UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

IMPLANTACIÓN DE UNA RED EN LA DIEEC PARA INTEGRARLA A LA RED UNAM

T E S I S

QUE PARA OBTENER EL TÍTULO DE

INGENIERO EN COMPUTACIÓN

PRESENTAN

SANTIAGO ALFREDO DÍAZ AZUARA
RENÉ ARMANDO ISLAS TORRES

DIRECTOR DE TESIS: ING. ALEJANDRO RAMIREZ LOZADA

MÉXICO, D.F. 1995

Dios bendiga a
todos nuestros amigos y seres queridos,
por que ellos son nuestra bendición.

A MIS PADRES

SANTIAGO DÍAZ GÓMEZ
DOLORES ALBA AZUARA

POR TODO EL AMOR, CONSEJOS, APOYO, CONFIANZA Y SU GRAN PACIENCIA QUE ME BRINDARON, A LO LARGO DE MIS ESTUDIOS Y EN EL DESARROLLO DE ESTE TRABAJO.

A MIS HERMANOS

VERÓNICA PATRICIA
JUAN MANUEL

POR SU APOYO INCONDICIONAL, ENTUSIASMO Y CONSEJOS.

Quiero reconocer mi agradecimiento, admiración y respeto al

DR. ENRIQUE CABRERA BRAVO

por la dirección y apoyo a lo largo de esta tesis,
así como mi reconocimiento por haberme brindado su amistad.

A mi amigo

DR. EDUARDO ANDRADE

por sus consejos, su apoyo y su valiosa amistad.

A FANNY ARENAS MARTÍNEZ,

por su comprensión, colaboración y su invaluable trabajo en la corrección de estilo.

A mis amigos
Cuauhtemoc Barrera, Lurdes y Carlos Cortazar por su apoyo incondicional;
a Pedro Morales y a Eduardo Muñoz por sus comentarios y su confianza.
A Pedro Mexia por sus consejos y comentarios. Así como a todos aquellos
que no llegue a mencionar en este trabajo.

A mis padres

JESÚS ISLAS URIBE
RITA TORRES GALICIA

Quienes me brindaron todo su apoyo y
comprensión en todo momento.

A mis hermanos

JESÚS HUGO
ROCÍO ELVIA
NORMA LILIA

Por darme su cariño, entusiasmo, recomendaciones y apoyo;
porque no se abaten ante la adversidad

A mis compañeros y amigos:

Mario, Samuel, Jorge, Víctor, Gustavo, Miriam, Juanelo,
Paty, Charly, Lulú, Alicia, Rodolfo, Alfredo Shaguy, Christopher, Pilar, Adriana,
Alfredo Z., Héctor Boy, Mario Molina, Manuel, Vicky,
José, José Juan, Cinthya, Héctor, Willy, Nacho, Rodrigo, Bernardo

Irma, Verónica, Elia, Lulú O., Juanita, Lalo, Hugo O., Agustín
Tania, Rocha, Luis Fernando, Helguera, Patricio
Claudia, Silvia, Sandra, Cecilia, Elisa, Marú

Oscar, Gerardo, Ángel, Rafael, Fanny

Ing. Abel Clemente, Ing. Hilarión Simón, Ing. Bonilla

y a todas aquellas personas que me han apoyado en algun momento de mi vida,
y que siempre aportaron distintos puntos de vista a la vida. Gracias.

Al ING. ALEJANDRO RAMÍREZ LOZADA

agradecemos sus valiosas observaciones y comentarios para la elaboración de este trabajo.

Al grupo de cómputo del Instituto de Física de la UNAM:

Oscar Mendoza González
Gerardo Caballero Treviño
Rafael Gómez Hernández
José Juan Pliego Silva
Juan Carlos González Torres

Por sus sugerencias, ayuda y paciencia en cada etapa de la investigación.

a la FACULTAD DE INGENIERÍA
y a nuestra Alma Mater, la UNAM

donde realizamos nuestros estudios, así como la elaboración de esta tesis.

al INSTITUTO DE FÍSICA de la UNAM por su comprensión y apoyo.

A nuestros compañeros y amigos:

Alfredo Z., Héctor G., Manuel, Virginia, Mario, Samuel, Jorge, Victor, Gustavo, Miriam, Juan, Patricia, Carlos, Lourdes, Alicia, Rodolfo, Pilar, Adriana, Mario Molina, José Juan, Cinthya, Héctor, Wilfredo, Ignacio, Rodrigo, Bernardo, Silvia.

IMPLANTACIÓN DE UNA RED EN LA DIEEC PARA INTEGRARLA A LA RED UNAM

ÍNDICE

OBJETIVOS

A través del siguiente trabajo se cumplirán los siguientes objetivos:

1) Implantar en la *DIEEC* una red que funcione bajo el Ambiente Operativo *UNIX*.

2) Establecer la comunicación de la *DIEEC* con *DGSCA* para formar parte de Red *UNAM*.

3) Lograr comunicación de la *DIEEC* con otras Universidades a través de *BITNET*.

4) Ofrecer un Servicio de Comunicaciones a través de la Red a los Académicos y Estudiantes de la Facultad de Ingeniería.

INTRODUCCIÓN

Sin lugar a dudas, uno de los inventos más revolucionarios en el presente siglo es el sistema digital de cómputo o computadora, sistema consistente en elementos de diversa índole lógica y física, intimamente relacionados entre si, que de acuerdo a un algoritmo predefinido relaciona un conjunto de entradas, con un conjunto de salidas, generando información útil al usuario para auxiliarlo en la toma de decisiones.

A medida que las computadoras se convierten en herramientas de trabajo necesarias en casi todas las actividades profesionales, las terminales independientes van dejando de ser la solución para los usuarios que necesitan información de diversas fuentes, además de los tradicionales disco duro, *floppy* e incluso el *CD-ROM*, en estos casos la solución es utilizar un enlace de computadoras. En la actualidad las computadoras registran las transacciones que tienen lugar cada día en los grandes almacenes, se ocupan de las actividades bancarias, gestionan las reservaciones de los hoteles, líneas aéreas, y muchas otras actividades económicas que dependen por completo de las comunicaciones entre computadoras. A este grupo de computadoras y terminales en general, conectadas entre sí, a través de uno o varios medios de transmisión, se les conoce como red de computadoras; los medios de transmisión pueden ser desde un simple cable, hasta señales de microondas y/o satélite.

Una red de computadoras puede proporcionar un poderoso medio de comunicación entre personas que se encuentran muy alejadas entre sí. Con el empleo de una red es relativamente fácil, para dos o más personas que viven en lugares distantes, escribir juntos un informe; cuando un autor que se mantiene en línea hace un cambio en un documento, los otros pueden verlo de inmediato en lugar de esperar varios días para recibirlo por correo; esta rapidez hace que la cooperación entre individuos, y que anteriormente había sido imposible de establecer, pueda realizarse. A la larga el uso de las redes, como un medio para enriquecer la comunicación entre los seres humanos será más importante que una mayor economía, compartir recursos, etcétera.

En este trabajo se estableció una red que funciona bajo un ambiente *UNIX*, para optimizar los recursos de computadoras y periféricos. Se logro la comunicación, sesión remota, ejecución remota, recepción y transmisión de archivos (binarios y *ascii*), con otras máquinas dentro y fuera de la *DIEEC*, a través de *TELNET*, *FTP*, *UNIX*, y el subsistema de comunicación *UUCP*. Por último se ofrece una serie de servicios que brinda *INTERNET* (*FTP* anónimo, *ARCHIE*, *WWW*, etc.) a la comunidad de la facultad de ingeniería.

También en este trabajo se explicarán algunos puntos del funcionamiento de los protocolos de comunicación más importantes como son *TCP* e *IP*, así como una serie de ejemplos que proporcionaran una mejor comprensión de estos. Por último se expondrá el funcionamiento y creación de *sockets* bajo un ambiente *UNIX* por medio de un programa en lenguaje C. Este programa fue compilado y desarrollado en un DECsystem 3100, con tecnología RISC y sistema operativo ULTRIX 3.1 de DIGITAL.

CAPITULO 1 Características de una red

La finalidad concreta que tienen las redes es transferir información entre computadoras y terminales; el uso que se le da a la información es lo que permite generar, una productividad y eficiencia en las labores diarias; sin embargo, esto dependerá de las características físicas y lógicas que presente la red.

1.1 TOPOLOGÍAS Y OBJETIVOS DE DISEÑO

Antes de continuar, conviene definir algunos términos utilizados. En la figura 1-1 se muestra un sencillo sistema de comunicación de datos. El proceso de aplicación (*PA*) es la aplicación que maneja el usuario final; suele tratarse de un programa de computadora o terminal de usuario, por ejemplo, los programas de contabilidad, nominas, control de inventario, etc. En la figura 1-1, el *nodo A* podría ejecutar un proceso de aplicación (PA_{A1}) en forma de programa, para tener acceso a otro proceso de aplicación situado en el *nodo B* (que en este caso incluye un programa [PA_{B1}] y una base de datos). También aparece un programa en el *nodo B* (PA_{B2}), que tiene acceso al *nodo A*, a través del programa de aplicación [PA_{A2}].

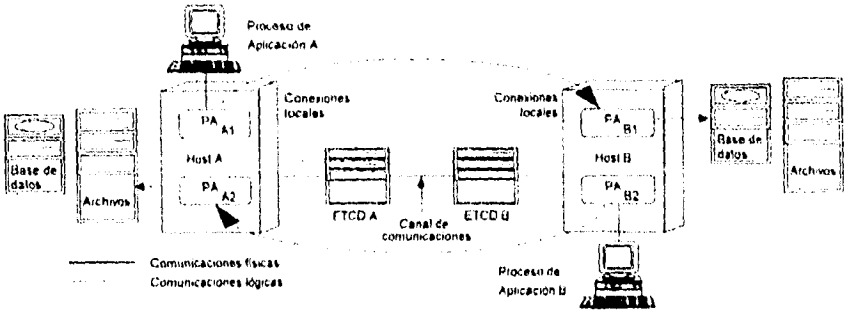


Figura 1-1. Sistema de comunicaciones

La aplicación reside en el equipo terminal de datos, o *ETD*; estas siglas con frecuencia se emplean en forma genérica para aludir a la máquina que emplea el usuario final. Un *ETD* puede ser una gran computadora, del tipo de las *IBM*, o una máquina más pequeña, como una terminal o computadora personal, conocidas muy a menudo como *host*.

La misión de las redes informáticas es conectar distintos *ETD* para que compartan recursos, intercambien datos y se apoyen mutuamente. En la figura 1-1, la red proporciona comunicaciones físicas y lógicas entre las computadoras y terminales conectados a ella. Las aplicaciones y los archivos emplean el canal físico para efectuar comunicaciones lógicas. En este contexto, al utilizar el término lógico, se indica que el *ETD* no tiene porqué conocer los aspectos físicos del procedimiento de comunicación. La *aplicación A1* sólo necesita generar una solicitud lógica de lectura que incluya una identificación de los datos; a su vez el sistema de comunicaciones será responsable de transportar esta solicitud de lectura hasta la *aplicación B1*, por medio de los canales físicos.

De la figura 1-1 se observa un equipo de terminación del circuito de datos (*ETCD*), también llamado equipo de comunicación de datos. Su misión es conectar los equipos *ETD* a la línea o canal de comunicaciones. Los primeros *ETCD*, fueron diseñados como dispositivos exclusivos de comunicaciones; sin embargo, en los últimos años estos equipos han incorporado más funciones que contienen parte de los procesos de aplicación. De cualquier modo, la principal función de un *ETCD* es servir de interfaz entre el *ETD* y la red de comunicaciones; como es el caso de un *módem*.

Las interfaces se especifican y establecen mediante protocolos. Los protocolos son acuerdos acerca de la forma en que se comunican entre sí los *ETD* y los dispositivos de comunicaciones, y pueden incluir regulaciones concretas que recomienden u obliguen a que se aplique una técnica o convenio determinados. Por lo general, son varios los niveles de interfaces y protocolos que necesitan las aplicaciones de usuario para funcionar. Los *ETD* y los *ETCD* pueden conectarse de dos formas; en la figura 1-1, los equipos están conectados en una configuración **punto a punto**, en la cual solo existen dos dispositivos *ETD* o *Hosts* por cada línea o canal de comunicación. En la figura 1-2 aparece una configuración distinta, llamada **multipunto**, en la cual hay más de dos dispositivos conectados a un mismo canal.

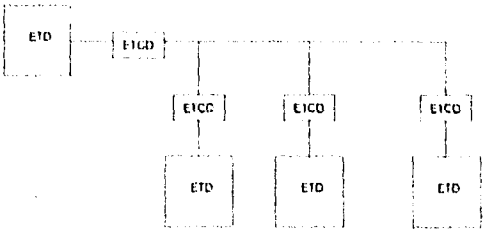


Figura 1-2 Circuitos multipunto

Las ventajas de las redes de comunicación que se han visto hasta el momento, no podrían hacerse realidad sin la participación de un componente muy importante, llamado *ECD* (Equipo de Conmutación de Datos). La figura 1-3 ilustra el uso de un *ECD* junto con varios *ETD* y *ETCD*; como sus siglas lo indican, la función principal del *ECD* es conmutar o encaminar el tráfico (datos de usuario) hasta su destino final, a través de la red. El *ECD* proporciona las funciones vitales de encaminamiento por la red, evitando los dispositivos y canales ocupados o fuera de servicio; asimismo el *ECD* puede dirigir los datos hacia su destino final a través de componentes intermedios, que pueden ser, a su vez, otros equipos de conmutación.

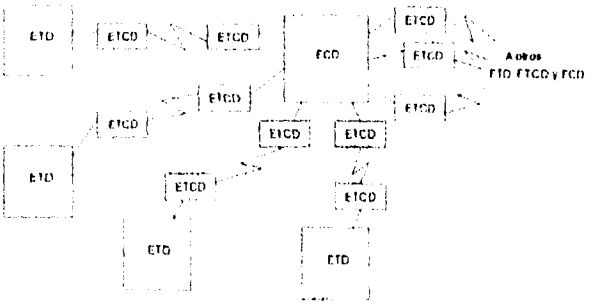


Figura 1-3 Equipo de Conmutación de Datos (ECD)

La ***topología*** es la disposición física del cable y la forma en que están conectados los dispositivos de una red; puede ser tan simple o compleja, según las necesidades del usuario. Se utilizan varios modelos principales como topologías de red para las diferentes redes de comunicación, como son: jerárquica, anillo, bus, estrella, en malla e híbrido. En el momento de establecer la topología de una red, el diseñador ha de plantearse tres objetivos principales :

- Proporcionar la máxima confiabilidad posible, para garantizar la recepción correcta de todo el tráfico.
- Encaminar el tráfico entre el *ETD* transmisor y el receptor a través del camino más económico dentro de la red (aunque, si se consideran más importantes otros factores, como la confiabilidad, este camino de costo mínimo puede no ser el más conveniente).
- Proporcionar al usuario final un tiempo de respuesta optimo y un caudal eficaz máximo.

La confiabilidad de una red es la capacidad que tiene para transportar datos correctamente de un *ETD* a otro, es decir, sin un solo error; esto incluye también la capacidad de recuperación de errores o datos perdidos en la red, ya sea por falla del canal, del *ETD*, del *ETCD* o los *ECD*. La confiabilidad está relacionada también con el mantenimiento del sistema, en el que se incluyen las comprobaciones diarias; el mantenimiento preventivo, que se ocupa de relevar de sus tareas a los componentes averiados o de funcionamiento incorrecto y, en su caso, el aislamiento de los focos de averías. Cuando un componente crea problemas, el sistema de diagnóstico de la red debe ser capaz de identificar y de localizar el error, aislar la avería y, si es preciso, aislar del resto de la red el componente defectuoso. El segundo objetivo a cumplir es proporcionar a los procesos de aplicación que residen en los *ETD*, el camino más económico posible. Para esto es preciso:

1. Minimizar la longitud real del canal que une a los componentes, lo cual suele implicar el encaminamiento del tráfico a través del menor número posible de componentes intermedios

2. Proporcionar el canal más económico para cada actividad concreta; por ejemplo transmitir los datos de baja prioridad a través de un enlace de baja velocidad por línea telefónica normal, lo cual es más barato que transmitir esos mismos datos a través de un canal vía satélite de alta velocidad

El tercer objetivo es obtener un tiempo de respuesta mínimo, y la forma de lograrlo es reducir el retardo entre la transmisión y recepción de datos de un *ETD* a otro. Además, se requiere de un caudal eficaz de datos o lo más elevado posible. A continuación se describen las topologías comúnmente empleadas en una red:

- La topología de árbol o jerárquica proporciona un punto de concentración de las tareas de control y solución de errores; en la mayoría de los casos el nodo situado en el nivel más elevado de la jerarquía es el que controla la red. Muchos fabricantes incorporan a esta topología un cierto carácter distribuido, dotando a los nodos subordinados de un control directo sobre los nodos situados en niveles inferiores dentro de la jerarquía, lo cual, reduce la carga de trabajo del nodo central. En determinadas situaciones, el nodo más elevado, tiene que controlar todo el tráfico entre los distintos subnodos; este hecho no sólo puede crear saturaciones de datos, sino que además plantea serios problemas de confiabilidad. Si el nodo principal falla, toda la red deja de funcionar, a no ser que exista otro nodo de reserva capaz de hacerse cargo de todas las funciones del nodo averiado. (figura 1-4a)

- La topología de *Bus* es un canal de comunicaciones conectado a las computadoras (o nodos); los mensajes viajan por este canal a 10 *mbps* (*megabits* por segundo) y cada nodo tiene una dirección destino. La principal limitante es el hecho de que existe un solo canal de comunicaciones para todos los dispositivos de la red; en consecuencia, si el canal de comunicaciones falla, toda la red deja de funcionar; además puede presentarse el fenómeno llamado colisión, que es cuando dos o más nodos quieren emplear el canal al mismo tiempo; otro inconveniente estriba en la dificultad de aislar las averías de los componentes individuales conectados al *bus*. Hay fabricantes que proporcionan canales alternativos por si falla el canal principal, y otros ofrecen conmutadores que permiten rodear a un nodo, en caso de que este falle. (figura 1-4b)

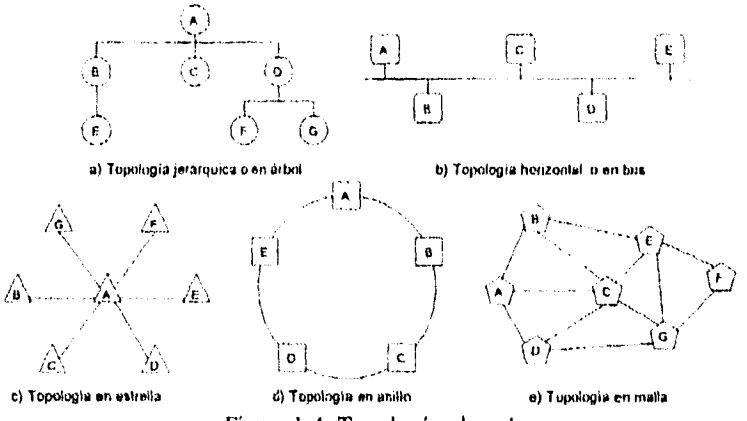


Figura 1-4. Topologías de red

- La topología en estrella recibe tal nombre por su aspecto; en el centro del complejo se encuentra una computadora que se le conoce como nodo central o servidor; cada nodo está conectado al servidor. El servidor abre y procesa los mensajes de los otros nodos. El servidor se encarga de localizar las averías; esto es sencillo ya que es posible aislar las líneas para identificar el problema; sin embargo, y al igual que en la estructura jerárquica, una red en estrella puede sufrir saturaciones y problemas en caso de avería del nodo central, es decir, mientras el nodo central esté en operación, lo estará la red. (figura 1-4c)
- La topología de anillo consiste en una serie circular de estaciones denominadas nodos. Cada nodo de esta red esta conectado a otros dos para formar un gran círculo. La transmisión de datos y mensajes es indirecta. Una transmisión tiene que viajar a una velocidad de 4 a 16 *mbps* de un nodo al siguiente a través del círculo, hasta alcanzar el destino correcto. El problema más importante de esta topología es que todos los componentes del anillo están unidos por un mismo canal; si falla el canal entre dos nodos, toda la red se interrumpe; algunos fabricantes construyen conmutadores, de tal manera que si hay una avería en un nodo, estos redirigen los datos automáticamente, saltándose el nodo averiado hasta el siguiente nodo del anillo, con el fin de evitar que el fallo afecte a toda la red. (figura 1-4d)
- La topología en malla se ha estado empleando en los últimos años; lo que la hace atractiva es su relativa inmunidad a los problemas de embotellamiento y averías; gracias a la multiplicidad de caminos que ofrece a través de los distintos nodos, es posible orientar el tráfico por trayectorias alternativas en caso de que algún nodo este averiado u ocupado. (figura 1-4e)

- El híbrido se conforma de dos o más tipos de topologías conectadas entre sí. Por ejemplo, una computadora central puede conectar a varias redes tipo bus, creando un híbrido de topologías *bus* y estrella.

