada ocupada por el.
** LLAMADO POR : recv_file.
**
*/
{    unsigned long int new_time;
     char     str[80];
     int      i;

     time(&new_time);
     for(i=0;i<RC_TBL_SIZE;i++){
         if(((new_time-r_tbl[i].recv_time)>=RC_MAX_TIME)
                  &&(r_tbl[i].free==0)){
             strcpy(str," timeout del receptor para el No = \0");
             disp_ACK_NACK(str,&r_tbl[i].header[4],window);
             write(window,cr,2);
             r_tbl[i].free=1;
             close(r_tbl[i].fd);
         }
     }
}

lock_entry(header)
unsigned char header[HEADER_SIZE];
/*
**
```

```
** PROPOSITO : Busca una entrada en la tabla de recepción. Regresa
** el indice del arreglo.
** LLAMADO POR : recv_file.
**
*/
{ unsigned int mid_p, mid_t,i;

  mid_p= (unsigned int) W_to_I(&header[4]);

  i=0;
  while(i<RC_TBL_SIZE){
   if(r_tbl[i].free==0){
    mid_t=(unsigned int) W_to_I(&r_tbl[i].header[4]);
    if(mid_p==mid_t && r_tbl[i].header[2]==header[2] &&
       r_tbl[i].header[3]==header[3]) return (i);
   }
   i++;
  }
  return (-1);
}

re_xmitt(header, recv_xmit, remote)
unsigned char    header[10];
int    recv_xmit, remote;
/*
**
** PROPOSITO : retransmite un paquete que no era para 'esta' PC.
** LLAMADO POR : recv_file, recv_msg.
**
*/
{
unsigned char    bufchar;
int    i;

    header[13]++;
    for (i=0; i<HEADER_SIZE; i++)
        write(recv_xmit, &header[i], 1);
    while(!(read(remote, &bufchar, 1)));
    while(bufchar != ETB){
        write(recv_xmit, &bufchar, 1);
        if (bufchar == DLE){
           while(!(read(remote, &bufchar, 1)));
           write(recv_xmit, &bufchar, 1);
        }
        while(!(read(remote, &bufchar, 1)));
    }
    write(recv_xmit, &bufchar, 1);
}


disp_spec(str,window)
char  str[80];
int   window;
/*
**
** PROPOSITO : utilizado para desplegar una condición especial para
** mensajes ademas de un mensaje.
** LLAMADO POR: recv_msg.
** LLAMA   : rt_queue.
**
*/{
unsigned char    bufchar;

    write(window, str, strlen(str));
    bufchar=NL;
    write(window, &bufchar, 1);
```

```
    bufchar=CR;
    write(window, &bufchar, 1);
    in_buf.rt=0;
    in_buf.st=2;
    while (rt_queue(&in_buf, &bufchar) != -1)
       write(window, &bufchar, 1);
}

disp_ACK_NACK(str,num,window)
char str[80];
unsigned char num[4];
int window;
/*
**
** PROPOSITO : despliega un mensaje en la ventana, ademas despliega
** el no de mensaje que se esta procesando.
** LLAMA    : conv_mid.
** LLAMADO POR : recv_msg, recv_file.
**
*/
{
int i,flag,mid;
unsigned char bufchar,auxbuf[10];

    write(window, str, strlen(str));
    mid= (unsigned int) W_to_I(num);
    conv_mid(mid,auxbuf);
    flag=0;
    for (i=0; i<10; i++){
        if (auxbuf[i]!='0' || flag){
           write(window, &auxbuf[i], 1);
           flag=1;
        }
    }
    if(!flag)
       write(window,&auxbuf[--i],1);
    bufchar=SP;
    write(window, &bufchar, 1);
}

echo_msg(window, header)
int window;
unsigned char header[HEADER_SIZE];
/*
**
** PROPOSITO: Despliega el destino y origen de un paquete.
** LLAMA: conv_char.
** LLAMADO POR: recv
**
*/
{
   int i, x, size,flag;
   char str[80];
   unsigned char auxbuf[3], bufchar;

   for (i=0; i<4; i++){
      conv_char(header[i], auxbuf);
      if (i == 0){
         strcpy(str, "Para \0");
         size=strlen(str);
         write(window, str, size);
      }
      if (i == 2){
         strcpy(str, "De \0");
```

```
            size=strlen(str);
            write(window, str, size);
         }
         flag=0;
         for (x=0; x<3; x++){
            if((auxbuf[x]!='0')||flag){
               write(window, &auxbuf[x],1);
               flag=1;
            }
         }
         if (!flag)
            write(window, &auxbuf[--x], 1);
         bufchar=SP;
         write(window, &bufchar, 1);
      }
   }

   fill_inbuf(xmit, recv_xmit, remote, window)
   int xmit, recv_xmit, remote, window;
   /*
   **
   ** PROPOSITO: Llena un buffer con el mensaje recibido, tomando en
   ** cuenta el byte stuffing, Transmite los datos cuando es la opción
   ** de broadcast.
   ** LLAMA:  init_queue, st_queue.
   ** LLAMADO POR: recv .
   **
   */
   {
      unsigned char bufchar;

      init_queue(&in_buf);
      while(!(read(remote, &bufchar, 1)));
      if (xmit){
         write(recv_xmit, &bufchar, 1);
      }
      while(bufchar != ETB){
         if (bufchar == DLE){
            while(!(read(remote, &bufchar, 1)));
            if (xmit){
               write(recv_xmit, &bufchar, 1);
            }
            st_queue(&in_buf, bufchar, BUF_SIZE);
         }
         else{
            st_queue(&in_buf, bufchar, BUF_SIZE);
         }
         while(!(read(remote, &bufchar, 1)));
         if (xmit){
            write(recv_xmit, &bufchar, 1);
         }
      }
   }

   send_pkt(header, recv_xmit, window, PC_id, type)
   unsigned char header[HEADER_SIZE];
   int recv_xmit, window, PC_id[3];
   unsigned char     type;
   /*
   **
   ** PROPOSITO: Envia un NACK o ACK a una PC.
   ** LLAMA: init_queue, st_queue, rt_queue.
   ** LLAMADO POR: recv.
   **
   */
   {
```