La red de computadoras que se conectan para obtener mayor velocidad en las comunicaciones dentro de un edificio o un conjunto de edificios, recibe el nombre de **redes de área local (*LAN's*)**. Las *LAN's*, acopladas a *LAN's* cercanas para ampliar su distancia o mejorar su rendimiento, se denominan *LAN's* extendidas. Las redes que utilizan líneas telefónicas, microondas y señales de satélite para cubrir mayores distancias se denominan ***redes de área DILATADA*** o ***WAN's***. No importa si la red utiliza una *LAN*, una *LAN* extendida o una *WAN*, el usuario ve la función de red como si fuera una sola entidad.

1.2 DISEÑO DE UNA RED

Las redes modernas están diseñadas en una forma muy estructurada. La mayoría de éstas se organizan en una serie de niveles o capas con objeto de reducir la complejidad de su diseño. Cada uno de los niveles se construye sobre su predecesor, el número, nombre, contenido y función de cada nivel varía de una red a otra. Sin embargo, en cualquier red el propósito de cada nivel es ofrecer ciertos servicios a los niveles superiores, liberándolos del conocimiento detallado de cómo se realizan dichos servicios.

El nivel *n* de una máquina, se comunica con el nivel *n* de otra máquina. Las reglas y convenciones utilizadas en esta conexión, se conocen conjuntamente como protocolo de nivel *n*. A las entidades que forman los niveles correspondientes, en máquinas diferentes, se les denomina procesos pares. En otras palabras, son los procesos pares los que se comunican mediante el uso del protocolo.

En realidad no existe una transferencia directa de datos desde el nivel *n* de una máquina al nivel *n* de otra; sino que cada nivel pasa la información de datos y control al nivel inferior inmediato, y así sucesivamente hasta que se alcanza el nivel localizado en la parte más baja de la estructura. Debajo del nivel uno está el medio físico, a través del cual se realiza la comunicación real. Entre cada par de niveles adyacentes hay una interfaz la cual define los servicios y operaciones primitivas que el nivel inferior ofrece al superior. Cuando los diseñadores de redes deciden el número de niveles por incluir en una red, así como lo que cada uno de ellos deberá de hacer, una de las consideraciones más importantes consiste en definir claramente las interfaces entre niveles. El diseño claro y limpio de una interfaz, además de minimizar la cantidad de información que debe pasarse entre niveles, hace más simple la sustitución de un nivel por otro totalmente diferente. Así todo lo que se necesita del nuevo nivel, es que ofrezca exactamente el mismo conjunto de servicios al nivel superior contiguo, tal como lo hacía el nivel anterior.

Al conjunto de niveles y protocolos se le denomina arquitectura de la red. Las especificaciones de ésta, deberán contener información que permita al diseñador escribir o construir el *hardware* correspondiente a cada nivel, de tal forma que siga en forma correcta el protocolo apropiado.

Considérese ahora el siguiente ejemplo donde se proporciona información al nivel superior de una red de siete niveles mostrada en la figura 1-5. Un proceso que se está ejecutando en el *nivel siete* produce un mensaje m, el cual pasa del *nivel siete* al *nivel seis* de acuerdo con la definición de la interfaz de *nivel 6/7*. El *nivel seis* transforma de cierta manera el *mensaje* (por ejemplo, mediante una compresión

de texto), y lo pasa como un nuevo *mensaje M* al *nivel cinco*, a través de la *interfaz 5/6*. En este ejemplo, el *nivel cinco* no modifica el *mensaje*, sino únicamente regula la dirección de flujo (es decir, evita que algún *mensaje* de entrada sea considerado por el nivel seis, mientras éste se encuentra ocupada enviando una serie de mensajes de salida al *nivel cinco*).

En general no existe ningún límite en el tamaño de los *mensajes* que son aceptados por el *nivel cuatro*, sino que este es impuesto por el *nivel tres*. Por consiguiente, el *nivel cuatro* deberá dividir el mensaje de entrada en unidades del tamaño aceptado por el *nivel tres*, y colocar una *cabecera* en cada una de ellas. Esta *cabecera* incluye información de control como números de secuencia, mediante los cuales se logra que el *nivel cuatro* en la máquina destinataria pueda reconstruir el *mensaje* mediante la colocación correcta de las unidades; también en muchos niveles las cabeceras incluyen campos relacionados con el tamaño, tiempo y otros tipos de control.

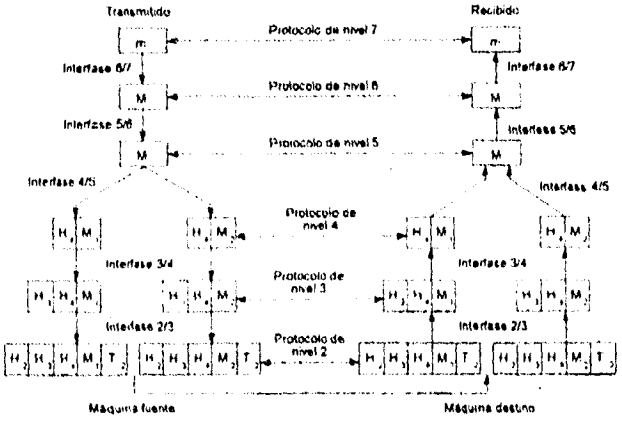


Figura 1-5. Ejemplo del flujo de información que soporta la comunicación virtual en el nivel 7.

El *nivel tres* se encarga de decidir cual de las líneas de salida va a utilizarse, coloca las cabeceras apropiadas y pasa los datos al *nivel dos*. En el nivel dos, no solo se añade una cabecera a cada una de las unidades, sino que también una etiqueta al final y entrega la unidad resultante al nivel uno para su transmisión física. En la máquina receptora, el mensaje se mueve de nivel en nivel hacia la parte superior, y las cabeceras se van retirando a medida que ascienden. Ninguna de las cabeceras correspondientes a los niveles inferiores al *n* pasan a éste. La importancia de la figura 1-5 radica en el entendimiento de la relación entre la comunicación virtual y la real, y la diferencia entre protocolos e interfaces.

Desde el punto de vista conceptual, los procesos pares del *nivel cuatro*, conciben su comunicación como si fuera horizontal, utilizando el protocolo del nivel cuatro. Así, cada uno pareciera que utiliza procedimientos llamados *enviar al otro lado* y *obtener del otro lado*, aun cuando estos realmente se comuniquen con los niveles inferiores, a través de la *interfaz 3/4*, y no con el otro lado. Cada nivel deberá tener un mecanismo para el establecimiento de la conexión. El mecanismo para terminar una conexión dentro de una red, una vez que esta ya no se necesita, esta intimamente ligado con aquel que se utiliza para establecerla.

La abstracción del proceso par es vital para el diseño de redes, sin esta técnica de abstracción será difícil, sino es que imposible, dividir el diseño de una red completa. El protocolo debe ser capaz de determinar el número de canales lógicos que corresponden a la conexión y sus prioridades. Existen varios métodos de comunicación; el término *full-dúplex* implica la transmisión y recepción simultánea de señales. En todas las redes digitales, esto se logra mediante dos pares de alambres. En las redes analógicas, esto se logra dividiendo el ancho de banda de la línea en dos juegos de frecuencias; uno para el envío, y el otro para la recepción. La comunicación *Half dúplex* significa transmisión en dos direcciones, pero solo en una dirección a la vez. Un radio Banda Civil es un ejemplo de transmisión *half-dúplex*. A cada instante, la transmisión es en un solo sentido. Su protocolo es «habla y di "cambio" para indicar fin de la transmisión». Por último, la comunicación *Simplex* es el método más sencillo en donde los datos viajan en una sola dirección.

- Dada la imperfección de los circuitos físicos de comunicación, el procedimiento para el control de errores es un aspecto de gran relevancia. En la actualidad se conocen varios códigos detectores y correctores de error, pero lo importante aquí es que los dos extremos de la conexión estén de acuerdo en cual utilizar. Además el receptor debe tener alguna forma de indicar al emisor qué mensajes se han recibido correctamente y cuales no.
- No todos los canales de comunicación mantienen el orden de los mensajes que les envían, de tal manera que, para recuperar una posible pérdida en la secuencia del mensaje, el protocolo deberá establecer, en forma explícita, un procedimiento seguro que permita al receptor colocar las unidades nuevamente en su forma original.
- Es necesario contemplar en cada uno de los niveles la protección de un receptor lento de una cantidad abrumadora de datos enviados por un transmisor rápido.
- Es común en varios niveles tener la incapacidad para aceptar mensajes arbitrariamente extensos por todos los procesos, esta propiedad nos conduce a mecanismos de segmentación, transmisión y ensamblaje de mensajes.
- Otro aspecto relacionado con el anterior, es el correspondiente a qué hacer cuando los procesos insisten en transmitir datos en unidades tan pequeñas, que su envío en forma separada, los hace muy ineficientes. Una solución en este caso sería, reunir varios de estos pequeños mensajes con encabezamientos, dirigidos a un destino común en un solo mensaje de gran extensión, de tal forma que al llegar al otro extremo, solo se tengan que volver a separar.
- Resulta inconveniente y muy costoso establecer una conexión para cada par de procesos comunicantes, en tales casos el nivel subyacente puede decidir utilizar la misma conexión para conversaciones múltiples, sin que éstas tengan necesariamente relación alguna. Este procedimiento se puede utilizar en cualquier nivel mientras que el proceso demúltiplexaje y múltiplexaje se haga en forma transparente. En el nivel físico, se utiliza el proceso de múltiplexaje dado que todo el tráfico de las conexiones, se tiene que enviar sobre un número reducido de circuitos físicos.
- Siempre que existan caminos múltiples entre la fuente y el destino se debe escoger una ruta en función del acceso. Algunas veces esta decisión debe tomarse en dos o más niveles.

1.3 El modelo de referencia OSI

El modelo de referencia *OSI* se ha estado gestando durante varios años. Este estándar es apoyado por los principales organismos de normalización, administraciones de telecomunicación y empresas. La Organización Internacional de Normalización (*ISO*) es un cuerpo voluntario. Está integrado por los organismos normalizadores de los diferentes países miembros. En *ISO* intervienen principalmente los comités de usuarios y los fabricantes.

El Comité Consultivo Internacional de Telefonía y Telegrafía (*CCITT*) es miembro de la Unión Internacional de Telecomunicaciones (*ITU*), organismo de cooperación internacional. El *CCITT* ha apoyado numerosos estándares, sobre todo en el campo de las redes de comunicación de datos, conmutación telefónica, sistemas digitales y terminales.

La organización *ISO* y el *CCITT* han desarrollado el modelo de referencia *OSI* para definir redes estratificadas y protocolos con varios niveles. Estos son los objetivos que persigue el modelo *OSI*:

- Proporcionar una serie de normas para la comunicación entre sistemas.
- Eliminar todos los impedimentos técnicos que pudieran existir para la comunicación entre sistemas.
- Abstraer el funcionamiento interno de los sistemas individuales.
- Definir los puntos de interconexión para el intercambio de información entre los sistemas.
- Ofrecer un punto de partida válido desde el cual comenzar en caso de que las normas del estándar no satisfagan todas las necesidades.

El modelo de referencia *OSI* consta de siete niveles, cuya estructura se muestra en la figura 1-6.

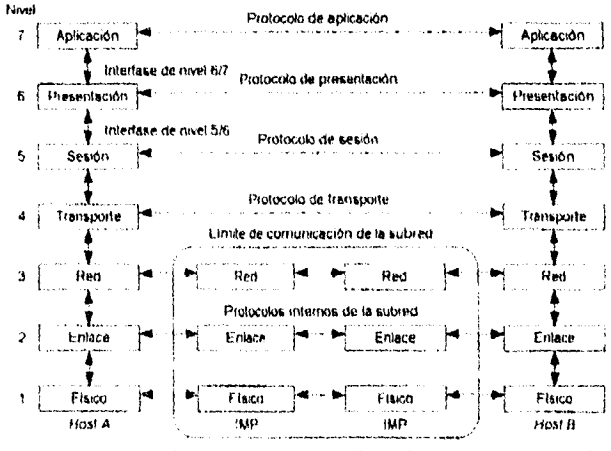


Figura 1-6. Arquitectura de una red basada en el modelo *OSI*

1.3.1 Nivel físico

El nivel físico se ocupa de la transmisión de *bits* a lo largo de un canal de comunicación. Su diseño debe asegurar que cuando un extremo envía un *bit* con valor 1, éste se reciba exactamente como un *bit* con ese valor en el otro extremo, y no como un *bit* de valor 0. Las preguntas comunes aquí son cuántos voltios deberán utilizarse para representar un *bit* de valor 1 o 0; cuántos microsegundos deberá durar un *bit*; la posibilidad de realizar transmisiones bidireccionales en forma simultánea; la forma de establecer la conexión inicial y cómo interrumpirla cuando ambos extremos terminan su comunicación; o bien, cuántas puntas terminales tiene el conector de la red y cuál es el uso de cada una de ellas. Los problemas de diseño a considerar aquí son los aspectos mecánico, eléctrico, de procedimiento de interfaz y el medio de transmisión física, que se encuentra bajo el nivel físico.

1.3.2 Nivel de enlace

La tarea primordial del nivel de enlace consiste en, a partir de un medio de transmisión común y corriente, transformarlo en una línea sin errores de transmisión para el nivel de red. Esta tarea la realiza al hacer que el emisor recorte la entrada de datos en tramas de datos (típicamente constituidas por algunos cientos de octetos), y las transmita en forma secuencial y procese las tramas de aprobación, devueltas por el receptor. Como el nivel físico básicamente acepta y transmite un flujo de *bits* sin tener en cuenta su significado o estructura, recae sobre el nivel de enlace la creación o reconocimiento de los límites de la trama. Esto puede llevarse a cabo mediante la inclusión de un patrón de *bit* especial al inicio y al término de la trama. Si estos patrones de *bits* pueden aparecer entre los datos, deberá tenerse un cuidado especial para evitar cualquier confusión al respecto.

La trama puede destruirse por completo debido a una ráfaga de ruido en la línea, en cuyo caso el software del nivel de enlace, perteneciente a la máquina emisora, deberá retransmitir la trama. Sin embargo, múltiples transmisiones de la misma trama introducen la posibilidad de duplicar la misma. Por ejemplo, el duplicado de una trama podría enviarse, si el acuse de recibo que regresa al receptor se hubiera destruido. Corresponde a este nivel resolver los problemas causados por daño, pérdida o duplicidad de tramas, así como la regulación del tráfico, para evitar que un transmisor muy rápido sature con datos a un receptor lento.

1.3.3 Nivel de red

El nivel de red se ocupa del control de la operación de la subred. Un punto de suma importancia en su diseño, es la determinación sobre cómo encaminar los paquetes del origen al destino. Las rutas podrían basarse en tablas estáticas que se encuentran "cableadas" en la red y que difícilmente podrían cambiarse. También, podrían determinarse al inicio de cada conversación, por ejemplo en una sesión de terminal. Por último, podrían ser de tipo dinámico, determinándose en forma diferente para cada paquete, reflejando la carga real de la red. Si en un momento dado hay demasiados paquetes presentes en la subred, ellos mismos se obstruirían mutuamente y darán lugar a un cuello de botella. El control de tal congestión dependerá también del nivel de red. Cuando un paquete tenga que desplazarse de una red a otra para llegar a su destino. El direccionamiento utilizado en la segunda red puede ser diferente al empleado en la primera. La segunda podría no aceptar el paquete en su totalidad, por ser demasiado grande o los protocolos podrían ser diferentes; la responsabilidad, para problemas de interconexión de redes heterogéneas recaerá, en todo caso, en el nivel de red.

1.3.4 Nivel de transporte

La función principal del nivel de transporte consiste en aceptar los datos del nivel de sesión, dividirlos, siempre que sea necesario, en unidades más pequeñas, pasarlos al nivel de red y asegurar que todos ellos lleguen correctamente al otro extremo. Además, todo este trabajo se debe hacer de manera eficiente, de tal forma que aísle el nivel de sesión de los cambios inevitables a los que esta sujeta la tecnología del *hardware*.

Bajo condiciones normales, el nivel de transporte crea una conexión de red distinta para cada conexión de transporte solicitada por el nivel de sesión. Si la conexión de transporte necesita un gran caudal, ésta podría crear múltiples conexiones de red, dividiendo los datos entre las conexiones de la red con objeto de mejorar dicho caudal. Por otra parte, si la creación o mantenimiento de la conexión de una red resulta costoso, el nivel de transporte podría multiplexar varias conexiones de transporte sobre la misma conexión de red para reducir dicho costo. En todos los casos, el nivel de transporte se necesita para hacer el trabajo de multiplexión transparente al nivel de sesión.

El nivel de transporte determina qué tipo de servicio debe dar al nivel de sesión, y en último término a los usuarios de la red. El tipo más popular de conexión de transporte corresponde al canal punto a punto sin error, por medio del cual se entregan los mensajes en el mismo orden en que fueron enviados. Sin embargo, el transporte de mensajes aislados sin garantizar el orden de distribución y la difusión de mensajes a destinos múltiples es otro servicio de transporte. El tipo de servicio se determina cuando se establece la conexión.

El nivel de transporte es un nivel del tipo **origen-destino** o **extremo a extremo**. Es decir, un programa en la máquina origen lleva una conversación con un programa parecido que se encuentra en la máquina destino, utilizando las cabeceras de los mensajes y los mensajes de control. Los protocolos, de los niveles inferiores, son entre cada máquina y su vecino inmediato, y no entre las máquinas origen y destino, las cuales podrían estar separadas por muchos *IMPs* (procesadores de intercambio de mensajes). En la figura 1-6 se ilustra la diferencia entre los niveles 1 a 3, que están encadenados, y los niveles 4 a 7, que son de extremo a extremo.

Algunos *host* son multiproceso, lo cual implica que múltiples conexiones estarán entrando y saliendo de cada uno de ellos. Se necesita alguna forma para decir qué mensaje pertenece a qué conexión. La cabecera de transporte (H4 en la figura 1-5), es un lugar en donde puede colocarse esta información.

1.3.5 Nivel de sesión

El nivel de sesión permite que los usuarios de diferentes máquinas puedan establecer sesiones entre ellos. A través de una sesión se puede llevar a cabo un transporte de datos ordinario, tal como lo hace el nivel de transporte, pero mejorando los servicios que éste proporciona y que se utilizan en algunas aplicaciones. Una sesión podría permitir a un usuario acceder a un sistema de tiempo compartido a distancia, o transferir un archivo entre dos máquinas.

Uno de los servicios del nivel de sesión consiste en gestionar el control del dialogo. Las sesiones permiten que el tráfico vaya en ambas direcciones al mismo tiempo, o bien, en una sola dirección en un momento dado, el nivel de sesión ayudará en el seguimiento de quien tiene el turno.

La administración del testigo es otro de los servicios relacionados con el nivel de sesión. Para el caso de algunos protocolos resulta esencial que ambos lados no traten de realizar la misma operación en el mismo instante. Para manejar estas mismas actividades, el nivel de sesión proporciona testigos que pueden ser intercambiados; solo el extremo en posesión del testigo puede realizar la operación crítica.

Otro de los servicios del nivel de sesión es la sincronización; el nivel de sesión proporciona una forma para insertar puntos de verificación en el flujo de datos, con objeto de que, después de cada error o caída, solamente tengan que repetirse los datos que se encuentran después del último punto de verificación.

1.3.6 Nivel de presentación

El nivel de presentación realiza ciertas funciones que se necesitan bastante a menudo como para buscar una solución general para ellas, más que dejar que cada uno de los usuarios resuelva los problemas. En particular y, a diferencia de los niveles inferiores, que únicamente están interesadas en el movimiento fiable de *bits* de un lugar a otro, el nivel de presentación se ocupa de los aspectos de sintaxis y semántica de la información que se transmite.

Un ejemplo típico de servicio del nivel de presentación es el relacionado con la codificación de datos conforme a lo acordado previamente. La mayor parte de los programas de usuario no intercambian tramas de *bits* binarios aleatorios, sino, más bien, cosas como nombres de personas, datos, cantidades de dinero y facturas. Estos artículos están representados por tramas de caracteres (en código *ASCII o EBCDIC*), números enteros (en complemento a uno o complemento a dos), etcétera. Para posibilitar la comunicación de ordenadores con diferentes representaciones, la estructura de los datos que se va a intercambiar puede definirse en forma abstracta, junto con una norma de codificación que se utilice "en el cable". El trabajo de manejar estas estructuras de datos y la conversión de la representación utilizada en el interior del ordenador a la representación normal de la red, se lleva a cabo a través del nivel de presentación.

El nivel de presentación está relacionado también con otros aspectos de representación de la información. Por ejemplo, la comprensión de datos se puede utilizar aquí para reducir el número de *bits* que tienen que transmitirse, y el concepto de criptografía se necesita utilizar frecuentemente por razones de privacidad y de autenticidad.

1.3.7 Nivel de aplicación

El nivel de aplicación contiene una variedad de protocolos que se necesitan frecuentemente. Por ejemplo, hay centenares de tipos de terminales incompatibles en el mundo. Considerese la situación de un editor orientado a pantalla, de secuencias de escape para insertar y borrar texto, de movimientos de cursor, etc.

Una forma de resolver este problema consiste en definir una terminal virtual de red abstracta, con el que los editores y otros programas pueden ser escritos para trabajar con él. Con objeto de transferir funciones de terminal virtual de una red a una terminal real, se debe escribir un *software* que permita el manejo de cada tipo de terminal. Por ejemplo, cuando el editor mueve el cursor de la terminal virtual al extremo superior izquierdo de la pantalla, dicho *software* deberá emitir la secuencia de comandos apropiados para que la terminal real ubique también su cursor en el lugar indicado. El *software* completo de la terminal virtual se encuentra en el nivel de aplicación.

Otra función del nivel de aplicación es la transferencia de archivos. Distintos sistemas de archivo tienen diferentes convenciones para denominar un archivo, así como diferentes formas para representar las líneas de texto. La transferencia de archivos entre dos sistemas diferentes requiere de la solución de éstas y otras incompatibilidades. Este trabajo, así como el correo electrónico, la entrada de trabajo a distancia, el servicio de directorio y otros servicios de propósito general y específico, también corresponden al nivel de aplicación.

1.3.8 Transmisión de datos en el modelo OSI

En la figura 1-7 se muestra un ejemplo de cómo pueden transmitirse los datos mediante el empleo del modelo *OSI*. El proceso emisor tiene algunos datos que desea enviar al proceso receptor. Este entrega los datos al nivel de aplicación, el cual añade entonces la cabecera de aplicación, *AH* (la cual puede ser nula), a la parte delantera de los mismos y entrega el elemento resultante al nivel de presentación.

El nivel de presentación transforma este elemento de diferentes formas, con la posibilidad de incluir una cabecera en la parte frontal, dando el resultado al nivel de sesión. Es importante observar que el nivel de presentación no sabe qué parte de los datos que le dio el nivel de aplicación, corresponden a *AH*, y cuáles son los que corresponden a los verdaderos datos del usuario.

Este proceso se sigue repitiendo hasta que los datos alcanzan el nivel físico, lugar en donde efectivamente se transmiten a la máquina receptora. En la otra máquina, se van quitando una a una las cabeceras, a medida que los datos se transmiten a los niveles superiores, hasta que finalmente llegan al proceso receptor.

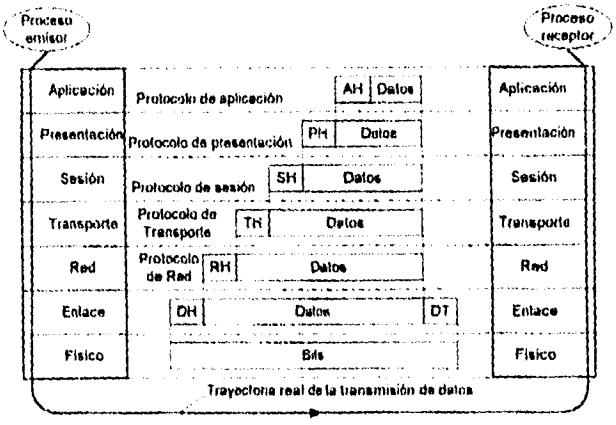


Figura 1-7. Ejemplo de transmisión de datos en el modelo *OSI*

La idea fundamental, a lo largo de este proceso, es que si bien la transmisión efectiva de datos es vertical, como se muestra en la figura 1-7, cada uno de los niveles está programado como si fuera una transmisión horizontal. Cuando el nivel de transporte emisor obtiene por ejemplo, un mensaje del nivel de sesión, le asigna una cabecera de transporte y lo envía al nivel de transporte receptor.

Desde el punto de vista de este nivel, el hecho de que debe realmente entregar el mensaje al nivel de red de su propia máquina es un detalle técnico sin importancia. Una analogía de este hecho es, por ejemplo, cuando un diplomático de habla francesa se dirige a las Naciones Unidas, piensa como si estuviera dirigiéndose personalmente a los otros diplomáticos ahí congregados, el hecho de que le esté hablando sólo a su traductor se ve como un simple detalle técnico.

1.4 USOS DE RED

Algunas aplicaciones típicas de RED son:

- ***Correo electrónico***. Un aspecto importante en una red es la comunicación de textos entre los usuarios, en la actualidad, es común el intercambio de información e ideas, y la comunicación regular entre usuarios a través del correo electrónico (*mail*), esta comunicación no es en tiempo real, ya que el mensaje enviado por un usuario solo será leído por el usuario al que fue destinado.
- ***Intercambio de archivos entre sistemas***. En muchas situaciones es más práctico distribuir la aplicación electrónicamente que enviar discos o cintas magnéticas por correo. La transferencia de archivos a través de una RED provee rapidez y comodidad al usuario remoto, en este caso, al contrario que en el *mail*, la transferencia es efectuada en tiempo real, debido a que se establece una sesión tanto en la máquina remota como en la máquina anfitrión.

- ***Compartir los recursos periféricos***. Uno de los principales objetivos de una red es el de economizar los recursos de cómputo, tales como dispositivos de almacenamiento, impresión, y la capacidad de procesamiento de la propia máquina. Debido a que en muchas ocasiones el costo de un periférico puede exceder el costo de la computadora, en una organización donde hay muchas computadoras personales o estaciones de trabajo, el compartir periféricos optimiza su utilización. Un ejemplo de esto es el uso compartido de una impresora por varios usuarios.
- ***Sesión remota***. Si dos computadoras están conectadas a una RED, se tiene la capacidad de iniciar una sesión en una de ellas, desde la otra (asumiendo que tenemos una cuenta en ambos sistemas). Usualmente es fácil interconectar computadoras que usan una red y proveer una aplicación de sesión remota, de tal manera que una terminal pueda conectarse a varios sistemas diferentes y que por lo tanto no sea necesario realizar un cambio de terminal.
- ***Ejecutar un programa en otra computadora***. En algunas ocasiones, la ejecución de grandes programas o que requieren muchos recursos, provoca una lentitud del sistema para los demás usuarios; al enviar a ejecutar un programa a una máquina remota, se busca una mayor capacidad de procesamiento, almacenamiento y rapidez; lo que a su vez libera de una gran carga de trabajo al sistema anfitrión.
- ***Servidor de archivos***. Por lo regular existen aplicaciones que son empleadas por varios usuarios, y no es necesario que cada uno de ellos posea una copia de los archivos de trabajo, en otras palabras permite efectuar un acceso compartido al mismo archivo.

1.5 REDES PUBLICAS EN UNIX

A lo largo de la historia del sistema *UNIX*, ha aparecido una comunidad de usuarios estrechamente ligada, a través de redes públicas fomentada por la facilidad de comunicación mediante *UUCP* y *mail*. Existen varios grupos de usuarios independientes, de los cuales el principal se denomina *UNIFORM*. Además varias convenciones y encuentros nacionales en EUA, están dedicados a discusiones del sistema *UNIX*, la mayor es la convención *USENIX*. Además, los usuarios del sistema *UNIX* soportan un boletín electrónico mundial y un sistema de correo electrónico que da servicio a miles de máquinas y a cientos de miles de usuarios. Conocida por varios nombres como *NETNEWS*, *USENET*, *READNEWS* o simplemente ***LA RED***, en inglés ***THE NET***, esta red se basa principalmente en el acceso mediante llamada telefónica entre máquinas vecinas, que intercambian mensajes por correo electrónico conocidos como noticias.

La red de noticias tiene pasarelas (*gateways*) que la enlazan con otras redes populares tales como la *INTERNET*, una red de aérea extensa a gran escala y *BITNET*, una red de computadoras para universidades. La red *USENET* (*USEr's NETwork*) es una red mundial, cuyo propósito es el de compartir información entre usuarios de computadoras con sistema *UNIX*. La colección de programas utilizados para compartir información se denomina *netnews*, y los mensajes son conocidos como artículos de noticias. Los artículos que contienen información sobre un tema común se remiten a uno o más grupos de noticias. La mayoría de los sistemas utilizan conexiones telefónicas y *software UUCP* para intercambiar *netnews*. Sin embargo, algunos sistemas utilizan redes existentes y sus protocolos de comunicación, tales como *ARPANET* con *TCP/IP* para intercambio de noticias. Un grupo de localizaciones primarias expiden artículos de noticias entre sí y a muchas otras localizaciones. Las localizaciones individuales también pueden expedir las noticias que reciben a uno o más lugares adicionales. Eventualmente, las noticias alcanzan a todas las maquinas de la *USENET*. Con frecuencia las noticias tienen que viajar a través de muchos sistemas intermedios diferentes hasta alcanzar una máquina en particular.

Grupos de noticias

Los artículos están organizados en grupos de noticias. Existen más de 500 grupos diferentes, organizados en varias categorías principales. Estas categorías son áreas de temas, instituciones o áreas geográficas. Los nombres de todos los grupos de una categoría comienzan por el mismo prefijo. Los prefijos basados en áreas temáticas son:

Prefijo	Contenido
comp	computación
news	netnews y el propio *USENET*
rec	recreaciones
sci	ciencias (sciences)
soc	temas sociales
talk	discusiones
all	estilos de vida alternativos
misc	misceláneas

TABLA 1-1 prefijos basados en áreas temáticas.

Un ejemplo de prefijo utilizado para grupos dentro de una institución particular es *att*, utilizado por *AT&T* para sus grupos internos. Ejemplos de prefijos utilizados por grupos para áreas geográficas específicas son, *nj* para artículos de interés local en New Jersey; *ca* para artículos de interés local en California.

Los grupos de noticias individuales se identifican por una categoría, un punto y un tema, que viene opcionalmente seguido por un punto y un subtema, y así sucesivamente. Por ejemplo, `comp.text` contiene artículos sobre procesamiento de texto por computadora, `comp.unix.questions` contiene artículos que presentan cuestiones sobre el sistema *UNIX* y `rec.art.movies.reviews` contiene artículos que dan reseñas de películas.

Una lista completa de los más de 500 grupos de noticias se encuentra en el archivo `/usr/lib/news/newsgroups` de cualquier host administrador de noticias. La tabla A-1, en el apéndice A incluye algunos de los grupos más populares, otros grupos representativos y grupos de distribución amplia, junto con una descripción de sus temas.

Lectura de netnews

Se pueden utilizar varios programas diferentes para leer las noticias de la red. Entre ellos se incluyen `readnews`, `vnews` (*visual news*) y `rn` (*read news*). Los programas para leer noticias de la red utilizan el archivo `.newsrc` de su directorio propio, que lleva la cuenta de qué artículos se han leído. En particular, el archivo `.newsrc` guarda una lista de los números *ID* de los artículos de cada grupo de noticias que se hayan leído. Cuando se utiliza uno de los programas para leer noticias, solo se muestran los artículos que no se hayan leído, a menos que se suministre una opción a la orden para indicar que se muestren todos los artículos.

CAPITULO 2 Características principales de las *LANs*

Debido a que las redes de área *extensa* (*WAN*) están compuestas de dos o más redes de área local (*LAN*) solo se hará mención al término *LAN*, salvo en aquellas ocasiones donde se mencionen las características propias de una *WAN*.
Existen cuatro características principales que son utilizadas para clasificar las redes de área local:

- ***Medio de Transmisión***
- ***Topología***
- ***Técnicas de señalización***
- ***Métodos de Acceso***

2.1 Medio de Transmisión

El medio de transmisión sirve para conectar los dispositivos en una *LAN*, proporcionando los medios para que las señales de datos viajen de dispositivo en dispositivo. Algunos medios de transmisión pueden soportar más tráfico que otros. La capacidad para transmitir datos no solo es medida por la cantidad de datos que se pueden enviar sobre el medio, sino que tan rápido y que tan lejos pueden viajar los datos sin interferencia o perdida de potencia. Los factores que intervienen en la transmisión de datos son el ancho de banda, interferencia eléctrica y la atenuación.

Ancho de Banda.- El ancho de banda es el rango de frecuencias permitidas en un medio. El ancho de banda puede compararse a los carriles de una carretera, entre más ancho, más tráfico puede transmitir. El ancho de banda puede ser de un solo canal (Banda Base) o puede consistir de muchos canales (Banda Ancha) El ancho de banda es medido por el rango de frecuencias (en *Hertz* o ciclos eléctricos por segundo). Un ancho de banda con numerosos canales de frecuencia permite que más datos sean transmitidos al mismo tiempo, que aquel con un ancho de banda con un solo canal de frecuencia.
Interferencia Eléctrica.- El ruido eléctrico de líneas telefónicas, cables de voltaje, y luces fluorescentes, provocan interferencia eléctrica a los datos que son transmitidos sobre los cables de la red. Forrar los cables de transmisión con un material aislante al ruido eléctrico, reduce los errores provocados por la interferencia eléctrica.
Atenuación.- Las señales se debilitan conforme viajan a través del cable. Así como el debilitamiento de las señales, la interferencia eléctrica externa se incrementa y ocurre un error. Para elevar las señales, se utilizan amplificadores para transmisión analógica (radio frecuencias) y repetidores son usados para la transmisión digital (*bits* electrónicos)

2.1.1 Los principales medios de transmisión
Cable Par Trenzado.- El cable par trenzado es flexible y fácil de instalar. Soporta una velocidad de transmisión de 10 *megabits* por segundo *(Mb/s)* sobre una distancia de hasta 100 metros. Este cable puede ser utilizado para conectar poderosas estaciones de trabajo y largos grupos de red, soporta ambientes que requieren muchos nodos y un tráfico pesado. El cable no es caro, y debido a que se encuentra en muchos edificios, los costos de equipo pueden reducirse. Ya que la transmisión es limitada a una distancia de 100 metros, esta puede incrementarse por medio de amplificadores o repetidores.