```
    unsigned char bufchar,ch,temp;
    unsigned int crc;
    int i;

    init_queue(&aux_buf);
    st_queue(&aux_buf, header[2], BUF_SIZE);
    st_queue(&aux_buf, header[3], BUF_SIZE);
    ch=(unsigned char) PC_id[0];
    st_queue(&aux_buf, ch, BUF_SIZE);
    ch=(unsigned char) PC_id[1];
    st_queue(&aux_buf, ch, BUF_SIZE);
    for (i=4; i<12; i++)
        st_queue(&aux_buf, header[i], BUF_SIZE);
    ch=type;
    st_queue(&aux_buf, ch, BUF_SIZE);
    ch=0;
    st_queue(&aux_buf, ch, BUF_SIZE);
    crc=calc_crc(aux_buf.q, aux_buf.st);
    bufchar=(unsigned char) crc;
    crc=(crc >> 8);
    temp=(unsigned char) crc;
    if(( temp == DLE ) || ( temp == ETB)){
       /* hace el byte stuffing */
       ch=DLE;
       st_queue(&aux_buf,ch,BUF_SIZE);
    }
    st_queue(&aux_buf, temp, BUF_SIZE);
    if(( bufchar == DLE ) || ( bufchar == ETB)){
       ch=DLE;
       st_queue(&aux_buf,ch,BUF_SIZE);
    }
    st_queue(&aux_buf, bufchar, BUF_SIZE);
    while(rt_queue(&aux_buf, &bufchar) != -1){
       write(recv_xmit, &bufchar, 1);
    }
    bufchar=ETB;
    write(recv_xmit, &bufchar, 1);
}


open_window(window_size)
int window_size[4];
/*
**
** PROPOSITO: Abre una ventana con el tamaño y testo asociado a ella.
** UTILIZA: open, ioctl, sleep, exit, strcat, strncat, y strcpy.
** LLAMA: setport
** LLAMADO POR: initially
**
*/
{
    int result, window;
    struct uwdata shape;

    if ((window=open("/dev/window", O_RDWR | O_NDELAY)) < 0){
       fprintf(stderr,"error al abrir ventana %d err: %d\n", errno);
       sleep(10);
       exit(1);
    }
    setport(window);
    ioctl(window, WIOCGETD, &shape);
    shape.uw_x=window_size[0];
    shape.uw_y=window_size[1];
    shape.uw_width=window_size[2];
    shape.uw_height=window_size[3];
```

```
    if ((result = ioctl(window, WIOCSETD, &shape)) == -1){
       fprintf(stderr,"ioctl en la ventana WIOCSETD %d,error
                        %d\n\r",errno);
       sleep(10);
       exit(1);
    }
    return (window);
}

conv_char(bufchar, auxbuf)
unsigned char bufchar;
unsigned char    auxbuf[3];
/*
**
** PROPOSITO: Convierte un entero en carácter.
** LLAMADO POR: echo_msg
**
*/
{
    unsigned int i;

    i=bufchar;
    auxbuf[0]='0';
    while (i >= 100){
       auxbuf[0]=auxbuf[0]+1;
       i=i-100;
    }
    auxbuf[1]='0';
    while (i >= 10){
       auxbuf[1]=auxbuf[1]+1;
       i=i-10;
    }
    auxbuf[2]='0';
    while (i > 0){
       auxbuf[2]=auxbuf[2]+1;
       i=i-1;
    }
}

open_file(fd,filename)
int *fd;
char filename[];
/*
**
** PROPOSITO : Abre o crea un archivo en disco. Necesita el nombre
** de archivo, incluyendo el path, y regresa el File Descriptor.
** Deja el archivo en modo de lectura/escritura.
** LLAMADO POR: recv_file.
** LLAMA    : open, chmod.
**
*/
{
     if((*fd= open(filename,O_RDWR | O_NDELAY | O_CREAT))<0){
               fprintf(stderr," error al abrir archivo.
                                 err:%d\n",errno);
               return (-1);
     }
     if(chmod(filename,00400|00200)<0){
               fprintf(stderr," error en chmod. err:%d\n",errno);
               return (-1);
     }
     return (1);
}
```

ARCHIVO DEL PROCESO DE ENTRADA (Entrada.C)

```
#include <fcntl.h>
#include <errno.h>
#include <termio.h>
#include <stdio.h>
#include <sys/window.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <string.h>
#include </usr/defred.h>


KB(KB_xmit, recv_KB, PC_id, window)
int KB_xmit, recv_KB, PC_id[3], window;
/*
**
**
** PROPOSITO: Lee continuamente del teclado, revisa sintaxis, envía
** paquetes al proceso receptor, para que los envíe a otros nodos.
** revisa si llego un ACK o un NACK para actualizar la tabla de ACKs.
** lleva el control de los paquetes a ser transmitidos, ya que
** mantiene la tabla de ACKs, cuidando las retransmisiones.
** LLAMA: syntax_check, init_queue, check_KB, checksyntax, st_queue,
** openfile, add_crc, flush_outbuf, update_blocktbl, y check_acktbl.
** LLAMADO POR: la función main del archivo RecTran.C
**
*/
{
   char letter[11];
   char tmp_buf[TMP_SIZE];
   unsigned char burchar, header[HEADER_SIZE], temp,
                 fname[FNAME_SIZE];
   int tmp_stat, syntaxerror, i, index, file_error, entry, blk;
   struct stat buf;
   unsigned int mid, crc;
   header[13]=0;              /* cuenta = 0 para todos los mensajes y
                                 archivos a enviar. */
   mid=1;                     /* Identificador de Mensaje empieza en 1 */
   quote=0;                   /* para hacer el "quote" de carácteres
                                 del teclado.*/
   ptr=0;                     /* apuntador de tmp_buf */
   done=0;
   index=0;
   for (i=0; i<TBL_SIZE; i++)
     /* pone disponibles las entradas en la tabla. */
     tbl[i].free=1;
   syntax_check(letter);
   /* Inicialización para revisión de sintaxis. */
   while (!done){
      tmp_stat=check_KB(window, tmp_buf);     /* lee del teclado  */
      if (tmp_stat){                          /* si es msg o arch */
         init_queue(&out_buf);                /* limpia out_buf */
         syntaxerror=checksyntax(letter, tmp_buf, window, header);
         if (!syntaxerror){
            header[2]=(unsigned char) PC_id[0];
            header[3]=(unsigned char) PC_id[1];
            header[7]=(unsigned char) mid;
            header[6]=(unsigned char) (mid >> 8);
            header[5]=(unsigned char) (mid >> 16);
            header[4]=(unsigned char) (mid >> 24);
```