Cable Coaxial.- El cable coaxial es más pesado y rígido para manipular que el cable par trenzado. Soporta una velocidad de transmisión libre de errores relativamente de 10 *Mb/s*; es utilizada tipicamente como *backbone* (un cable que conecta pequeñas redes sobre una mayor ruta de transmisión). Algunos tipos de cable coaxial permiten accesos de múltiples canales y son flexibles para ambientes duros que representan aplicaciones de manufactura. El cable coaxial soporta la transmisión de voz, texto y vídeo simultáneos; es muy manejable, lo cual hace fácil de extender la red, por lo mismo, es fácil de instalar y es utilizado en forma frecuente para las *LANs*.

Fibra Óptica.- La fibra óptica soporta una velocidad de transmisión de 100 *Mb/s* (la velocidad de transmisión más rápida de los tres medios descritos). Es inmune al ruido eléctrico y por consiguiente es capaz de proveer un rango bajo de errores en una gran distancia de transmisión. La fibra óptica es ligera y flexible, sin embargo, no es muy manejable, y por lo tanto, es difícil de instalar; también es difícil introducir un nuevo nodo, por lo que la fibra óptica se limita a conexiones punto a punto. Las ventajas de la fibra óptica son que maneja un mayor ancho de banda, existe poca atenuación e interferencia, cubre una mayor distancia (2 *Km.* aproximadamente). La fibra óptica es utilizada cada vez más como *backbone* en campos universitarios, y empresas.

Transmisión por Trayectoria Óptica.- Aunque muchos de los sistemas de comunicación de datos utilizan cables de cobre o fibras para realizar la transmisión, algunos simplemente emplean el aire como un medio para hacerlo. La transmisión de datos por rayos infrarrojos, láser, microondas o radio, no necesita de ningún medio físico, cada una de estas técnicas se adapta a la perfección a ciertas aplicaciones.

Como una alternativa del cable coaxial, en aplicaciones para comunicaciones de larga distancia, se ha utilizado muy ampliamente la transmisión por radio de microondas. Las antenas parabólicas se pueden montar sobre torres para enviar un haz de señales a otra antena que se encuentre a decenas de kilómetros de distancia; cuanto mayor altura tenga la torre, más grande será el alcance que se obtenga. La transmisión mediante microondas se lleva a cabo en una escala de frecuencias que va desde 2 a 40 *GHz*, correspondiendo a longitudes de onda de 15 y 0.75 *cm.* respectivamente. Estas frecuencias se han dividido en bandas de portadoras comunes para aplicaciones de tipo gubernamental, militar y otras.

Comunicación por Satélites.- La comunicación mediante satélite tiene algunas propiedades que la hacen atractiva en algunas aplicaciones. Este tipo de comunicación puede imaginarse como si un enorme repetidor de microondas estuviese localizado en el cielo. Esta constituido por uno o más dispositivos receptor-transmisor, cada uno de los cuales escucha una parte del espectro, amplificando la señal de entrada y, después, la retransmite a otra frecuencia, para evitar los efectos de interferencia con las señales de entrada

Las bandas de 3.7 a 4.3 *GHz* y 5.925 a 6.425 *GHz* se han designado como frecuencias de telecomunicación vía satélite, para flujos de información provenientes del satélite o hacia el satélite, respectivamente. Un satélite típico divide su ancho de banda de 500 *MHz* en aproximadamente una docena de receptores transmisores, cada uno con un ancho de banda de 36 *MHz*. Cada receptor-transmisor puede emplearse para codificar un flujo de información de 50 *Mbps* u 800 canales de voz digitalizada de 64 *kbps*. Una propiedad interesante de la difusión vía satélite es precisamente el poder de difusión; Todas las estaciones incluidas bajo el área del haz, pueden recibir la transmisión, incluso las "estaciones piratas", por consiguiente, se necesita de alguna forma de codificación para mantener el secreto de la información privada.

2.2 Técnicas de Señalización

Existen dos principales técnicas de señalización en las *LAN*: la transmisión de datos digitales en una sola frecuencia y la transmisión de datos en forma analógica dentro de un número de diferentes frecuencias. La técnica de señalización es utilizada dependiendo del ancho de banda del medio, entre más grande el ancho de banda, más grande la capacidad. Muchas redes de área local utilizan técnicas de señalización en banda base; en este caso, cada dato es transmitido como un flujo continuo de dígitos binarios (ceros y unos). Un receptor muestrea los valores de la señal entrante a la misma velocidad utilizada para la transmisión de la señal, pueden existir variaciones menores en cuanto a la sincronía del ritmo entre el transmisor y el receptor.

2.2.1 Código Manchester Diferencial

Un método de codificación de transmisión proporciona una forma al receptor para ajustarse dinámicamente al ritmo correcto, es decir, es un método autosincronizado, el código Manchester diferencial es el método más utilizado en las *LANs*. El método utiliza las siguientes reglas:

- Existe una transición de la señal en la transmisión de cada *bit*.
- Si el *bit* es un cero, entonces existe una transición de la señal al principio de la transmisión del *bit*, si el *bit* es un uno, se mantiene la condición de la señal en el mismo estado.

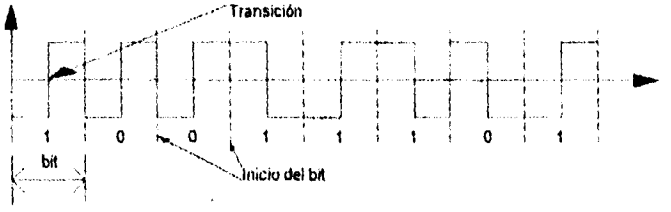


Fig. 2-1. Código Manchester Diferencial

2.3 Métodos de Acceso

Los métodos de acceso son procedimientos utilizados para obtener un buen acceso al ancho de banda para enviar mensajes a otros nodos de la red. Los métodos de acceso pueden clasificarse también en centralizados o distribuidos. Una buena analogía para comparar los métodos de acceso es el flujo y control del tráfico automovilístico. En algunos casos, el tráfico es controlado por un *semáforo* en un modelo DETÉNGASE y SIGA; en otros, los automóviles avanzan y esperan la oportunidad de continuar por turnos. Los métodos centralizados pueden compararse al tráfico dirigido con semáforo, mientras que los métodos distribuidos pueden compararse a tomar turnos o practicar reglas de cortesía en el manejo.

2.3.1 Método de acceso Token Passing

En este método, una canastilla o *token* es enviado de un nodo a otro en una forma continua y circular; un nodo puede acceder a la red solo cuando esta en posesión del *token*; si el nodo no tiene algún mensaje para enviar, mandará el *token* al siguiente nodo, o bien, si el nodo tiene mucho que decir, el *token* llegará al siguiente nodo por tiempo. Este método provee acceso garantizado a cada usuario, y es eficiente bajo cargas pesadas de tráfico; sin embargo, si un *token* se pierde debido a algún tipo de falla en la red, toda la red podría fallar.

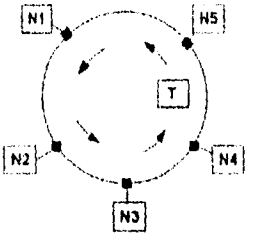


Figura 2-2. Método de Acceso *Token-Passing*

2.3.2 Método de acceso CSMA/CD

El método *CSMA/CD* puede ser definido como escucha, observa, espera, y escucha; un nodo escucha para utilizar la red, si esta ocupada, el nodo espera una oportunidad para tener acceso, enviar el mensaje, y ponerse a escuchar si existe una colisión de mensajes en la red.

- El nodo escucha para determinar si el cable se encuentra en uso (figura 2-3a).
- Cada nodo puede usar la red a un tiempo, si ésta se encuentra libre (figura 2-3b).
- Retrasos en la transmisión pueden provocar colisiones. Si una colisión se detecta, los nodos reintentan enviar el mensaje después de un tiempo aleatorio (figura 2-3c).

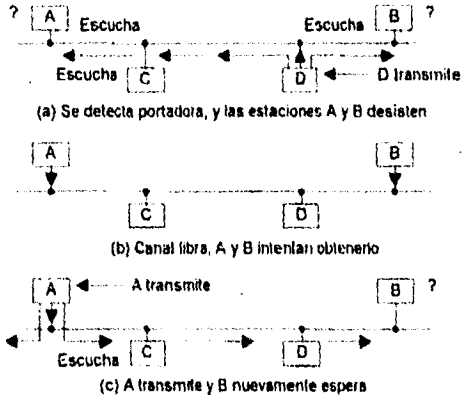


Figura 2-3. Método de Acceso *CSMA/CD*

CSMA/CD es altamente eficiente bajo condiciones en las que existen muchos usuarios, y la probabilidad de que varios usuarios intenten enviar mensajes al mismo tiempo es baja.

2.4 El estándar 802 de IEEE

Debido a la necesidad de estándares en el mercado de las *LAN*, el Instituto de Ingenieros Eléctricos y Electrónicos (*IEEE*) crea el proyecto 802, el cual dirige sus esfuerzos a los niveles físico y de enlace del modelo *OSI*. Los puntos de acceso al servicio (*SAPs*) proporcionan comunicación con los niveles adyacentes.

El nivel físico del modelo de la *IEEE* es similar a su contraparte *OSI*. Sus responsabilidades incluyen codificación y decodificación de la señal, transmisión y recepción de *bits* en forma serial, y proporcionar la conexión física al medio de transmisión, a través de cable coaxial, cable par trenzado o fibra óptica.

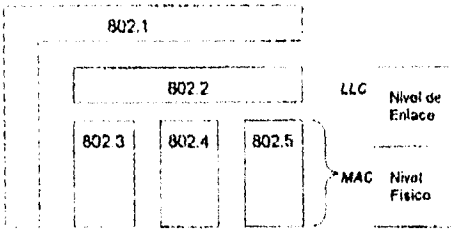


Figura 2-4. Estándar 802 de la IEEE.

Abarcando los niveles *OSI* físico y de enlace, se encuentra el nivel de control de acceso al medio (*MAC*) de la *IEEE*; este nivel controla el acceso al medio de transmisión; además se encuentra dividido en otros estándares, cada uno de estos estándares *MAC* definen una estructura única:

- 802.3.- Detecta portadora, múltiple acceso al *bus* con detección de colisiones (*CSMA/CD*)
- 802.4.- *Token-passing* en *bus*.
- 802.5.- *Token passing* en *anillo*.

2.4.1 El estándar 802.1 y 802.2

El estándar 802.1 de la *IEEE* define las características de interconexión de red y arquitectónicas que son universales para el modelo *LAN*. El nivel más alto de este modelo es el control lógico de enlace (*LLC*), que se encuentra definido por el estándar 802.2 de la *IEEE*.

El Control Lógico de Enlace (LLC)

El control de enlace lógico define dos tipos de servicio:

- Tipo 1: Servicio de conexión sin reconocimiento.
  - + No necesita establecer la conexión de enlace de datos.
  - + Las unidades de datos del protocolo no son reconocidas.
  - + No existe control de flujo en este nivel.
- Tipo 2: Servicio orientado a conexión.
  - + Necesita establecer la conexión de enlace de datos.
  - + Las unidades de datos del protocolo son reconocidas.
  - + Existen control de flujo y recuperación de errores en este nivel.

Estaciones de la clase 1 soportan solo los servicios del tipo 1, y las estaciones de la clase 2 soportan los servicios de los tipos 1 y 2.

2.4.2 El estándar 802.3

Las características principales del estándar 802.3 son:

- Topología de *bus*.
- *CSMA/CD* es el método de acceso.
- Como medio se usa el cable coaxial (50 *ohms*) o cable par trenzado.
- Técnicas de señalización: Banda base, 10 *MHz*, Código Manchester Diferencial (Transmisión Digital)

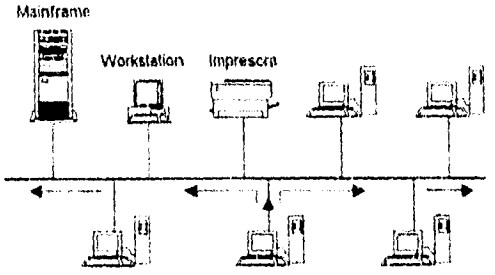


Figura 2-5. Red *Ethernet*.

La ventaja más significativa de utilizar el estándar 802.3, es la gran cantidad de vendedores que diseñan nodos y otros dispositivos asociados a este tipo de red. Muchos establecimientos tienen instalado el cableado con el estándar 802.3; las redes con el estándar 802.3 son llamadas a menudo una red *Ethernet*, pero las redes *Ethernet* no conforman todas las especificaciones del estándar 802.3.

2.4.3 El estándar 802.4

Las características principales del estándar 802.4 son:

- Topología de *bus*.
- Método de acceso: *Token-Passing* con dirección.
- Medio: Cable Coaxial (75 *ohms*).
- Técnicas de señalización:
  - + Banda Base, 10 *MHz*, Código Manchester Diferencial
  - + Banda Ancha, Tres canales (1.5 *MHz*, 6 *MHz*, 12 *MHz*).

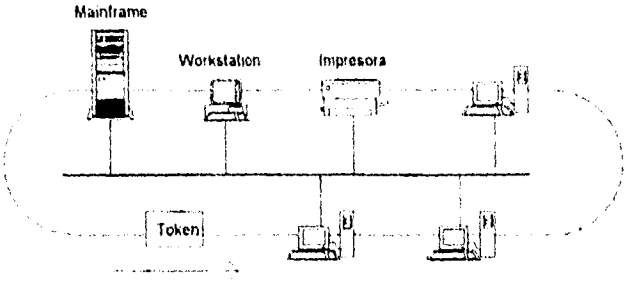


Figura 2-6. El estándar 802.4.

El estándar 802.4 utiliza una dirección en el *token*, para crear un anillo virtual en una topología de *bus*; este estándar es utilizado en fabricas, especialmente en el protocolo de automatización de fabricas (*MAP*).

2.4.4 El estándar 802.5

Las características principales del estándar 802.5 son:

- Topología: Anillo.
- Método de acceso: *Token.Passing*.
- Medio: Cable par trenzado (utilizado más frecuentemente).
- Técnicas de señalización: Banda Base, 1 *MHz* a 10 *MHz*, Código Manchester Diferencial

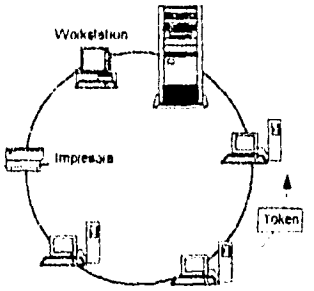


Figura 2-7. El estándar 802.5.

El estándar 802.5 define una red en anillo utilizando el método de acceso *token-passing*; actualmente la conexión de una estación al anillo consiste de un cable par trenzado blindado de 1, 4 o 16 *megabits* por segundo. Las redes que utilizan el estándar 802.5 son nombradas también como redes de anillo *IBM*, ya que el estándar esta basado en la tecnología de *LAN* desarrollada por *IBM*.

2. 5 La Interfaz FDDI

El FDDI es una especificación *ANSI* (*ANSI* X3T9.5) para una red de alta velocidad; sus características principales son:

- Topología: Anillo doble.
- Método de Acceso: *Token-Passing*.
- Medio: Cable de fibra óptica.
- Técnicas de señalización: Banda base, 100 *MHz*, Código NRZI.

Una red *FDDI* consiste de dos anillos de fibra óptica referidos como el anillo primario y secundario; el anillo primario puede ser utilizado para el tráfico de datos, el segundo anillo queda como un anillo de reserva, o ambos anillos pueden utilizarse para el tráfico de datos simultáneamente; los datos en los dos anillos fluyen en direcciones opuestas.

Cuando ocurre una falla en uno de los anillos, las interfaces de red en cada lado de la falla reconfiguran los dos anillos dentro de uno solo, aislando la sección con falla; este provee un grado de tolerancia a fallas en la red. Sus principales aplicaciones son:

- Como *backbone*
- Cualquier aplicación donde la alta velocidad de transferencia para grandes cantidades de información es requerida.
- Cualquier aplicación donde la seguridad y un alto grado de tolerancia a fallas es requerido.

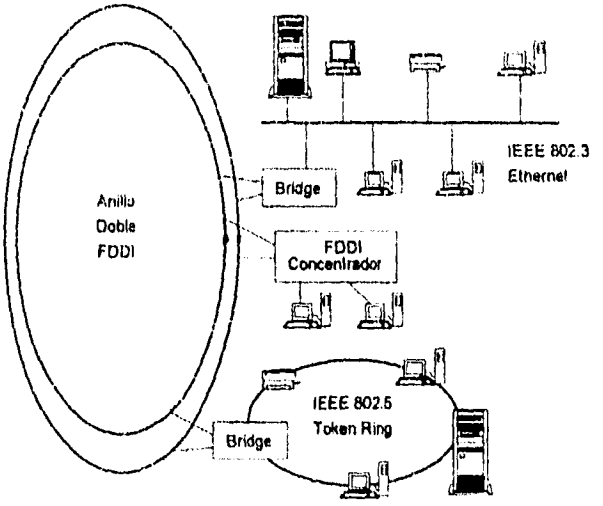


Figura 2-8. Red FDDI

2.6 Implantación en el nivel físico.

La implantación del nivel físico es realizada en dos partes: la señalización física (*PLS*) y la unidad de enlace al medio (*MAU*). La señalización física (*PLS*) se encuentra en la tarjeta de interfaz; esta se conecta al cable de la unidad de enlace de interfaz (*AUI*), y de ahí a la unidad de enlace al medio (MAU); adicionalmente, existe un adaptador de medio, de acuerdo al tipo de medio utilizado (MDI).

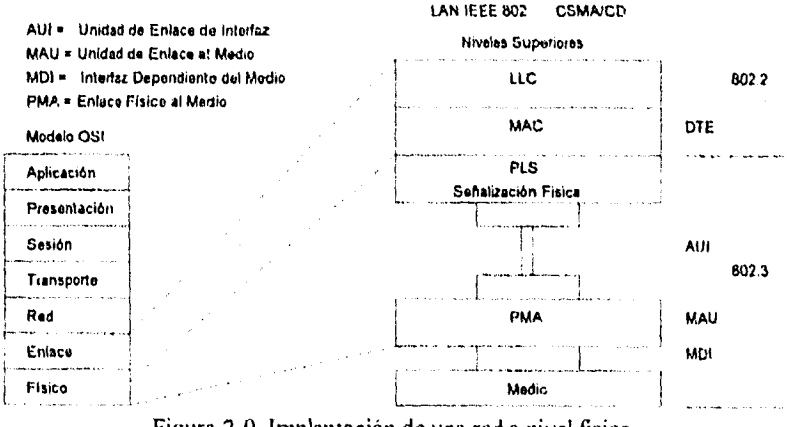


Figura 2-9. Implantación de una red a nivel físico.

2.6.1 MAU y AUI

La unidad de enlace al medio funciona como el vehículo de acceso al medio. La *MAU* transmite y recibe datos del medio, intercambiando señales de control y de datos con la tarjeta de interfaz. Las funciones de control son detecciones de colisiones conocidas como *jabber* y *heartbeat*.

- La detección *Jabber* prevé que un nodo monopolice el uso de la red al terminar una transmisión excesiva, el *MAU* se auto-aisla del medio y regresa una señal de detección de colisión al *DTE*, después de la detección, el *MAU* necesita ser restablecido. Esta detección es un servicio adicional del estándar 802.3, el cual no existe en la *Ethernet*.
- La detección *Heartbeat* prueba la señal con una señal de calidad (*SQE*) enviado del *MAU* al *DTE* después de cada paquete transmitido;

El cable de la unidad de enlace de interfase conecta el *MAU* al equipo terminal de datos; este consiste de cuatro pares de cables par trenzado blindado, con una longitud de máxima de 50 metros, terminado por un conector *D* con 15 *pines*. Algunos *MAU* son autoalimentables y no utilizan el voltaje *DC* que viene del equipo terminal de datos.

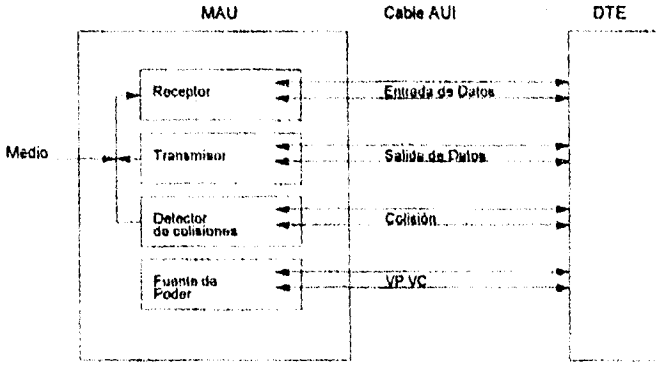


Figura 2-10. Enlace del *MAU* al *DTE*

Existen dos formas de implantar la detección de colisiones en el *MAU*:

- ***Detección basada en recepción***.- Permite la detección de una colisión de una estación que no transmite, pero depende de la resistencia del cable, ya que detecta una colisión si el nivel de *DC* en el cable aumenta.

- ***Detección basada en transmisión***.- Es la detección más confiable y la utilizada más frecuentemente. Este método transmite y compara lo que escucha con lo que transmitió, y si es diferente, entonces hubo una colisión.

No importando el método utilizado, cuando el *MAU* detecta una colisión envía una señal de colisión al *PLS* a través de la conexión *AUI*.

2.6.2 Las funciones de la señalización física

La señalización física (*PLS*) se encuentra en la tarjeta de interfaz del *DTE*. Este es responsable de la codificación de la señal (código Manchester) y el control del método de acceso al medio *CSMA/CD*; este incluye escuchar al medio para la detección de la señal portadora, y algunos servicios como la señal de interferencia, y el algoritmo de espera para el manejo de colisiones.

Método de Acceso CSMA/CD
El método de acceso al medio utiliza dos valores de tiempo:

- El intervalo de tiempo entre paquetes (*IPG*), que es de 9.6 microsegundos, y es el tiempo que el *DTE* debe esperar después de observar una portadora y antes de que pueda transmitir.

- El tiempo de ranurado, es el tiempo que toma la señal para viajar de un extremo a otro en una longitud máxima del segmento. El tiempo de ranurado para un segmento de 512 *bits* (185 *mts*) es de 51.2 microsegundos.

Cuando el medio fue adquirido (esto es que no hubo una colisión durante una transmisión de 51.2 microsegundos), la estación no necesita escuchar buscando una nueva colisión por el resto del la transmisión.

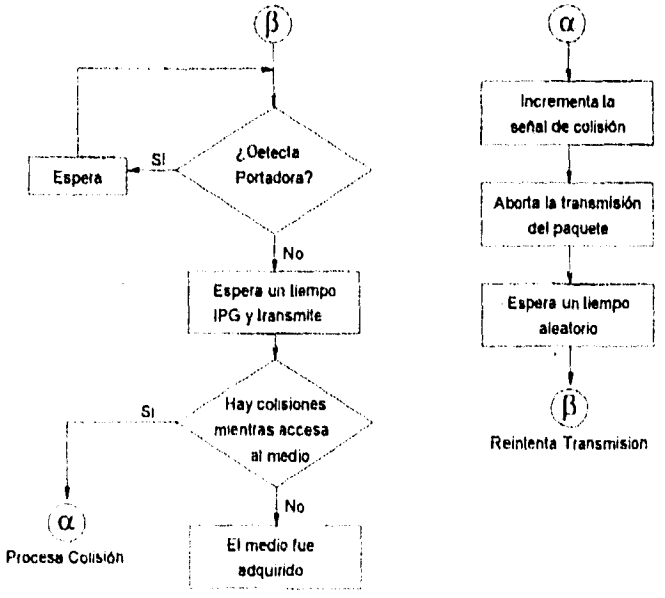


Figura 2-11 Algoritmo del método *CSMA/CD*

Si ocurre una colisión, cada estación continua transmitiendo por 3.2 microsegundos, esta es la señal de interferencia, la cual incrementa la colisión; entonces la estación detiene la transmisión. A continuación, cada estación espera un tiempo aleatorio antes de reintentar la transmisión; este tiempo de espera es calculado utilizando la siguiente condición:

$0 <$ tiempo de espera $< 2^{*}k^{*}$tiempo de ranurado.

en donde k es el número de reintento.

2.7 CONEXIÓN DE LA DIEEC A RED UNAM

2.7.1 antecedentes

A medida que las computadoras se convierten en instrumentos de trabajo indispensables en casi todas las actividades, las terminales independientes dejarán de ser la solución para muchos usuarios que necesitan intercambio de información y optimización de recursos. Además la falta de comunicación directa entre usuarios provoca que los mensajes sean traspapelados, que nunca lleguen a su destino, que tarden mucho tiempo, que sean leídos por *n* usuarios antes de llegar al usuario final o en el peor de los casos que el mensaje con la idea original sea modificada.

Así mismo la distribución de un texto, programas, o una aplicación resulta muy costosa a través de discos flexibles, por lo tanto origina grandes gastos, lentitud e incomodidad para un usuario que se encuentre en un lugar remoto. En estos casos se hace indispensable utilizar un enlace de computadoras, para aprovechar todas las capacidades de cómputo al intercambiar y compartir sus recursos.

Y de esta forma los sistemas pequeños o con menor capacidad, pueden utilizar los recursos y el potencial de procesamiento de los sistemas más grandes, mientras que estos pueden descargar aquellas aplicaciones que se manejan mejor en otras computadoras; así, cualquier recurso de computación agregado a la red se convierte en un elemento positivo para esta. Red UNAM es el proyecto más ambicioso que la UNAM ha desarrollado para la transmisión de datos entre las facultades, institutos, centros de difusión, coordinaciones y demás dependencias que la conforman. A ella también se encuentran conectadas otras instituciones educativas, de investigación y comerciales.

Con el funcionamiento de esta red, se persigue:

- Acercar los bancos de información y otras fuentes de conocimiento a todo estudiante, personal académico y administrativo, y en general, a todo aquel que así lo requiera.
- Promover el intercambio de ideas, pensamientos y opiniones que enriquezcan a las instituciones e individuos.
- Apoyar el crecimiento de la UNAM y de México, brindando una opción tangible para el libre tránsito de información entre las diversas instituciones generadoras y transformadoras de conocimientos del país y del mundo.

Para la *DIECC*, por ser una institución dentro de la UNAM, y por sus características de docencia, era necesaria y prioritaria su conexión a red UNAM.

2.7.2 Necesidades

- Optimizar computadoras y periféricos
- Comunicar los edificios *DIME* y *DIEEC* a través de una red, con un bajo costo y gran funcionalidad.
- Solucionar el problema de correo electrónico, transmisión y recepción de datos de la *DIEEC* desde una *PC* a cualquier parte de la red.
- Acceder a recursos integrados a Red UNAM.
- Ofrecer diversos servicios a la comunidad (estudiantes y académicos)
  - *Gopher*
  - *WWW*
  - *FTP* anónimos
  - sesiones remotas
  - *Netscape*
  - *Mosaic*
  - Etcétera

2.7.3 Propuesta
Instalación de una red bajo una plataforma *UNIX* con sistema operativo *SCO*, con *TCP/IP*, *TELNET*, *FTP*, y *UUCP*.

2.7.4 Fundamentos
La estructura principal de Red UNAM es un anillo de *FDDI* (una fibra óptica activa y una de respaldo que pueden transportar información hasta 100 *Mbps*) que enlaza a 5 enrutadores principales. Conectadas a ellos se encuentran las redes locales de cada dependencia: las que se encuentran en el campus de C.U. son enlazadas por fibra óptica. Aquellas que se hallan fuera de él, se comunican con Red UNAM a través de alguno de los siguientes medios:

1. Dentro del área metropolitana
   - Radio modem
   - Líneas conmutadas o privadas
   - Microondas
   - *RDI*

2. Resto de la República Mexicana
   - RDI
   - Enlaces satelitales

En las redes de la UNAM las topologías más empleadas son variantes de *Ethernet*:
1. Se tienen las redes tipo estrella, (conocida también como red de par trenzado pues éste es el medio físico con el que se construyen). Es posible encontrar este tipo de redes complementadas con redes verticales de coaxial grueso en edificios de varios pisos.
2. Las más empleadas son la de coaxial delgado, aunque su uso empieza a decaer debido a sus desventajas frente al par trenzado. Las redes de *Token Ring* se encuentran en franca desaparición.

Una red de datos de tales características requiere un protocolo de comunicaciones de tal forma que:
- Permita la conexión transparente entre diferentes clases de computadoras: *PC's*, *mainframes*, sistemas *UNIX*, *MAC's*, etc., así como pueda convivir con sistemas operativos de red que se utilizarán en las redes locales.
- Sea fácil de configurar y requiera pocos ajustes de acuerdo al crecimiento de la red.
- Sea altamente confiable bajo cualquier condición operativa y en caso necesario, cuente con herramientas poderosas para la corrección de errores. También deberá brindar al administrador facilidades para el monitoreo y mantenimiento preventivo del funcionamiento de la red.

La *suite* de protocolos *TCP/IP* se perfila como la solución natural a esta lista de requisitos. Es además, el protocolo de facto para la comunicación en *Internet*. Sobre él pueden instalarse sistemas operativos de red tales como *Windows NT* y sus variantes, *LAN Manager*, *Lantastic*, etc., así como *NetWare*.

El sistema *UNIX* permite a un usuario invocar a un máquina remota desde una terminal, sobre líneas telefónicas a un bajo costo. Esto hace que el sistema sea ideal para aplicaciones de *servidor* en las cuales la interfaz de usuario no está asociada con una máquina *UNIX* centralizada, sino que se ejecuta directamente sobre una terminal *inteligente* remota. Proporcionando un entorno moderno de arquitectura.

Para los usuarios que les agrada trabajar en ambiente gráfico, *UNIX* a través del *software X Windows* permite el uso de pantallas gráficas (ventanas) muy similar al *Windows* de *Microsoft*. El sistema *UNIX* es un sistema abierto, por lo tanto, va en contra de la tendencia de *ocultar* o hacer *invisible* el sistema a los usuarios, ya que permite que a base de pequeñas herramientas se logren construir otras más poderosas; con los sistemas cerrados, el administrador o los usuarios deben crear esas herramientas a partir de cero.

Mientras que los ambientes de red *Novell*, *Windows NT*, *PC Network*, *Apple Talk*, etcétera, requieren de potentes servidores para funcionar en mínimas condiciones y con algunos usuarios; *UNIX* con la misma configuración ofrece servicio a varios usuarios conectados y trabajando.

Además hay que recordar que mientras estos ambientes solo ofrecen un pequeño paquete de comunicación, *UNIX* ofrece un paquete muy completo de comunicación a través de máquinas con el mismo sistema operativo, así como con otros ambientes como son *DOS*, con máquinas remotas, sistemas con *UUCP*, y además instalando *TCP/IP*, *TELNET* y *FTP*, se obtiene una conexión total a todos los sistemas operativos a través de *INTERNET*.

2.7.5 Selección de la topología para la DIEEC

Después de tomar en cuenta los tres objetivos principales para establecer una topología de red vistos en la sección 1.1, y las características físicas y geográficas que presentan el área donde se pretende implantar la red se determinó que la topología de *bus*, se ajustaba a estas necesidades. La siguiente figura muestra un esquema de la red implantada en el edificio de la *DIME*.

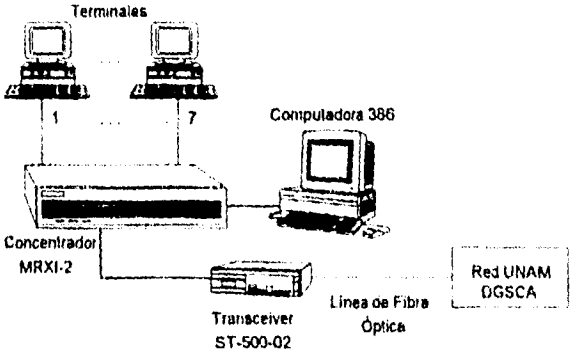


Figura 2-12. Red del departamento de computación DIME.

El siguiente diagrama muestra la red del edificio *DIME* conectada a la del edificio *DIECC* y estas a su vez conectadas a *DGSCA* a través de la fibra óptica.

2.7.5.1 Equipo necesario

- Sistema operativo *DOS* versión 5 o subsecuente.
- Sistema operativo *UNIX SCO*.
- *Software* de comunicación *PC/TCP* o *Windows* 3.11, *TCP/IP* y tarjeta de comunicación *3c509*.

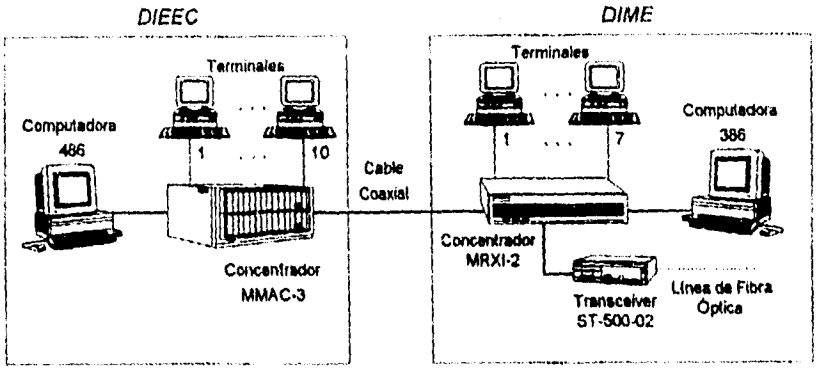


Figura 2-13. Red del departamento de computación *DIME* y *DIEEC*

Para poder realizar la implantación de esta red en la *DIME* se requiere el siguiente material y equipo:

- Microprocesador 386 como servidor con las siguientes características:
  - Mínimo 4 *Mbytes* en memoria *RAM*
  - Más de 150 *Mb* en espacio libre para realizar dos particiones:
    - la primera de 100 *Mb* para *UNIX* como mínimo
    - la segunda el espacio restante para una partición de *DOS*.
  - Drive de alta densidad 3½" o de 5¼".
  - Monitor y teclado.
- Siete Microcomputadores 286 o 386 como terminales.
- Un concentrador *MRXI-2*
- Un *Tranceiver ST-500-02*

Para poder realizar la implantación de esta red en la *DIEEC* se requiere el siguiente material y equipo:

- Microprocesador 486 como servidor con las siguientes características:
  - Mínimo 4 *Mbytes* en memoria *RAM*
  - Más de 150 *Mb* en espacio libre para realizar dos particiones:
    - la primera de 100 *Mb* para *UNIX* como mínimo.
    - la segunda el espacio restante para una partición de *DOS*.
  - Drive de alta densidad 3 ½ o de 5¼ pulgadas.
  - Monitor y teclado.
- Diez Microcomputadores 286 o 386 como terminales.
- Un concentrador *MMAC-3*

Nota: El disco debe ser muy rápido ya que múltiples tareas puede necesitar almacenar temporalmente mientras otra tarea utiliza la memoria real.

2.7.5.2 Conexión de la DIEEC a DGSCA

Antes de realizar la instalación, cableado y conexión de la red , se solicito a *DGSCA* se proporcionara el *Domain Name* de la *DIEEC*, para poder obtenerlo se nos pidió que se hiciera un pequeño *croquis* de la ubicación del edificio, y hasta donde se encontraba el segmento de Fibra óptica. Una vez realizados los tramites el *Domain Name* nos fue entregado y es la siguiente 132.248.59.XX.

Una vez que se obtuvo el *Domain Name* las direcciones de las máquinas fueron puestas cuando el *software* nos lo solicito. También la clave *BITNET*, fue solicitada pero como *DGSCA* nos explico que pronto *BITNET*, iba a ser sustituido y obsoleto; era preferible enviar el correo electrónico desde el *host* (las máquinas servidoras de *UNIX*).

2.7.5.3 Instalacion
Se realizó la instalación del *software UNIX SCO* en los servidores, de la siguiente manera :

- Partición del disco duro (con el comando *fdisk* del *DOS*)
  ```
  c:\> fdisk
  ```

- Instalación del software *PC/TCP* :
  - Se creo un directorio en la partición de *DOS*, con el siguiente nombre ***PCTCP***
    ```
    c:\> md pctcp
    ```
  - Se inserta en el *drive "a"*, el primer disco (*install*) y se cambia de a dicho *drive*
    ```
    c:\> a:
    a:\> install
    ```
    y la máquina ira solicitando disco por disco hasta concluir (1...3). Este *software* será ubicado en el directorio ***PCTCP***
  - Mientras se realiza la instalación del *software* este ira solicitando una serie de datos (nombre del usuario, ubicación del *host*, teléfono, nombre del *host*, nombre del usuario y la dirección correspondiente del sistema).