```
        for (i=8; i<12; i++)  /* 1er paquete, nombre de arch */
           header[i]=0;
        file_error=0;
        if (hop_case){        /* opción con origen erróneo */
           header[2]=9;
           header[3]=9;
           hop_case=0;
        }
        if (fname_case)
           header[12]=FILE;       /* para enviar un archivo */
        else
           header[12]=MSG;        /* enviar un mensaje */
        for (i=0; i<HEADER_SIZE; i++){
           /* guarda el encabezado en out_buf */
           st_queue(&out_buf, header[i], BUF_SIZE);
        }
        if (fname_case){
           /* para enviar el nombre de archivo */
           for (i=0; i<FNAME_SIZE; i++)
              fname[i]=SP;
           i=0;
           bufchar=DLE;
           while(tmp_buf[point+1] != CR){
             /* obtiene el nombre de archivo de tmp_buf */
             if (tmp_buf[point] == DLE || tmp_buf[point] == ETB
                 || tmp_buf[point] == NULL)
                 st_queue(&out_buf, bufchar, BUF_SIZE);
             /* hace el byte stuffing */
             fname[i++]=tmp_buf[point++];
             st_queue(&out_buf, fname[i-1], BUF_SIZE);
           }
           fname[i]=NULL;
           st_queue(&out_buf, fname[i], BUF_SIZE);
           /* abre el archivo y guarda el primer paquete en la
              tabla de ACKs */
           file_error=openfile(fname, header, &entry, window);
           if (!file_error){
              add_crc();  /* calcula y adiciona el crc */
              flush_outbuf(KB_xmit, window);
              /* transmite el primer paquete */
              time(&tbl[entry].block[0].send_time);
              tbl[entry].block[0].try--;
              mid++;
           }
        }
        if (msg_case){        /* para enviar un mensaje */
           for (i=++point; i<ptr; i++)/* guarda msg en out_buf */
              st_queue(&out_buf, tmp_buf[i], TMP_SIZE);
           add_crc();
           /* calcula crc y se lo agrega al paquete. */
           flush_outbuf(KB_xmit, window);
           /* envia el paquete al proceso transmisor. */
           mid++;
        }
     }
     ptr=0;
  }
  for (i=0; i<ACK_WINDOW; i++){
     fstat(recv_KB, &buf);
     if (buf.st_size > 8){
        /* revisa si el receptor envio un ACK, NACK o FILE, para
           un paquete en particular */
        update_blocktbl(recv_KB, KB_xmit, window, header);
     }
  }
  /* revisa la tabla de ACKs para retransmitir */
```

```
        check_acktbl(&index, KB_xmit, window, header);
    }
}


check_KB(window, tmp_buf)
int window;
char tmp_buf[TMP_SIZE];
/*
**
** PROPOSITO: Lee del teclado haciendo echo a la ventana, y guarda el
** paquete en un buffer temporal, para después revisar la sintaxis.
** LLAMADO POR: KB.
**
*/
{
    int tmp_stat, x;
    char bufchar;

    tmp_stat=0;
    if (read(window, &bufchar, 1)){
        tmp_buf[ptr]=bufchar;

        /* Revisa si hay espacio en el buffer temporal */
        if (ptr < (TMP_SIZE-3) || tmp_buf[ptr] == BS ||
            tmp_buf[ptr] == CR){
            if (tmp_buf[ptr] == '$'){/* para el quote de un carácter */
                if (!quote){
                    write(window, &bufchar, 1);
                    ptr=ptr-1;
                    quote=1;
                }
                else{
                    write(window, &bufchar, 1);
                    quote=0;
                }
            }
            else{
                write(window, &bufchar, 1);
                if (tmp_buf[ptr] == BS){
                    ptr=ptr-1;
                    if (ptr > -1)
                        ptr=ptr-1;
                }
                if (tmp_buf[ptr] == CR){
                  tmp_buf[ptr]=NL;
                  ptr++;
                  tmp_buf[ptr]=CR;
                  tmp_stat=1;
                  bufchar=NL;
                  write(window, &bufchar, 1);
                }
                if (tmp_buf[ptr] == KL){
                  bufchar=CR;
                  write(window, &bufchar, 1);
                  for (x=0; x<TMP_SIZE; x++)
                      tmp_buf[x]=SP;
                  ptr=-1;
                }
                quote=0;
            }
            ptr=ptr+1;
        }
        else{                       /* el buffer se sobrelleno */
            bufchar=BL;
            write(window, &bufchar, 1);
```

```
        }
    }
    return(tmp_stat);
}


checksyntax(letter, tmp_buf, window, header)
char letter[11], tmp_buf[TMP_SIZE];
int window;
unsigned char header[HEADER_SIZE];
/*
**
** PROPOSITO: Revisa la sintaxis, si no despliega el mensaje
** apropiado. También revisa si el paquete contiene un archivo, un
** mensaje, y si el comando fue solicitado con alguna opción
** (opciones: -s, -e).
** LLAMA: error_display
** LLAMADO POR: k8.
**
*/
{

    int check_one, check_two, check_three, check_four;
    int i, pnt, syntax_error, size, flag;
    char bufchar, str[80];

    syntax_error=0;
    header[0]=0;
    header[1]=0;
    check_one=0;
    check_two=0;
    check_three=0; check_four=0;
    fname_case=0; msg_case=0; crc_case=0; hop_case=0;

    for (point=0; point<4; point++){  /* Es el comando send ? */
        if (tmp_buf[point] != letter[point])
            check_one=1;
    }
    if (check_one){                   /* Es el comando down ? */
        for (i=0; i<6; i++){
            pnt=5+i;
            if (tmp_buf[i] != letter[pnt])
                check_two=1;
        }
        if (!check_two)               /* Terminar la ejecución. */
            done=1;
    }
    if (!check_one){
        if (tmp_buf[point] == 'f'){   /* comando sendf (archivos) */
            fname_case=1;
            point++;
            if (tmp_buf[point] != SP){
                check_one=1;
                check_two=1;
            }
        }
        else{
            if (tmp_buf[point] == SP)
                msg_case=1;                 /* comando send (mensajes) */
            else{
                check_one=1;
                check_two=1;
            }
        }
        if (!check_one){
          if (tmp_buf[point+1] == '-'){  /* alguna opción? : s o e */
            point+=2;
```