- Instalación del *software UNIX SCO* .
  - El sistema (la máquina) es reinicializada.
  - Se inserta en el *drive "a"*, el primer disco (*install*)
  - Una vez que la máquina reconoce los recursos con los que cuenta , el *drive "a"* comienza a leer el primer disco y así sucesivamente hasta concluir. Este *software* solicitara que se le proporcione que partición del disco debe usar, una vez indicado esto, esa partición será formateada (no en el formato *DOS*) y los archivos del sistema serán colocados ahi, en diversos directorios los cuales describiremos más adelante.
  - El *software* de comunicación se instala una vez instalado el sistema operativo, desde *root* se ejecuta *admin* y después instalación de *software*.

CAPITULO 3 El enlace Internet

El Protocolo *Internet* está diseñado para su uso en sistemas interconectados de redes de comunicación de computadoras con conmutación de paquetes. El protocolo *INTERNET* estipula la transmisión de bloques de información llamados datagramas, de remitentes a destinatarios, donde los remitentes y destinatarios son *hosts* identificados por una dirección de longitud fija. El protocolo *INTERNET* también estipula la fragmentación y reensamblado de datagramas grandes, si es necesario, para su transmisión a través de redes que manipulan "paquetes pequeños". El protocolo *INTERNET* está limitado específicamente en alcance para proveer las funciones necesarias para suministrar un paquete de *bits* (un datagrama *Internet*) de un remitente a un destinatario sobre un sistema interconectado de redes. No hay mecanismos para aumentar la confiabilidad de la información, control de flujo, secuenciamiento, u otros servicios de extremo-a-extremo, comúnmente encontrados en protocolos *host*-a-*host*. El protocolo *INTERNET* puede capitalizarse en los servicios de las redes soportadas para proveer diversos tipos y calidades de servicio. Este protocolo es llamado por los protocolos *host*-a-*host* en un ambiente *INTERNET*. Este protocolo llama a los protocolos de red local para llevar el datagrama *Internet* al siguiente *gateway* o *host* de destino.

Por ejemplo, un módulo *TCP* llamaría al módulo *Internet* para tomar un segmento *TCP* (incluyendo el encabezado *TCP* y la información de usuario) como la porción de información de un datagrama *Internet*. El módulo de *TCP* provee las direcciones y otros parámetros en el encabezado *Internet* al módulo *Internet* como argumentos de la llamada. El módulo *Internet*, entonces creará un datagrama *Internet* y llamará a la interfaz de red local para transmitir el datagrama *Internet*. El Protocolo de Control de Transmisión (*TCP*) pretende utilizarse como un protocolo *host*-a-*host* altamente confiable entre *hosts*, en redes de comunicación con conmutación de paquetes, y en sistemas interconectados de las mismas redes.

TCP es un protocolo confiable de extremo a extremo, orientado a conexión, diseñado para acomodarse en una jerarquía de protocolos de nivel que soportan aplicaciones multired. El *TCP* proporciona comunicación interproceso confiable entre pares de procesos en computadoras *host* asociadas a distintas redes de comunicación interconectadas. Muy pocas suposiciones están hechas en cuanto a la confiabilidad de los protocolos de comunicación que se encuentran debajo de la capa de *TCP*. *TCP* supone que puede obtener un servicio de datagramas simple, potencialmente, poco confiable de los protocolos inferiores. En principio, el *TCP* debería ser capaz de operar sobre un amplio espectro de sistemas de comunicación, variando desde redes de conexiones alambradas a redes de conmutación de paquetes o redes de conmutación de circuitos.

3.1 EL NIVEL DE RED INTERNET

El protocolo *INTERNET* instrumenta dos funciones básicas: direccionamiento y fragmentación. Los módulos *Internet* utilizan las direcciones llevadas en el encabezado *Internet* para transmitir datagramas *Internet* hacia sus destinatarios. La selección de una ruta para la transmisión se llama encaminado o ruteo. Los módulos *Internet* utilizan campos en el encabezado *Internet* para fragmentar y reunir datagramas *Internet* cuando sea necesario para su transmisión a través de redes que manipulan "paquetes pequeños". El modelo de operación es que un módulo *Internet* reside en cada

host comprometido en la comunicación *Internet* y en cada *gateway* que interconecta redes. Estos módulos comparten reglas para interpretar campos de dirección y para fragmentar y ensamblar datagramas *Internet*. Además, estos módulos (especialmente en *gateways*) tienen procedimientos para tomar decisiones de ruteo y otras funciones. El protocolo *INTERNET* trata cada datagrama *Internet* como una entidad independiente no relacionada a cualquier otro datagrama *Internet*. No hay conexiones o circuitos lógicos (virtuales o de otra manera). El protocolo *INTERNET* utiliza cuatro mecanismos clave al ofrecer su servicio: Tipo de Servicio, Tiempo de Vida, Opciones, y Suma de Control del Encabezado.

- El **Tipo de Servicio** es utilizado para indicar la calidad del servicio deseado. El tipo de servicio es un conjunto abstracto o generalizado de parámetros que caracterizan las posibilidades de servicio provistas en las redes que hacen posible a *Internet*. Este tipo de indicación del servicio va a ser utilizada por *gateways* para seleccionar los parámetros actuales de transmisión para una red particular, la red a ser utilizada para el siguiente salto, o el siguiente *gateway* cuando se encamina un datagrama *Internet*.
- El **Tiempo de Vida** es la indicación de un límite máximo en la vida de un datagrama *Internet*. Este es establecido por el emisor del datagrama y reducido en los puntos a lo largo del itinerario donde este es procesado. Si el tiempo de vida llega a cero antes que el datagrama *Internet* alcance su destino, el datagrama *Internet* es destruido. El tiempo de vida puede ser concebido como un tiempo límite de autodestrucción.
- Las **Opciones** estipulan funciones de control necesarias o útiles en algunas situaciones pero innecesarias para las comunicaciones más comunes. Las opciones proveen marcas de tiempo, seguridad, y encaminado especial.
- La **Suma de Control del Encabezado** es una verificación de que la información utilizada, en el proceso del datagrama *Internet* ha sido transmitida correctamente. La información puede contener errores; si la suma de control del encabezado no concuerda, el datagrama *Internet* será desechado de inmediato por la entidad que detecta el error.

El protocolo *INTERNET* no provee una facilidad de comunicación confiable. No hay reconocimientos ya sea extremo-a-extremo o salto-a-salto. No hay control de error para la información, solamente una suma de control del encabezado. No hay retransmisiones. No hay control de flujo. Los errores detectados pueden ser informados a través del Protocolo de Mensajes de Control *INTERNET* (***ICMP***) que es instrumentado en el módulo *Internet* de protocolo.

3.1.1 Protocolo INTERNET (IP)

El siguiente diagrama ilustra el lugar del protocolo *INTERNET* en la jerarquía de protocolos:

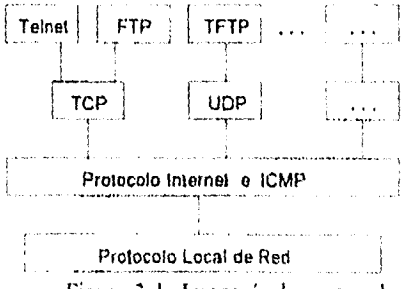


Figura 3-1. Jerarquía de protocolos

3.1.1.1 Modelo de Operación

El modelo de operación para transmitir un datagrama de un programa de aplicación a otro se ilustra en el siguiente escenario:

Supongamos que esta transmisión involucrará un *gateway* intermediario. El programa de aplicación emisor prepara su información y llama a su módulo *Internet* local para enviar esa información como un datagrama y pasa la dirección destinataria y otros parámetros como argumentos de la llamada. El módulo *Internet* prepara un encabezado de datagrama y asocia la información a él. El módulo *Internet* determina una dirección de red local para esta dirección *Internet*, en este caso es la dirección de un *gateway*, y envía el datagrama junto con la dirección de red local a la interfaz de red local.

La interfaz de red local crea un encabezado de red local, e incluye el datagrama al mismo, entonces envía el resultado a través de la red local. El datagrama llega a un *gateway-host* envuelto en el encabezado de red local, la interfaz de red local deja fuera este encabezado, y envía el datagrama al módulo *Internet* en el *gateway*. El módulo *Internet* determina a partir de la dirección *Internet* que el datagrama va a ser enviado a otro *host* en una segunda red.

El módulo *Internet* determina una dirección neta local para el *host* destinatario. Este llama a la interfaz de red local de esa red para enviar el datagrama. Esta interfaz de red local crea un encabezado de red local e incluye el datagrama, enviando el resultado al *host* destinatario. En este *host* destinatario, el datagrama es despojado del encabezado de red local por la interfaz de red local y es entregado al módulo *Internet*. El módulo *Internet* determina que el datagrama es para un programa de aplicación en ese *host*. Pasa la información al programa de aplicación en respuesta a una llamada al sistema, pasando la dirección remitente y otros parámetros como resultados de la llamada.

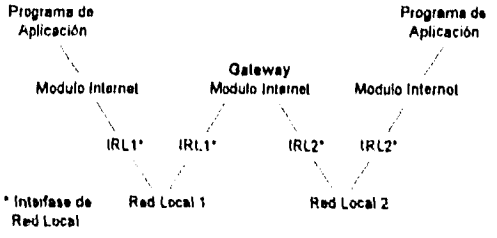


Figura 3-2. Ruta de transmisión

En el apéndice B.1 se muestra el formato del encabezado *Internet*

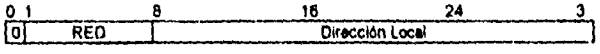
3.1.1.2 Descripción del Funcionamiento

La función o propósito del Protocolo *INTERNET* es mover datagramas a través de un conjunto interconectado de redes. Esto se realiza pasando los datagramas de un módulo *Internet* a otro hasta que el destino es alcanzado. Los módulos *Internet* residen en los *hosts* y *gateways* del sistema *Internet*. Los datagramas son encaminados de un módulo *Internet* a otro, a través de redes individuales basadas en la interpretación de la dirección *Internet*. Así, un mecanismo importante del protocolo *INTERNET* es la dirección *Internet*. En el encaminado de mensajes de un módulo *Internet* a otro, los datagramas pueden necesitar atravesar una red cuyo tamaño máximo de paquete es más pequeño que el tamaño del datagrama. Para vencer esta dificultad, se usa la fragmentación.

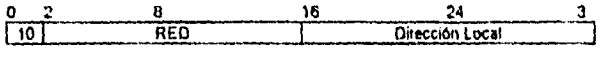
3.1.1.3 Direccionamiento

Una dirección está compuesta de nombres, direcciones, y rutas. Un *Nombre* indica qué buscamos. Una *Dirección* indica donde es tal. Una *Ruta* indica como llegar ahí. El protocolo *INTERNET* trata con direcciones principalmente. Es tarea de los protocolos de nivel superior, (i.e., *host*-a-*host* o una aplicación) hacer el cambio de nombres a direcciones. El módulo *Internet* proyecta las direcciones *Internet* a direcciones de red local. Es tarea de procedimientos del nivel inferior, (i.e., red local o *gateways*) realizar la proyección de direcciones de red local a rutas. Las direcciones tienen una longitud fija de cuatro octetos (32 *bits*). Una dirección comienza con un número de red, seguida por una dirección local (llamada el campo restante). Hay tres formatos o clases de dirección *Internet*:

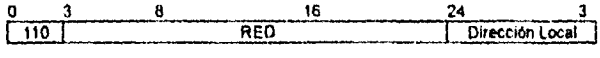
En la ***clase A***, el bit más significativo es cero, los siguientes 7 *bits* son la red, y los últimos 24 *bits* son la dirección local;



En la ***clase B***, los dos *bits* más significativos son "10", los siguientes 14 *bits* son la red y la últimos 16 *bits* son la dirección local;



En la ***clase C***, los tres *bits* más significativos son "110", los siguientes 21 *bits* son la red y la últimos 8 *bits* son la dirección local.



Debe tenerse cuidado en la proyección de direcciones *Internet* a direcciones de red local; un único *host* físico tiene que ser capaz de actuar como si existieran varios *hosts* distintos para la extensión de utilizar varias direcciones *Internet* distintas. Algunos *hosts* también tienen varias interfaces físicas (***multi-homing***). Esto es, tiene que preveerse que un *host* tiene varias interfaces físicas a la red, cada una a su vez tiene diversas direcciones *Internet* lógicas.

3.1.1.4 La fragmentación

La fragmentación de un datagrama *Internet*, es necesaria cuando éste se origina en una red local que permite un tamaño de paquete grande, y tiene que cruzar para alcanzar su destino, una red local que limita a los paquetes a un tamaño más pequeño. Un datagrama *Internet* puede ser marcado como "no fragmentable." Cualquier datagrama *Internet* marcado no será fragmentado por *Internet* bajo ninguna circunstancia. Si el datagrama *Internet* marcado como no fragmentable, no puede ser entregado a su destino sin fragmentarlo, será descartado.

La fragmentación, transmisión y reensamblado a través de una red local la cual es invisible al módulo de protocolo *INTERNET* es llamada fragmentación intranet y puede ser utilizada. La fragmentación *Internet* y el procedimiento de reensamblado necesitan ser capaces de romper un datagrama en un número casi arbitrario de piezas que pueden ser reunidas posteriormente. El receptor de los fragmentos utiliza el campo de identificación para garantizar que fragmentos de diferentes datagramas no son mezclados. El campo de desplazamiento del fragmento dice al receptor la posición de un fragmento en el datagrama original.

El desplazamiento y la longitud del fragmento determina la porción del datagrama original cubierto por este fragmento. La bandera más-fragmentos indica (al ser restablecida) el último fragmento. Estos campos proveen información suficiente para reunir los datagramas. El campo de identificación es utilizado para distinguir los fragmentos de un datagrama de aquellos que pertenecen a otro. El módulo de protocolo, creador de un datagrama *Internet* sitúa en el campo de identificación, un valor que tiene que ser único para el par remitente-destinatario y el tiempo que el datagrama estará activo en el sistema *Internet*. El módulo de protocolo creador de un datagrama completo coloca en la bandera más-fragmentos un cero y en el campo desplazamiento del fragmento un cero.

Para fragmentar un datagrama *Internet* grande, un módulo de protocolo *INTERNET* (por ejemplo, en un *gateway*), crea dos nuevos datagramas *Internet* y copia los contenidos de los campos del encabezado *Internet* del datagrama original en los dos nuevos encabezados *Internet*. La información del datagrama original es dividida en dos porciones dentro de un múltiplo de 8 octetos (64 *bits*) (la segunda porción podría no ser un múltiplo integro de los 8 octetos, pero el primero si tiene que serlo). Llama al número de los bloques de 8 octetos en la primera porción *Número de Bloques de Fragmento* ***NFB*** (*Number of Fragment Blocks*).

La primer porción de la información es situada en el primer datagrama *Internet*, y el campo de longitud total es puesto a la longitud del primer datagrama. El valor de la bandera más-fragmentos toma el valor "uno". La segunda porción de la información es situada en el segundo datagrama *Internet*, y el campo de longitud total toma como valor la longitud del segundo datagrama. La bandera más fragmentos lleva el mismo valor que en el datagrama original. El campo desplazamiento del fragmento del segundo datagrama *Internet* toma el valor del mismo campo en el datagrama original más el *NFB*.

Este procedimiento puede ser generalizado para una división de n-fragmentos, en vez de la división de dos fragmentos descrita anteriormente. Para ensamblar los fragmentos de un datagrama *Internet*, un módulo de protocolo *INTERNET* (por ejemplo en un *host* destinatario) combina todos los datagramas *Internet* que tienen el mismo valor para los cuatro campos: identificación, remitente, destinatario, y protocolo. La combinación es hecha, colocando la porción de información de cada fragmento en la posición relativa indicada por el desplazamiento del fragmento en el encabezado *Internet* del fragmento. El primer fragmento tendrá cero en el desplazamiento del fragmento, y el último fragmento tendrá la bandera más-fragmentos restablecida a cero.

3.1.1.5 Gateways

Los *gateways* instrumentan el protocolo *INTERNET* para reemitir los datagramas entre las redes. Los *gateways* también instrumentan el Protocolo *Gateway* a *Gateway* (***GGP***) para coordinar el encaminado y otra información de control *Internet*. En un *gateway* los protocolos de nivel superior no tienen que ser instrumentados y las funciones *GGP* son añadidas al módulo *IP*.

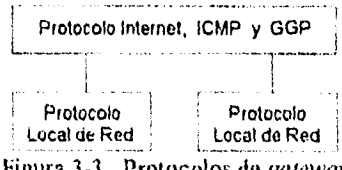


Figura 3-3. Protocolos de *gateway*

3.1.2 Protocolo de mensajes de control INTERNET (ICMP)

El protocolo *INTERNET* (*IP*) es usado para el servicio de datagramas *host* a *host* en un sistema de redes interconectadas llamada *Catenet*. Los dispositivos que conectan las redes son conocidos como *gateways*. Estos se comunican entre ellos para los propósitos de control vía el Protocolo *Gateway* a *Gateway* (*GGP*). Ocasionalmente un *gateway* o un *host* destino se comunican con un *host* fuente, por ejemplo, para reportar un error en el proceso de un datagrama. Para tal propósito es usado el Protocolo de Mensajes de Control *INTERNET* (*Internet Control Message Protocol*, [**ICMP**]). *ICMP*, utiliza el soporte básico de *IP* como si este fuera un protocolo de alto nivel, sin embargo, actualmente *ICMP* es una parte integral de *IP*, y debe ser instrumentado para cada modulo *IP*.

Los mensajes *ICMP* son enviados en situaciones distintas: por ejemplo, cuando un datagrama no puede alcanzar su destino, cuando un *gateway* no tiene memoria intermedia capaz de enviar un datagrama, y por ultimo cuando el *gateway* envía tráfico directamente al *host* a través de una ruta corta. El protocolo *Internet* no esta diseñado para ser totalmente seguro. El propósito de estos mensajes de control es dar un respaldo a los problemas del medio de comunicación, no hacer a *IP* confiable. Aun no hay una garantía fija de que un datagrama pueda ser entregado o que un mensaje de control pueda regresar. Algunos datagramas se pueden llegar a perder sin que se reporte su perdida. Los protocolos de alto nivel que utilizan *IP* deben realizar su propio procedimiento de seguridad, si es necesaria una comunicación segura.

Los mensajes *ICMP* típicos reportan errores en el proceso de los datagramas. Para evitar el regreso infinito de mensajes sobre mensajes, no se envían mensajes *ICMP* de los propios mensajes *ICMP*. Además los mensajes de *ICMP*, solo son enviados cuando ocurren errores en el manejo del fragmento cero de los datagramas fragmentados (el fragmento cero tiene el desplazamiento del fragmento igual a cero). En el apéndice B.2 se muestra el formato del encabezado ICMP, así como varios mensajes.

3.1.3 Protocolo de administración de grupos INTERNET (IGMP)

La multi-emisión *IP* es la transmisión de un datagrama de *IP* para un "grupo *host*", un conjunto de cero o más *hosts* identificados por una dirección de destino de *IP* única. Un datagrama de multi-emisión es entregado a todos los miembros del grupo *host* de destino, con los mismos esfuerzos en confiabilidad, al igual que si se tratara de una emisión-única regular de datagramas *IP*, i.e., no se garantiza que el datagrama llegue intacto a todos los miembros del grupo destinatario o en el mismo orden relativo a otros datagramas.

La membresía de un grupo *host* es dinámica; esto es, los *hosts* pueden salir o unirse a grupos en cualquier momento. No hay restricción en la ubicación o número de miembros en un grupo *host*. Un *host* puede ser un miembro de más de un grupo a la vez. Un *host* no tiene que ser miembro de un grupo para enviarle datagramas.

Un grupo *host* puede ser permanente o temporal. Un grupo permanente tiene una dirección *IP* asignada administrativamente perfectamente conocida. Es la dirección, no la membresía del grupo, que es permanente; en cualquier momento, un grupo permanente puede tener cualquier número de miembros, aún cero o ninguno. Aquellas direcciones *IP* de multi-emisión que no sean reservadas para grupos permanentes, están disponibles para su asignación dinámica a grupos temporales que existen solamente en tanto tengan miembros.

El envío a través de redes de datagramas *IP* de multi-emisión es manejado por "encaminadores de multi-emisión" los cuales pueden ser co-residentes con, o estar separados de los *gateways* de *Internet*. Un *host* transmite un datagrama *IP* de multi-emisión como una red local de multi-emisión que alcanza a todos sus miembros que son vecinos inmediatos del grupo *host* de destino. Si el datagrama tiene un tiempo de vida *IP* mayor que 1, los o el encaminador de multi-emisión asociado a la red local toma la responsabilidad para enviarlo hacia todas las demás redes que tienen miembros del grupo destinatario. En aquellas otras redes miembro que son alcanzables dentro del tiempo de vida *IP*, un encaminador de multi-emisión asociado completa la entrega transmitiendo el datagrama como una multi-emisión local.

Los algoritmos y protocolos utilizados dentro y entre encaminadores de multi-emisión son transparentes para los *hosts*. En el apéndice B.3 se muestra el formato del encabezado IGMP.

3.1.3.1 Niveles de Conformidad

Hay tres niveles de conformidad en esta especificación.

- Nivel 0: Ningún apoyo para la multi-emisión *IP*

  En este momento, no existe algún requisito para que todas las instrumentaciones de *IP* soporten la multi-emisión. Los *hosts* de nivel 0 deberán, en general no ser afectados por la actividad de multi-emisión. La única excepción surge en algún tipo de red local, donde la presencia de *hosts* de nivel 1 o 2, pueden provocar una entrega errónea de datagramas *IP* de multi-emisión a los *host* de nivel 0. Tales datagramas pueden ser fácilmente identificados por la presencia de una dirección *IP* ***clase D*** en su campo dirección de destino; y deberían ser descartados silenciosamente por *hosts* que no soportan la multi-emisión *IP*.
- Nivel 1: Soporte para envío, pero no para recepción datagramas *IP* de multi-emisión.

  El nivel 1 permite que un *host* participe de algunos servicios basados en la multi-emisión, como, informe de ubicación o condición, pero no permite que un *host* se una a algún grupo *host*. Una instrumentación de *IP* puede ser actualizada del nivel 0 al nivel 1 muy fácilmente, y con poco código nuevo.
- Nivel 2: Soporte completo para multi-emisión *IP*

  El nivel 2 permite que un *host* se una y deje grupos *host*, así como el envío de datagramas *IP* a grupos *host*. Requiere una instrumentación del Protocolo de Administración de Grupos *Internet* (*IGMP*), extensión del *IP* e interfaces de servicio de red local dentro del *host*.

3.1.3.2 Modelo de una Instrumentación de un Host IP

Las extensiones de multi-emisión a una instrumentación de un *host IP* son especificadas en términos del modelo de niveles ilustrado más adelante. En este modelo, *ICMP* e *IGMP* (para *hosts* de nivel 2) son considerados para ser instrumentados dentro del módulo de *IP*, y la representación de direcciones *IP* a direcciones de red local es considerada para ser responsabilidad de los módulos de red local.

Para proveer el nivel 1 de multi-emisión, una instrumentación de un *host IP* tiene que soportar la transmisión de datagramas *IP* de multi-emisión. Para proveer el nivel 2 de multi-emisión, un *host* tiene que soportar también la recepción de datagramas *IP* de multi-emisión. Para cada servicio, las extensiones se especifican para la interfaz de servicio *IP*, el módulo *IP*, la interfaz de servicio de red local, y un módulo de red local de *Ethernet*.

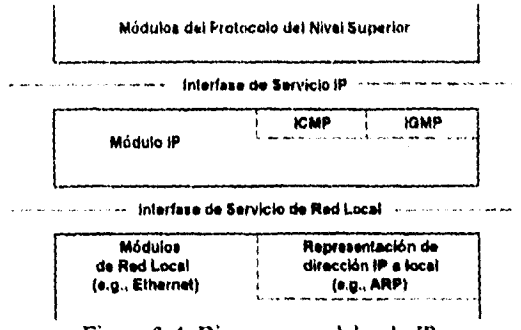


Figura 3-4. Diagrama modular de IP.

3.1.3.3 Direcciones de Grupo Host

Los grupos *host* son identificados por direcciones ***clase D***, i.e., aquellas con "1110" en sus cuatro *bits* más significativos. Las direcciones ***clase E***, i.e., aquellas con "1111" en sus cuatro *bits* más significativos, están reservadas para futuros modos direccionadores. En la notación estándar *INTERNET* de "punteo decimal", el rango de direcciones de grupo *host* va desde 224.0.0.0 a 239.255.255.255. La dirección 224.0.0.0 está protegida para no ser asignada a algún grupo, y 224.0.0.1 es asignada al grupo permanente de todos los *hosts IP* (incluyendo *gateways*). Esta es utilizada para direccionar a todos los *hosts* de multi-emisión en la red conectada directamente. No hay una dirección de multi-emisión (o cualquier otra dirección *IP*) para todos los *hosts* en todo el *Internet*.

3.1.3.4 Descripción del protocolo

Los encaminadores de multiple-emisión envían mensajes de CONSULTA, para descubrir qué grupos *host* tienen miembros en sus redes locales asociadas. Las consultas son direccionadas al grupo de todos los *host* (dirección 224.0.0.1) y llevan un tiempo de vida *IP* igual a uno. Los *hosts* responden a la CONSULTA generando INFORMES; reportando cada grupo *host* al cual ellos pertenecen en la interfaz de red, desde la cual fue recibida la consulta. Para evitar redundancias en los INFORMES simultáneos, así como la reducción del número total de reportes transmitidos, son usadas dos técnicas:

1. Cuando un *host* recibe una CONSULTA, en vez de enviar un reporte inmediatamente, activa un contador de retraso para cada uno de los miembros del grupo en esa interfaz de red, en la que se está desarrollando la CONSULTA. Cada contador es puesto con un valor aleatorio diferente, entre cero y D segundos. Cuando un contador expira, el reporte es generando para el correspondiente grupo *host*. Esto es, los reportes son enviados con un intervalo de D segundos, para evitar que todos se envíen a la vez.
2. Un INFORME es enviado con una dirección destino *IP*, semejante a la del grupo *hosts* que esta informando, con un tiempo de vida *IP* de uno, de tal forma que los otros miembros del mismo grupo en la misma red, pueden alcanzar a oír este reporte. Si un *host* escucha un reporte para algún grupo al cual pertenezca en esa red, el *host* detiene su propio contador para ese grupo y no genera un INFORME para ese grupo. Esto es, en un caso normal, solo un INFORME será generado por cada grupo presente en la red, por el *host* miembro, cuyo contador de retraso expire primero. Se observa que los encaminadores de multiple-emisión, reciben todos los datagramas *IP* de multiple-emisión, por esto no es necesario ser direccionado explícitamente. También se nota que los encaminadores no necesitan saber que *host* pertenece a que grupo, a menos que un *host* pertenezca a un grupo de una red particular.

Existen dos excepciones para las técnicas descritas anteriormente:

1. Si un contador de retraso esta corriendo para un miembro del grupo, cuando una CONSULTA es recibida, entonces ese contador no es reinicializado a un nuevo valor aleatorio, dejándolo correr con su valor actual.
2. El contador de retardo, nunca es inicializado, por un miembro del *host* en el grupo de todos los *hosts* (224.0.0.1) y ese miembro nunca es reportado.

Si un *host* usa un número generador pseudoaleatorio, para calcular los retrasos de los informes; una de las direcciones *IP* individuales, propias del *host* deberá ser usada como parte de la semilla para el generador, para reducir la posibilidad de que varios *hosts* generen la misma secuencia de retrasos. Un *host* deberá confirmar que un INFORME recibido tiene la misma dirección *IP* de grupo *host* en su campo de destino *IP*. Y en su campo de dirección de grupo *IGMP* , para asegurar que su propio INFORME, no sea cancelado por un informe erróneo recibido. Un *host* deberá descartar discretamente cualquier mensaje *IGMP* diferente a una CONSULTA o un INFORME.

Los encauzadores de multiple-emisión envían CONSULTAS periódicamente, para refrescar su conocimiento de los miembros presentes en una red en particular. Si no hay INFORMES recibidos para un grupo en particular, después de un número de CONSULTAS, el encaminador asumirá que ese grupo no tiene miembros locales y que no es necesario reenviar la multi-emisión originada en un lugar remoto para ese grupo dentro de la red local. Normalmente las CONSULTAS son enviadas en rara ocasión (no más de una por minuto), esto es con el fin, de tener poco tráfico *IGMP* en los *hosts* y las redes. Sin embargo cuando un encauzador de múltiples-emisiones comienza a trabajar, este puede generar varias CONSULTAS con intervalos de tiempo muy pequeños para refrescar su conocimiento de los miembros locales rápidamente.

Cuando un *host* se une a un nuevo grupo, este inmediatamente transmitirá un INFORME para ese grupo; esto lo realiza antes de que llegue una CONSULTA, en caso de que este sea el primer miembro de ese grupo en la red. Para cubrir la posibilidad de que el INFORME inicial pueda ser perdido o dañado, se recomienda que se repita, una o dos veces después de pequeños intervalos. (Una forma sencilla de realizar esto es actuar como si una CONSULTA fuera recibida, solo para este grupo, cambiando el contador aleatorio de retraso del reporte de grupo. El diagrama de Transición de Estados que se muestra posteriormente, ilustra este aprovechamiento) Note que, una red que no presenta encaminadores de multiple-emisión, el único tráfico *IGMP* solo consta de uno o más INFORMES enviados siempre que un *host* se une a un nuevo grupo.

3.1.3.5 Diagrama de Transición de Estados

El comportamiento del *IGMP* es especificado formalmente por el siguiente diagrama de estados de transición. Un *host* puede estar en uno de los tres posibles estados, con respecto a un grupo *host IP* sencillo en una interfaz de red simple:

- *No es miembro*, cuando un *host* no puede pertenecer al grupo en la interfaz. Este es el estado inicial para todos los miembros en todas las interfaces de red, que no requieren almacenarse en el *host*.
- *Miembro retrasado*, cuando un *host* pertenece a un grupo sobre la interfaz y tiene un contador de retraso de informe corriendo para ese miembro.
- *Miembro inactivo*, cuando el *host* pertenece a un grupo sobre la interfaz y no tiene un contador de retraso de informe corriendo para ese miembro.

Hay cinco efectos significativos que pueden causar estados de transiciones *IGMP*:

1. *Unión de Grupo*, ocurre cuando el *host* decide unir el grupo sobre la interfaz. Solo puede ocurrir en el estado no-es-miembro.
2. *Abandono de Grupo*, ocurre cuando el *host* decide dejar el grupo sobre la interfaz . Solamente puede ocurrir en el estado de miembro de retrasado y en el estado de miembro inactivo.
3. *Consulta recibida*, ocurre cuando el *host* recibe un mensaje de CONSULTA *IGMP* valido. El mensaje de CONSULTA debe ser al menos de 8 octetos de longitud; tener una suma de control correcta *IGMP*, y contener la dirección *IP* de destino de 224.0.0.1. Una sola CONSULTA se aplica a todos los miembros en la interfaz de la cual la consulta es recibida. Esta, es ignorada por los miembros en el estado de miembro retrasado o sin miembro.
4. *Reporte recibido*, ocurre cuando el *host* recibe un mensaje de INFORME *IGMP* valido. El mensaje de INFORME debe ser al menos de 8 octetos de longitud; tener una suma de control correcta *IGMP*, y contener la misma dirección de grupo *host IP* en su campo de destino y en su campo de dirección de grupo *IGMP*. Un INFORME se aplica solo a los miembros del grupo identificado por el INFORME, en la interfaz de la cual el informe es recibido. Es ignorada por los miembros en los estados sin miembro o de miembro inactivo.
5. *Contador expirado*, sucede cuando el contador de retardo del INFORME para el grupo en la interfaz expira. Solo ocurre en el estado de miembro retrasado.

Cualquier otro evento, como recibir mensajes *IGMP* inválidos, o mensajes *IGMP* diferentes a la CONSULTA o INFORME son ignorados en cualquier estado.

Hay tres acciones posibles que pueden ser tomadas en respuesta a los eventos arriba mencionados:

1. *Enviar reporte*, para el grupo en la interfaz.
2. *Iniciar contador*, para el grupo en la interfaz, usando un valor aleatorio entre 0 y D segundos.
3. *Detener el contador*, para el grupo en la interfaz.

En la figura 3-5, cada arco de transición de estado es etiquetado con el evento que causa la transición y en paréntesis, algunas acciones hechas durante la transición.

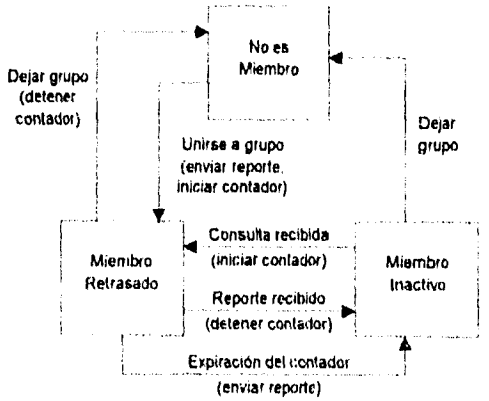


Figura 3-5. Estados de transición.

El grupo de todos los *hosts* (dirección 244.0.0.1) es manejado como un caso especial. El *host* inicia en estado de miembro inactivo para ese grupo en cada interfaz, nunca cambia a otro estado, tampoco envía un INFORME para ese grupo. El máximo retardo en el reporte de D es de 10 SEGUNDOS.

3.2 EL NIVEL DE TRANSPORTE INTERNET

El *TCP* se coloca en una arquitectura de protocolos de nivel justamente sobre el Protocolo *INTERNET* básico, que provee un modo para que el *TCP* pueda enviar y recibir segmentos de longitud variable de información incluida en "sobres" de datagramas *Internet*. El datagrama *Internet* provee un medio para direccionar *TCP*s fuentes y destinos en diferentes redes.

El protocolo *INTERNET* también tiene que ver con cualquier fragmentación o reensamblado requerido por los segmentos *TCP* para lograr su transporte y entrega a través de múltiples redes y *gateways* interconectados. El protocolo *INTERNET* también lleva información al principio, clasificación de la seguridad y como compartir los segmentos *TCP*, de manera que esta información pueda ser comunicada de extremo a extremo a través de redes múltiples.

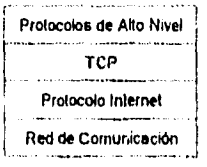


Figura 3-6. Protocolos de Nivel.

Algunos sistemas de computadora estarán conectados a redes a través de computadoras frontera que alojan los niveles de los protocolos *TCP* e *Internet*, así como software específico de red. La especificación de *TCP* describe una interfaz a los protocolos de nivel más altos los cuales parecen ser instrumentados hasta para el caso de frontera, así como un apropiado protocolo final *host* a frontera es instrumentado.

El *TCP* actúa de interfaz a procesos de usuario o aplicación por un lado, y en el otro, a un protocolo de bajo nivel como el Protocolo *Internet*. Esta interfaz consiste de un conjunto de llamadas muy parecidas a las llamadas que un sistema operativo provee a un proceso de aplicación para manipulación de archivos. Por ejemplo, hay llamadas para abrir y cerrar conexiones, para enviar y recibir información en conexiones establecidas. Se espera también que el *TCP* pueda comunicarse asincrónicamente con programas de aplicación.

La interfaz entre el protocolo *TCP* y el protocolo de la capa inferior no es especificada esencialmente, excepto que se espera que exista un mecanismo donde cada uno de los niveles puede pasar la información asincrónicamente hacia el otro. Típicamente, se espera que el protocolo de la capa inferior especifique esta interfaz. *TCP* está diseñado para trabajar en un entorno propio general de redes interconectadas. Generalmente se espera que el protocolo de la capa inferior es el Protocolo *Internet*.

3.2.1 Protocolo de control de transferencia (TCP)

El propósito principal de *TCP*, es proveer un circuito lógico o servicio de conexión confiable y seguro entre pares de procesos. Para proveer este servicio sobre al menos un sistema de comunicación *Internet*, requiere facilidades en las siguientes áreas:

- *Transferencia de Datos Básica*
- *Confiabilidad*
- *Control de Flujo*
- *Multiplexaje*
- *Conexiones*
- *Precedencia y Seguridad*

La operación básica del *TCP* en cada una de estas áreas es descrita en los siguientes párrafos.

Transferencia de Datos Básica:

El *TCP* es capaz de transferir una corriente continua de octetos en cada dirección entre sus usuarios, empaquetando algún número de octetos en segmentos para su transmisión a través del sistema *Internet*. En general, el *TCP* decide cuando bloquear y enviar información de acuerdo a su propia conveniencia.