```
            if (tmp_buf[point] == 'e'){
               /* enviar paquete con CRC errónea */
               crc_case=1;
               point++;
            }
            else{
               if(tmp_buf[point] == 's'){
                  /* enviar paquete con origen erróneo. */
                  hop_case=1;
                  point++;
               }
               else{
                  check_one=1;
                  check_two=1;
               }
            }
         }
      }
   }
}
if (!check_one){
   if (tmp_buf[point] != SP){
      check_one=1;
      check_two=1;
   }
   else{
      i=0;
      flag=0;
      while (tmp_buf[++point] != SP){
         /* convierte la dirección carácter a entero */
         flag=1;
         if (tmp_buf[point] > '9' || tmp_buf[point] < '0')
            check_three=1;
         i=(i*10)+(tmp_buf[point]-'0');
      }
      if (i < 0 || i > 255 || !flag)  /* grupo fuera de rango */
         check_three=1;
      else
         header[0]=i;    /* asigna el grupo destino. */
   }
}
if (!check_one && !check_three){
   /* convierte la dirección carácter a entero */
   i=0;
   flag=0;
   point++;
   while (tmp_buf[point] != SP && tmp_buf[point] != NL){
      flag=1;
      if (tmp_buf[point] > '9' || tmp_buf[point] < '0')
         check_three=1;
      i=(i*10)+(tmp_buf[point]-'0');
      point++;
   }
   if (i < 0 || i > 255 || !flag) /* nodo fuera de rango */
      check_three=1;
   else
      header[1]=i;   /* asigna el nodo destino */
   if (fname_case){
      /* revisa si se encuentra el nombre de Arch */
      while (tmp_buf[point] == SP)
         point++;
      if (tmp_buf[point] == NL && tmp_buf[point+1] == CR)
      check_four=1;
   }
}
if (check_one && check_two){
   strcpy(str, "***** comando desconocido\0");
```

```
        error_display(str, window);
    }
    if (check_three){
        strcpy(str, "***** destino invalido\0");
        error_display(str, window);
    }
    if (check_four){
        strcpy(str, "***** el nombre del archivo no se encontró\0");
        error_display(str, window);
    }
    if ((check_one || check_three) || check_four)
        syntax_error=1;
    return(syntax_error);
}


update_blocktbl(recv_KB, KB_xmit, window, header)
int recv_KB, KB_xmit, window;
unsigned char header[HEADER_SIZE];
/*
**
** PROPOSITO: Actualizar la tabla de ACKs, si se recibio alguna señal
** de ACK, NACK o FILE. Si se recibio algún NACK entonces
** retransmitir dicho paquete.
** UTILIZA: close, strcpy, strlen.
** LLAMA: error_display, xmit_block y disp_block_num.
** LLAMADO POR: KB.
**
*/
{
    int i, z, entry, file_found, block_found, id_error, j, k, size,
        file_error;
    char str[80];
    unsigned char aux[9], auxbuf[10], pkt_aux[4];
    int pkt_num;

    for (i=0; i<9; i++)   /* lee el IdM, número de paquete y la señal
                             mandada por el proceso receptor.*/
        read(recv_KB, &aux[i], 1);
    file_found=0;
    i=0;

    while (!file_found && (i < TBL_SIZE)){ /* busca en la tabla de
        ACKs un archivo existente con dicho IdM */
        id_error=0;
        k=0;
        while (!id_error && k < 4){
            if (aux[k] != tbl[i].header[k+4])
                id_error=1;
            else
                k++;
        }
        if (!id_error && !tbl[i].free)
            file_found=1;
        else
            i++;
    }
    if (file_found){
        block_found=0;
        j=0;
        pkt_num=W_to_I(&aux[4]);
        while (!block_found && (j < ACK_WINDOW)){
        /* busca un bloque con ese número de paquete.
            if (pkt_num == tbl[i].block[j].pkt_num &&
                !tbl[i].block[j].free)
                    block_found=1;
```

```
            else
                j++;
        }
        if (block_found){
            if (aux[8] == ACKF){
               tbl[i].block[j].free=1; /* libera el lugar de la tabla */
               tbl[i].block[j].stat=ACKF;
               tbl[i].last=tbl[i].block[j].pkt_num;
               if (tbl[i].last == tbl[i].num_pkts){
                /* Revisa si fue el último. */
                  tbl[i].free=1;        /* libera la entrada */
                  close(tbl[i].fd);
               }
            }
            else{
               if (aux[8] == FILE){
               /* revisa si el paquete lo enviamos nosotros mismos. */
                  tbl[i].free=1;        /* libera la entrada */
                  close(tbl[i].fd);
               }
               else{
                  if (tbl[i].block[j].try < 1){  /* revisa si tiene
                     oportunidades para ser retransmitido. */
                     tbl[i].block[j].free=1;
                     tbl[i].free=1;
                     close(tbl[i].fd);
                     strcpy(str, "**** transmisión fallida archivo =>
                                  \0");
                     size=strlen(str);
                     write(window, str, size);
                     strcpy(str, tbl[i].fname);
                     error_display(str, window);
                  }
                  else{
                     strcpy(str, "***** NACK recibido, retransmitiendo
                                  bloque \0");
                     size=strlen(str);
                     write(window, str, size);
                     disp_block_num(tbl[i].block[j].pkt_num, j, window);
                     xmit_block(i, j, KB_xmit, header, window);
                  }
               }
            }
        }
        else{
           strcpy(str, "***** bloque no existe en la tabla de
                        bloques.\0");
           error_display(str, window);
        }
    }
    else{
       strcpy(str, "***** archivo no existente en la tabla de ACKs
                    \0");
       error_display(str, window);
    }
}


check_acktbl(index, KB_xmit, window, header)
int *index, KB_xmit, window;
unsigned char header[HEADER_SIZE];
/*
** PROPOSITO: Revisa el tiempo transcurrido desde que cada paquete de
** la tabla de bloques fue transmitido, retransmitiendo los archivos
** cuyo tiempo haya transcurrido y que tengan oportunidades de
** retransmisión.
```

```
** UTILIZA : close, strcpy, strlen.
** LLAMA: load_block, error_display, xmit_file, y disp_block_num.
** LLAMADO POR: KB.
**
*/
{
   unsigned long int curr_time;
   int entry, blk, size, file_error, count, z;
   char str[80];

   count=0;
   while(tbl[*index].free && count < TBL_SIZE){
      /* busca un archivo en la tabla. */
      if (*index == (TBL_SIZE -1 ))
         *index=0;
      else
         *index=(*index % TBL_SIZE) + 1;
      count++;
   }
   entry=*index;
   if (count < TBL_SIZE){
      for (blk=0; blk<ACK_WINDOW; blk++){
         if (tbl[entry].block[blk].free){
          if (tbl[entry].next <= tbl[entry].num_pkts){
             tbl[entry].block[blk].free=0;
             load_block(entry, blk, KB_xmit, header, window);
          }
         }
         else{
          time(&curr_time);
          if ((curr_time - tbl[entry].block[blk].send_time) >
                MAX_TIME){
             if (tbl[entry].block[blk].try < 1){
               tbl[entry].block[blk].free=1; /* libera el bloque */
               tbl[entry].free=1;            /* libera la entrada */
               close(tbl[entry].fd);
               strcpy(str, "***** transmisión fallida, archivo =>
                                \0");
               size=strlen(str);
               write(window, str, size);
               strcpy(str, tbl[entry].fname);
               error_display(str, window);
             }
             else{
               strcpy(str, "***** timeout, retransmitiendo bloque
                                \0");
               size=strlen(str);
               write(window, str, size);
               disp_block_num(tbl[entry].block[blk].pkt_num, entry,
               window);
               xmit_block(entry, blk, KB_xmit, header, window);
             }
          }
         }
      }
   }
}


disp_block_num(pkt_num, entry, window)
int pkt_num;
int entry, window;
/*
**
** PROPOSITO: Despliega el número de block en la pantalla.
** UTILIZA: strcpy.
```

```
** LLAMA: conv_mid, error_display.
** LLAMADO POR: update_blocktbl, check_acktbl.
**
*/
{
   unsigned char auxbuf[10], bufchar;
   int signal, flag, x, size;
   char str[80];
   unsigned int i;

   for (x=0; x<10; x++)
      auxbuf[x]='0';
   conv_mid(pkt_num, auxbuf);
   signal=0;
   flag=0;

   for (x=0; x<10; x++){
      if (auxbuf[x]=='0' && !flag) signal++;
      else{
         write(window, &auxbuf[x], 1);
         flag=1;
      }
   }
   if (!flag){
      bufchar='0';
      write(window, &bufchar, 1);
   }

   strcpy(str, " of file => \0");
   size=strlen(str);
   write(window, str, size);
   strcpy(str, tbl[entry].fname);
   error_display(str, window);
}


load_block(entry, blk, KB_xmit, header, window)
int entry, blk, KB_xmit;
unsigned char header[HEADER_SIZE];
int window;
/*
**
** PROPOSITO: Lee el siguiente bloque de un archivo, lo carga a
** la tabla de bloques y lo transmite por primera vez.
** UTILIZA: read, strlen.
** LLAMA: xmit_block.
** LLAMADO POR: check_acktbl.
**
*/
{
   int i, size;

      tbl[entry].block[blk].pkt_num=tbl[entry].next;
      /* número del paquete. */
      tbl[entry].block[blk].try=4;
      for (i=0; i<PKT_SIZE; i++)
         tbl[entry].block[blk].data[i]=NULL;
      size=(read(tbl[entry].fd, tbl[entry].block[blk].data,
                 PKT_SIZE));
      tbl[entry].block[blk].size=size;
      xmit_block(entry, blk, KB_xmit, header, window);
      tbl[entry].next++;  /* número del paquete siguiente. */
}
```

```
xmit_block(entry, blk, KB_xmit, header, window)
int entry, blk, KB_xmit;
unsigned char header[HEADER_SIZE];
int window;
/*
**
** PROPOSITO: Inicializa el out_buf, revisa si el paquete es el
** último, o el primero. Guarda el paquete en el out_buf, calcula y
** adiciona el CRC al paquete y lo transmite. Decrementa el número de ** intentos
de re-transmisión que quedan, y guarda el tiempo en el
** que se mando el paquete.
** LLAMA: st_queue, add_crc, y flush_outbuf.
** LLAMADO POR: KB, update_tbl, y check_acktbl.
**
*/
{
   int fd, size, i;
   char str[80];
   unsigned char bufchar;

   init_queue(&out_buf);

   tbl[entry].header[11]=(unsigned char)
                         tbl[entry].block[blk].pkt_num;
   tbl[entry].header[10]=(unsigned char)
                         (tbl[entry].block[blk].pkt_num >> 8);
   tbl[entry].header[9]= (unsigned char)
                         (tbl[entry].block[blk].pkt_num >> 16);
   tbl[entry].header[8]= (unsigned char)
                         (tbl[entry].block[blk].pkt_num >> 24);

   if (tbl[entry].block[blk].pkt_num == tbl[entry].num_pkts)
      tbl[entry].header[12]=LAST; /* Es el último paquete del arch */
   for (i=0; i<HEADER_SIZE; i++)  /* Almacena el encabezado */
      st_queue(&out_buf, tbl[entry].header[i], BUF_SIZE);
   size= tbl[entry].block[blk].size;
   if (tbl[entry].block[blk].pkt_num == 0){
      fname_case=1;
      for (i=0; i<size; i++){
       /* hace el byte stuffing del nombre de archivo. */
       if ((tbl[entry].fname[i] == ETB || tbl[entry].fname[i] == DLE)
           || tbl[entry].fname[i] == NULL){
          bufchar=DLE;
          st_queue(&out_buf, bufchar, BUF_SIZE);
       }
       st_queue(&out_buf, tbl[entry].fname[i], BUF_SIZE);
      }
      bufchar=NULL;
      st_queue(&out_buf, bufchar, BUF_SIZE);
   }
   else{
      block_case=1;
      for (i=0; i<size; i++){ /* almacena los bloques en out_buf. */
        st_queue(&out_buf, tbl[entry].block[blk].data[i], BUF_SIZE);
      }
   }
   add_crc();  /* calcula y adiciona el crc al paquete. */
   flush_outbuf(KB_xmit, window);
   /* envía el paquete al proceso transmisor. */
   time(&tbl[entry].block[blk].send_time);
   /* guarda el tiempo de la transmisión */
   tbl[entry].block[blk].try--;
   /* decrementa el número de veces que se puede retransmitir */
   tbl[entry].header[12]=FILE;
}
```

```
openfile(fname, header, entry, window)
unsigned char fname[FNAME_SIZE], header[HEADER_SIZE];
int *entry, window;
/*
**
** PROPOSITO: Abre un archivo y abre la entrada en la tabla de ACKs
** UTILIZA: open, strcpy.
** LLAMA: error_display, entry_acktbl.
** LLAMADO POR: KB
**
*/
{
   int fd, file_error;
   char str[80];

   file_error=0;
   if ((fd=open(fname, O_RDWR)) < 0){
      strcpy(str, "***** archivo inexistente\0");
      error_display(str, window);
      file_error=1;
   }
   if (!file_error)
      *entry=entry_acktbl(fname, header, fd);
   return(file_error);
}

entry_acktbl(fname, header, fd)
unsigned char fname[FNAME_SIZE], header[HEADER_SIZE];
int fd;
/*
**
** PROPOSITO: Guarda la información del archivo en la tabla de ACKS.
** LLAMADO POR: openfile.
**
*/
{
   int index, i, num_pkts, z;
   struct stat buf;

   index=0;
   while (!tbl[index].free)          /* busca un lugar libre. */
      index++;
   for (i=0; i<HEADER_SIZE; i++)
      tbl[index].header[i]=header[i]; /* almacena el encabezado */
   tbl[index].fd=fd;                 /* almacena el FD del archivo */
   tbl[index].free=0;                /* lugar ocupado */
   tbl[index].next=1;    /* siguiente paquete he ser escrito */
   tbl[index].last=-1;   /* ultimo paquete con ACK */
   fstat(fd, &buf);
   num_pkts=(buf.st_size / PKT_SIZE);
   /* calcula el número de paquetes a enviar */
   if ((buf.st_size % PKT_SIZE) > 0)
      num_pkts++;
   tbl[index].num_pkts=num_pkts; /* número de paquetes */
   strcpy(tbl[index].fname, fname);
   /* guarda el nombre del archivo */
   for (i=0; i<PKT_SIZE; i++)
      tbl[index].block[0].data[i]=NULL;
   strcpy(tbl[index].block[0].data, fname);
   /* nombre de archivo almacenado en la tabla de bloques */
   tbl[index].block[0].size=strlen(fname);
   /* tamaño de los datos del bloque. */
   tbl[index].block[0].free=0;       /* bloque ocupado */
   tbl[index].block[0].pkt_num=0;    /* primer pkt */
   tbl[index].block[0].try=4;
```

```
   /* No total # de transmisiones posibles. */
   for (i=1; i<ACK_WINDOW; i++)
      tbl[index].block[i].free=1;
   /* libera las otras entradas en el bloque */
   return(index);
}


add_crc()
/**
**
** PROPOSITO: Calcula y adiciona el CRC-16 al paquete.
** LLAMA: calc_crc, st_queue.
** LLAMADO POR: KB, y xmit_block
**
*/
{
   unsigned int crc;
   unsigned char temp, bufchar;

   crc=calc_crc(out_buf.q, out_buf.st); /* calcula el crc */
   temp=(unsigned char) crc;
   if (crc_case){
      temp=temp+'5';  /* opción -e : agrega un CRC erróneo */
      crc_case=0;
   }
   crc=(crc >> 8);
   bufchar=(unsigned char) crc;
   st_queue(&out_buf, bufchar, BUF_SIZE); /* adiciona el CRC al Pkt*/
   st_queue(&out_buf, temp, BUF_SIZE);
}


flush_outbuf(KB_xmit, window)
int KB_xmit, window;
/*
**
**
** PROPOSITO: Transfiere el paquete al proceso transmisor.
** LLAMA: rt_queue, init_queue.
** LLAMADO POR: KB, y xmit_block.
**
**
*/
{
   int i;
   unsigned char bufchar, ch;

   for (i=0; i<HEADER_SIZE; i++){ /* transmite el encabezado */
      rt_queue(&out_buf, &bufchar);
      write(KB_xmit, &bufchar, 1);
   }
   if (fname_case){              /* transmite el nombre de archivo */
      rt_queue(&out_buf, &bufchar);
      while(bufchar != NULL){
         write(KB_xmit, &bufchar, 1);
         if (bufchar == DLE){    /* hace el byte stuffing */
           rt_queue(&out_buf, &bufchar);
           write(KB_xmit, &bufchar, 1);
         }
         rt_queue(&out_buf, &bufchar);
      }
      write(KB_xmit, &bufchar, 1);
      fname_case=0;
   }
```

```
    while(rt_queue(&out_buf, &bufchar) != -1){
        if ((bufchar == DLE) || (bufchar == ETB)){
            /* hace el byte stuffing */
            ch=DLE;
            write(KB_xmit, &ch, 1);
        }
        write(KB_xmit, &bufchar, 1);
    }
    ch=ETB;
    write(KB_xmit, &ch, 1);        /* transmite el carácter ETB */
    init_queue(&out_buf);
    msg_case=0;
    block_case=0;
}

error_display(str, window)
char str[80];
int window;
/*
**
** PROPOSITO: Escribe el mensaje de error a una ventana.
** LLAMADO POR: checksyntax, update_blocktbl, check_acktbl, openfile,
** y disp_block_num.
**
*/
{
    char bufchar;
    int size;

    size=strlen(str);
    write(window, str, size);
    bufchar=NL;
    write(window, &bufchar, 1);
    bufchar=CR;
    write(window, &bufchar, 1);
    bufchar=BL;
    write(window, &bufchar, 1);
}

syntax_check(letter)
char letter[11];
/*
**
** PROPOSITO: Inicializa un arreglo para la revisión de sintaxis.
** LLAMADO POR: KB
**
*/
{
    letter[0]='s';
    letter[1]='e';
    letter[2]='n';
    letter[3]='d';
    letter[4]=BSP;
    letter[5]='d';
    letter[6]='o';
    letter[7]='w';
    letter[8]='n';
    letter[9]=NL;
    letter[10]=CR;
}

get_PC_id(PC_id)
int PC_id[3];
```

```
/*
**
** PROPOSITO: Obtiene la dirección de la PC y el límite de la cuenta.
** LLAMADO POR: initially
**
*/
{
   int end;

   printf("Teclee el número de Grupo .. [1 a 255] = ");
   end=0;
   while(!end){
      scanf("%d", &PC_id[0]);
      if (PC_id[0] > 0 && PC_id[0] < 256)
         end=1;
   }
   printf("\r");
   printf("Teclee el número de Nodo  [1 a 255] = ");
   end=0;
   while(!end){
      scanf("%d", &PC_id[1]);
      if (PC_id[1] > 0 && PC_id[1] < 256)
         end=1;
   }
   printf("\r");
   printf("Teclee el límite de la cuenta  [0 a 255] = ");
   end=0;
   while(!end){
      scanf("%d", &PC_id[2]);
      if (PC_id[2] >= 0 && PC_id[2] < 256)
         end=1;
   }
   printf("\n\r");
}


init_queue(queue)
struct queue   *queue;        /* Apuntador a la cola */
/*
**
** PROPOSITO: Inicializa una cola, para que pueda ser utilizada.
** LLAMA    : Ninguna.
** LLAMADO POR : Puede ser llamada por cualquier función que use
** colas.
**
*/
{
   queue->st=0;
   queue->rt=0;
}


st_queue(queue, ch, maxlim)
struct queue *queue;  /* Cola en la que el carácter va a ser
                                 almacenado. */
unsigned char ch;       /* carácter a ser almacenado en la cola. */
int          maxlim;     /* Tamaño máximo de la cola */
/*
** PROPOSITO: Almacena un carácter en una cola, regresa un valor de
** -1 si la cola está llena, o un 0 si la operación fue realizada con
** éxito.
** LLAMA : Ninguna.
** LLAMADO POR : Puede ser llamada por cualquier función que utiliza
** colas.
**
*/
```

```
{
    if(queue->st == maxlim){
        return -1;
    }
    queue->q[queue->st]=ch;
    queue->st++;
    return 0;
}

rt_queue(queue, ch)
struct  queue   *queue;     /* Cola de donde se lee un carácter */
unsigned char   *ch;        /* Regresa el carácter leído. */
/*
**
** PROPOSITO: Lee un carácter de la cola. Regresa un valor de -1 si
** la cola está vacía, de lo contrario regresa un cero y el carácter
** en char.
** LLAMA : Ninguna.
** LLAMADA  POR: Puede ser llamada por cualquier función que utilize
** colas.
**
*/
{
   if(queue->rt == queue->st){
      *ch=NULL;
      return -1;
   }
   queue->rt++;
   *ch=queue->q[queue->rt-1];
   return 0;
}


setport(fd)
int fd;             /* file descriptor */
/*
**
** PROPOSITO: Pone el puerto en modo 'raw', de manera de que solo
** pueda ser accesado mediante instrucciones de bajo nivel.
** LLAMA: ioctl.
** LLAMADO POR: main, openwindow.
**
*/
{
struct termio newtio;   /* información de la interface del puerto. */
int result;             /* indica si hubo un error en ioctl */

   ioctl(fd,TCGETA,&newtio);
   newtio.c_cflag &= ~ (PARENB | CS7 | CBAUD);
   newtio.c_cflag |=   (CS8 | B9600 | CLOCAL | CREAD);
   newtio.c_iflag &= ~ (BRKINT | ISTRIP | ICRNL | IXON | IGNPAR |
                        INPCK);
   newtio.c_iflag |= (0);
   newtio.c_oflag &= ~ (OPOST);
   newtio.c_oflag |= (0);
   newtio.c_lflag &= ~ (ISIG | ICANON | ECHO);
   newtio.c_lflag |= (0);
   newtio.c_cc[4]  = 0;        /* carácter de EOF deshabilitado */
   newtio.c_cc[5]  = 0;        /* carácter de EOL deshabilitado */
   if ((result = ioctl(fd,TCSETA,&newtio)) == -1){
      fprintf(stderr,"Error al poner en modo raw el puerto, error = ,
                      %d\n\r",errno);
      exit(1);
   }
}
```

```
kill_close_all(remote, recv_xmit, KB_xmit, recv_KB, pid_port, window)
int remote, recv_xmit[2], KB_xmit[2], recv_KB[2], pid_port[2], window[2];
/*
**
** PROPOSITO: Cierra el puerto, ventanas y termina los procesos
** hijos.
** UTILIZA: close, y kill.
** LLAMADO POR: main.
**
*/
{
    int i;

    close(remote);
    for (i=0; i<2; i++){
        close(recv_xmit[i]);
        close(recv_KB[i]);
        close(KB_xmit[i]);
        close(window[i]);
    }
    for (i=0; i<2; i++)
        kill (pid_port[i], 9);
}


int calc_crc(pntr, count)
char  *pntr;
int   count;
/*
**
** PROPOSITO: Calcula el CRC-16 de un paquete.
** Regresa el valor en los dos bytes menos significativos de un
** entero, la variable count es el número de carácteres en el
** paquete.
**
*/
{
    unsigned int crc;
    int          i;

    crc = 0;
    crc = *(int *)pntr & HIGH2;
    pntr+= 2;
    while (--count>=0){
        if(count){
          crc ^=(int) ((unsigned char)*pntr++)<<8;
        for(i=0;i<8;i++){
          if(crc & MSB){
              crc=(crc<<1);
              crc^=CRCPOLY;
          }
          else
              crc=crc<<1;
        }
    }
    return (crc>>16);
}


conv_mid(mid,auxbuf)
unsigned int mid;
unsigned char auxbuf[10];
/*
**
** PROPOSITO: Convierte un entero a carácteres.
```

```
** LLAMADO POR: programa RecTran.c, funciones recv, y disp_block_num.
**
*/
{
    unsigned int x,i,comp;

    for(i=0;i<10;i++) auxbuf[i]='0';
    i=mid;
    comp=1000000000;
    x=0;
    while(i>0){
      auxbuf[x]='0';
      while(i>=comp){
         auxbuf[x]++;
         i=i-comp;
      }
      comp=comp/10;
      x++;
    }
}

W_to_I(array)
unsigned char array[4];
/*
**
** PROPOSITO: convierte cuatro carácteres en un entero.
** LLAMADO POR: programa Rectran.c, y check_acktbl.
**
*/
{   int i, num;

    num=array[0];
    for(i=1;i<4;i++){
       num=(num << 8);
       num=(num | array[i]);
    }
    return (num);
}
```

ARCHIVO DE DEFINICIONES (DefRed.H)

```
/* PROGRAMA: DefRed.h
** PROPOSITO: Proveo las constantes y variables globales al programa
** controlador de red, que se encuentra en los archivos, Entrada.c y
** RecTran.c
*/


#define CR 13                   /* return */
#define NL 12                   /* avance de linea, Line Feed */
#define BS  8                   /* retroceso, Back Space */
#define KL 21                   /* ^U  */
#define BL  7                   /* campana */
#define SP 32                   /* espacio */
#define ETB 23                  /* fin de transmisión */
#define DLE 16                  /* delete, borra, para enmascaramiento */
#define NULL 0                  /* carácter NULO. */

#define MACK 20                 /* tipo ACK para mensajes */
#define NMACK 22                /* tipo NACK para mensajes */
#define MSG  2                  /* tipo mensaje */
#define ACKF 10                 /* ACK para archivos */
#define     NACKF 11            /* NACK para archivos */
#define FILE 1                  /* tipo archivo */
#define LAST 3                  /* tipo ultimo paquete, de archivo */

#define MAX_TIME 15             /* límite para retransmitir */
#define FNAME_SIZE 20           /* tamaño del nombre de archivo */
#define TBL_SIZE 10             /* tamaño de la tabla de ACKs */
#define ACK_WINDOW 3            /* No de entradas en la tabla de bloques */
#define RC_MAX_TIME 30          /* máximo tiempo, para esperar antes de  */
                                /* terminar una recepción. */
#define RC_TBL_SIZE 10          /* tamaño de la tabla de recepción */
#define PKT_SIZE 100            /* tamaño de los bloques */
#define TMP_SIZE 93             /* tamaño del buffer temporal */
#define BUF_SIZE 310            /* tamaño de in_buf, aux_buf y out_buf */
#define HEADER_SIZE 14          /* tamaño del encabezado del paquete */
#define QUE_LEN 320             /* tamaño máximo para una cola */
#define HIGH2 0xffff0000        /* las siguientes constantes se utilizan */
#define MSB 0x80000000          /* para calcular el CRC-16 de los pkts. */
#define CRCPOLY 0x80050000

int done, point, ptr, quote, fname_case, msg_case,hop_case, crc_case,block_case;
char    cr[2];

struct packet{   /* guarda la información acerca de un bloque. */
            int             pkt_num;
            int               free;
            int               stat;
            int                try;
            int               size;
   unsigned char    data[PKT_SIZE];
   unsigned long int     send_time;
};
```

```
struct table{     /* guarda la información sobre un archivo */
                  /* ha ser mandado.                        */
   unsigned char header[HEADER_SIZE];
            int                  fd;
            int                next;
            int                last;
            int                free;
            int             num_pkts;
   struct packet  block[ACK_WINDOW];
   unsigned char  fname[PKT_SIZE];
} tbl[TBL_SIZE];

struct rec_table{ /* guarda la información de un archivo que */
                  /* se está recibiendo                     */
   unsigned char header[HEADER_SIZE];
            int                free;
            int                  fd;
            int                next;
   unsigned long int       recv_time;
} r_tbl[TBL_SIZE];

struct queue{     /* define las colas utilizadas como buffers */
                  /* de Entrada y Salida */
   unsigned char q[QUE_LEN];
   int           st, rt;
} in_buf, out_buf, aux_buf;
```

ARCHIVO PARA LA UTILERIA MAKE (Makefile)

```
Red: RecTran.o Entrada.o
     cc -o Red RecTran.o Entrada.o

Rectran.o: DefRed.h RecTran.c
     cc -c Rectran.c

Entrada.o: DefRed.h Entrada.c
     cc -c Entrada.c
```

BIBLIOGRAFIA.

Kernighan, W.B. y Ritchie, M.D. The C Programming Language. Englewood Cliffs, N.J. E.E.U.U. : Prentice-Hall Inc. 1988

Kernighan, W.B. y Ritchie, M.D. The UNIX programming environment. Englewood Cliffs, N.J. E.E.U.U. : Prentice-Hall Inc.

Lane, G.M. Data Communications software design. Boston, Mass. E.E.U.U. : Boyd & Fraser Publishing Co. 1985

Lefkon, D. A LAN Primer. Byte magazine. McGraw-Hill Inc. Julio 1987, Vol12. No.8.

Schildt, H. C: The Complete Reference. Berkeley, Ca. E.E.U.U. : Osborne McGraw-Hill. 1987

Tanenbaum, S.A. Structured computer organization. Englewood Cliffs, N.J. E.E.U.U. : Prentice-Hall Inc. 1984

Tazelar, J.M. PC communications. Byte magazine. McGraw-Hill Inc. Enero 1989, Vol14. No.1

Wetterston, E. CS457 DATA COMMUNICATIONS Course description. Fort Collins Co. E.E.U.U. : Colorado State University. Agosto 1988.