A veces, los usuarios necesitan asegurarse de que toda la información presentada al *TCP* ha sido transmitida. Para este propósito, una función *PUSH* (empujar) es definida. Para asegurarse que esa información presentada a *TCP* está siendo realmente transmitida, el usuario emisor indica que debería ser "empujado" a través de la linea al usuario receptor. Un *PUSH* provoca al *TCP* para remitir y suministrar prontamente la información suministrada hasta ese punto al receptor. El punto exacto del empuje podría no ser visible al usuario receptor y la función *PUSH* no suministra un marcador limite de registro.

Confiabilidad:

El *TCP* tiene que recuperar información que está dañada, perdida, duplicada, o entregada fuera de orden por el sistema de comunicación *Internet*. Esto se logra asignando un número de secuencia a cada octeto transmitido, y requiriendo un reconocimiento positivo (*ACK - ACKnowledge*) del *TCP* receptor. Si el *ACK* no es recibido dentro de un intervalo de tiempo, la información será retransmitida. En el receptor, los números de secuencia son utilizados para ordenar correctamente, los segmentos que pueden ser recibidos fuera de orden, y para eliminar los duplicados. El daño se maneja añadiendo una suma de control a cada segmento transmitido, revisándolo en el receptor, y descartando los segmentos dañados.

En tanto que los *TCP* continúen funcionando adecuadamente y el sistema *Internet* no se particione completamente, errores de transmisión no afectarán la entrega correcta de la información. *TCP* se recobra de errores del sistema de comunicación *Internet*.

Control de Flujo:

TCP provee un medios para que el receptor gobierne la cantidad de información enviada por el emisor. Esto se logra regresando una "ventana" con cada *ACK*, indicando un rango de números de secuencia aceptables más allá del último segmento exitosamente recibido. La ventana indica unos número de octetos permitido que el emisor puede transmitir antes de recibir un permiso posterior.

Multiplexaje:
Para permitir que muchos procesos dentro de un *Host* único, utilicen facilidades de comunicación de *TCP* simultáneamente, el *TCP* provee un conjunto de direcciones o puertos dentro de cada *host*. Concatenado con las direcciones de red y de *host* del nivel de comunicación *Internet*, se forma un *socket*. Un par de *sockets* únicamente identifica cada conexión. Esto es, un *socket* puede ser utilizado simultáneamente en conexiones múltiples

La asociación de puertos a procesos es manejada independientemente por cada *Host*. Sin embargo, es útil el asociar procesos frecuentemente utilizados (e. g., un *logger* o servicio de tiempo compartido) a *sockets* preparados que son dados a conocer al público. A estos servicios se puede entonces, tener acceso a través de las direcciones conocidas. Estableciendo y aprendiendo las direcciones de puerto de otros procesos, puede involucrar mecanismos más dinámicos.

Conexiones:
Los mecanismos de control de flujo y confiabilidad descritos anteriormente requirieron que *TCP* inicializará y mantuviera cierta información de condición por cada corriente de información. La combinación de esta información, incluyendo *sockets*, números de secuencia, y tamaño de ventana, es llamada una conexión. Cada conexión está especificada únicamente por un par de *sockets* identificando sus dos lados.

Cuando dos procesos desean comunicarse, sus *TCPs* tienen que establecer primero una conexión, (inicializar la información de condición en cada lado). Cuando su comunicación es completa, la conexión es terminada o cerrada para liberar los recursos para otros usos.

Ya que las conexiones tienen que ser establecidas entre *hosts* no confiables y sobre el no confiable sistema de comunicación *Internet*, un mecanismo *handshake* con números de secuencia basados en reloj es utilizado para evitar un inicio erróneo de conexiones.

Precedencia y Seguridad:
Los usuarios de *TCP* puede indicar la seguridad y precedencia de su comunicación. Se prevé utilizar unos valores por omisión, cuando estas características no son necesitadas. En el apéndice B.4 se muestra el formato del encabezado *TCP*.

3.2.1.1 Elementos del Sistema de Interconexión
El entorno de interconexión consiste de *hosts* conectados a redes, las cuales a su vez están interconectadas a través de *gateways*. Se espera que las redes puedan ser, ya sea redes locales (e. g., *ETHERNET*) o redes grandes (e. g., *ARPANET*), pero en cualquier caso, están basadas en tecnología de conmutación de paquetes. Los agentes activos que producen y consumen mensajes son procesados. Diversos niveles de protocolos en las redes, los *gateways*, y los *hosts* sostienen un sistema de comunicación interproceso que provee flujo de datos de dos modos en conexiones lógicas entre puertos de proceso.

El término paquete es utilizado genéricamente para indicar la información de una transacción entre un *host* y su red. El formato de los bloques de información intercambiados dentro de la red generalmente no será del interés de *TCP*.

Los *hosts* son computadoras asociadas a una red, y desde el punto de vista de una red de comunicación, son el origen y destino de los paquetes. Los procesos son vistos como los elementos activos en las computadoras *host* (de acuerdo con la definición bastante común de que un proceso es un programa en ejecución). Aún las terminales y archivos u otros dispositivos de E/S son vistos en comunicación con algún otro a través del uso de procesos. Así, toda la comunicación es vista como una comunicación interproceso.

Ya que un proceso puede necesitar distinguir varias corrientes de comunicación entre si mismo y otro proceso (o procesos), imaginemos que cada proceso puede tener un número de puertos a través de los cuales se comunica con los puertos de otros procesos.

3.2.1.2 Modelo de Operación de TCP

Los procesos transmiten información llamando al *TCP* y pasando *buffers* de información como argumentos. El *TCP* empaqueta la información de estos pedazos en segmentos y llama al módulo *Internet* para transmitir cada segmento al *TCP* de destino. El *TCP* receptor coloca la información de un segmento en los *buffers* del usuario receptor y notifica al usuario receptor. El *TCP* incluye información de control en los segmentos, los cuales se utilizan para garantizar una transmisión de datos ordenada y confiable.

Existe un módulo de protocolo *INTERNET* asociado con cada *TCP* que provee una interfaz a la red local. Este módulo *Internet* empaqueta segmentos *TCP* dentro de datagramas y encauza estos datagramas a un módulo *Internet* de destino o un *gateway* intermediario. Para transmitir el datagrama a través de la red local, se incorpora en un paquete de red local. La conmutación de paquetes puede desempeñar más empaquetamiento, fragmentación, u otras operaciones para lograr la entrega del paquete local al módulo *Internet* de destino.

En un *gateway* entre redes, el datagrama *Internet* es "desenvuelto" de su paquete local y examinado para determinar a través de cual red el datagrama *Internet* deberá viajar después. El datagrama *Internet* es entonces "envuelto" en un paquete local apropiado para la siguiente red y es enviado al siguiente *gateway*, o al destino final.

Un *gateway* tiene permiso para romper un datagrama *Internet* en fragmentos *Internet* más pequeños si esto fuera necesario para su transmisión a través de la siguiente red. Para hacer esto, el *gateway* produce un conjunto de datagramas *Internet*; cada uno lleva un fragmento. Los fragmentos pueden ser rotos en fragmentos más pequeños en los *gateways* posteriores. El formato del fragmento del datagrama *Internet*, es diseñado de modo que el módulo *Internet* de destino, puede reunir esos fragmentos en datagramas *Internet*.

Un módulo *Internet* de destino desenvuelve el segmento del datagrama (después de rearmar el datagrama, si esto es necesario) y lo pasa al *TCP* destinatario.

Este simple modelo de operación, encubre muchos detalles. Una característica importante es el tipo de servicio. Este, provee información al *gateway* (o módulo *Internet*) para ayudarlos a seleccionar los parámetros de servicio a ser utilizados para la travesía en la siguiente red. Incluida en el tipo de servicio, la información se encuentra al principio del datagrama. Los datagramas pueden también llevar información de seguridad, para permitir a los *host* y *gateways* que operen en entornos de seguridad multinivel, para discriminar datagramas adecuadamente por consideraciones de seguridad.

3.2.1.3 Entorno del Host

El *TCP* asume ser un módulo en un sistema operativo. El acceso de usuarios a *TCP* es muy parecido al acceso al sistema de archivos. El *TCP* puede llamar a otras funciones del sistema operativo, por ejemplo, para manipular estructuras de datos. Se pretende que la interfaz actual a la red es controlada por un módulo manejador de dispositivo. El *TCP* no llama al manejador del dispositivo de red directamente, sino más bien, llama al módulo del protocolo de datagrama *Internet*, el cual puede llamar a su vez al manejador del dispositivo.

3.2.1.4 Interfaces

La interfaz *TCP*/usuario suministra llamadas hechas por el usuario en el *TCP* para abrir (*OPEN*) o cerrar (*CLOSE*) una conexión, enviar (*SEND*) o recibir (*RECEIVE*) información, o para obtener la condición (*STATUS*) acerca de una conexión. Estas llamadas son como otras llamadas de programas de usuario en el sistema operativo, por ejemplo, las llamadas para abrir, leer de, y cerrar un archivo.

La interfaz *TCP/INTERNET* provee llamadas para enviar y recibir datagramas dirigidos a módulos *TCP* en *hosts* localizados en cualquier parte del sistema *Internet*. Estas llamadas tienen parámetros para pasar la dirección, tipo de servicio, precedencia, seguridad, y alguna otra información de control.

3.2.1.5 Relación con Otros Protocolos

El siguiente diagrama ilustra el lugar del *TCP* en la jerarquía de protocolos:

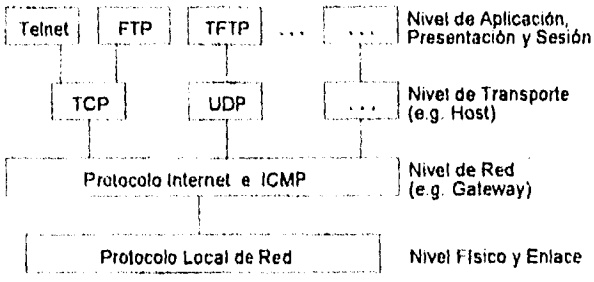


Figura 3-7. Las Relaciones del protocolo

Se espera que el *TCP* sea capaz de soportar eficientemente a los protocolos que se encuentran en un nivel más alto. Debería ser fácil conectar los protocolos de nivel más alto como el *TELNET* de *ARPANET* o el *AUTODIN II THP* al *TCP*.

3.2.1.6 Comunicación Confiable

Una corriente de información enviada en una conexión *TCP* es entregada confiadamente y en orden al destino. La transmisión se hace confiable a través del uso de números de secuencia y reconocimientos. Conceptualmente, cada octeto de información está asignado a un número de secuencia. El número de secuencia del primer octeto de información en un segmento es transmitido con ese segmento y es llamado el número de secuencia del segmento. Los segmentos también llevan un número de reconocimiento, que es el número de secuencia del siguiente octeto de información esperado en la dirección contraria. Cuando el *TCP* transmite un segmento que contiene información, pone una copia en una cola de retransmisión e inicia un cronómetro; cuando el reconocimiento para esa información es recibido, el segmento es eliminado de la cola. Si el reconocimiento no es recibido antes de que el cronómetro termine, el segmento es retransmitido.

Un reconocimiento hecho por *TCP* no garantiza que la información ha sido entregada al usuario final, pero solamente que el *TCP* receptor ha tomado la responsabilidad para hacer eso. Para controlar el flujo de información entre los *TCPs*, un mecanismo de control de flujo es empleado. El *TCP* receptor proporciona una "ventana" al *TCP* emisor. Esta ventana especifica la cantidad de octetos, comenzando con el número de reconocimiento, que el *TCP* receptor esta preparado actualmente para recibir.

3.2.1.7 Establecimiento de Conexión y Aclaración

Para identificar las corrientes de información separadas que un *TCP* puede manipular, el *TCP* provee un identificador de puerto. Ya que los identificadores de puerto son seleccionados independientemente por cada *TCP*, no pueden ser únicos. Para proveer direcciones únicas dentro de cada *TCP*, se concatena la dirección *INTERNET* que identifica el *TCP* con un identificador de puerto para crear un *socket* que será único a lo largo de todas las redes que están conectadas..

Una conexión está completamente especificada por el par de *sockets* en los extremos. Un *socket* local puede participar en muchas conexiones a diferentes *sockets* foráneos. Una conexión puede ser utilizada para llevar información en ambas direcciones, esto es *full duplex*.

Los *TCPs* son libres para asociar puertos con procesos. Sin embargo, varios conceptos básicos son necesarios en cualquier instrumentación. Allí tiene que existir *sockets* perfectamente conocidos, los cuales el *TCP* asocia solamente con los procesos "apropiados" por algunos medios. Se vislumbra que esos procesos pueden "poseer" puertos, y que esos procesos pueden iniciar conexiones solamente en los puertos que poseen. (Los medios para instrumentar esto, es un tratado local, pero se señala un comando de usuario Solicitud de Puerto, o un método para asignar únicamente un grupo de puertos a un proceso dado, e. g., asociando los *bits* más significativos de un nombre de puerto con un proceso dado.)

Una conexión se especifica en la llamada *OPEN* por el puerto local y los argumentos del *socket* foráneo. En recompensa, el *TCP* suministra un nombre de conexión local (pequeño) por medio del cual el usuario se refiere a la conexión en llamadas posteriores. Hay varias cosas que tienen que ser recordadas acerca de una conexión. Para abastecer esta información, imaginamos que hay una estructura de datos llamada ***Bloque de Control de Transmisión*** (***TCB***). Una estrategia de instrumentación tendría el nombre de conexión local como un apuntador al *TCB* para esta conexión. La llamada *OPEN* también especifica si el establecimiento de la conexión es para ser monitoreada activamente, o para estar en espera pasivamente.

Una solicitud de apertura (*OPEN*) pasiva, significa que el proceso quiere aceptar solicitudes de conexión que están entrando, en vez de intentar iniciar una conexión. A menudo, el proceso que solicita una apertura pasiva, aceptará una solicitud de conexión de cualquier llamante. En este caso un *socket* foráneo de ceros es utilizado para denotar un *socket* no especificado. Los *sockets* foráneos no especificados son permitidos solamente en aperturas pasivas.

Un proceso de servicio que desea proveer servicios para otros procesos desconocidos emitiría una solicitud de apertura pasiva con un *socket* foráneo no especificado. Entonces una conexión puede ser hecha con cualquier proceso que solicitó una conexión a este *socket* local. Ayudaría si este *socket* local fuera conocido para ser asociado con este servicio.

Los *sockets* perfectamente conocidos son un mecanismo conveniente para, en primer instancia, asociar una dirección de *socket* con un servicio estándar. Por ejemplo, el proceso "servidor *Telnet*" está asignado permanentemente a un *socket* particular, y otros *sockets* son reservados para los procesos *File Transfer, Remote Job Entry, Text Generator, Echoer*, y *Sink* (los últimos tres son para propósitos de prueba). Una dirección de *socket* podría ser reservado para tener acceso a un servicio *Look-Up*, el cual regresará el *socket* específico en que un servicio creado nuevamente estuviera provisto. El concepto de un *socket* perfectamente conocido es parte de la especificación de *TCP*, pero la asignación de *sockets* para servicios esta fuera de la especificación.

Los procesos pueden emitir aperturas pasivas y esperan a aperturas activas coincidentes de otros procesos, y ser informados por el *TCP* cuando las conexiones han sido establecidas. Dos procesos que emitan aperturas activas hacia el otro al mismo tiempo, serán conectados correctamente. Esta flexibilidad es critica para el apoyo de la computación distribuida en la cual cada componente actúa de forma asincrona con respecto al otro.

Hay dos casos principales para comparar los *sockets*, en las aperturas pasivas locales y en las aperturas activas foráneas. En el primer caso, la apertura pasiva local ha especificado completamente el *socket* foráneo. En este caso, la coincidencia tiene que ser exacta. En el segundo caso, la apertura pasiva local ha abandonado el *socket* foráneo no especificado. En este caso, cualquier *socket* foráneo es aceptable en tanto que concuerde con los *sockets* locales. Otras posibilidades incluyen coincidencias parcialmente restringidas.

Si hay varias aperturas pasivas pendientes (registradas en los *TCBs*) con el mismo *socket* local, una apertura activa foránea será cotejada a un *TCB* con el *socket* foráneo específico en la apertura activa foránea, si tal *TCB* existe, antes de que seleccione un *TCB* con un *socket* foráneo no especificado. Los procedimientos para establecer conexiones utilizan la bandera de control de sincronización (*SYN*) e involucra un intercambio de tres mensajes. Este intercambio ha sido bautizado como un "apretón de manos" de tres mensajes o vías.

Una conexión es iniciada por la cita de un segmento que arribó conteniendo un *SYN* y un espacio de espera *TCB*, cada uno creada por un comando *OPEN* de usuario. La coincidencia de los *sockets* locales y foráneos determina cuando ha iniciado una conexión. La conexión se "establece" cuando los números de secuencia han sido sincronizados en ambas direcciones. La claridad de una conexión también involucra el intercambio de segmentos, llevando en este caso la bandera de control FIN.

3.2.1.8 Comunicación de Datos

La información que fluye en una conexión puede ser concebida como una corriente de octetos. El usuario emisor indica en cada llamada *SEND* si la información en esa llamada (y cualquier llamada anterior) debería ser inmediatamente enviada al usuario receptor al activar la bandera *PUSH*.

Un envío *TCP* está autorizado para reunir información del usuario emisor y para enviar esa información en segmentos de acuerdo a su propia conveniencia, hasta que la función *PUSH* es establecida, entonces tiene que enviar toda la información no enviada. Cuando un *TCP* receptor ve la bandera *PUSH*, no tiene que esperar más información del *TCP* emisor antes de pasar la información al proceso receptor.

No hay una relación necesaria entre las funciones *PUSH* y las fronteras de segmento. La información en cualquier segmento particular puede ser en parte o total, el resultado de una sola llamada *SEND*, o de múltiples llamadas *SEND*. El propósito de la función *PUSH* y la bandera *PUSH* es para empujar información a través del usuario emisor al usuario receptor. No provee un servicio de registro.

Hay un acoplamiento entre la función *PUSH* y el uso de *buffers* de información que cruza la interfaz *TCP*/usuario. Cada vez que una bandera *PUSH* es asociada con información situada en el *buffer* del usuario receptor, el *buffer* es enviado al usuario para procesar, aún si el *buffer* no está lleno. Si la información que llega llena el *buffer* antes de que un *PUSH* sea visto, la información es pasada al usuario en unidades de tamaño de *buffer*.

TCP también provee unos medios, para comunicar al receptor de información que en algún punto, a lo largo de la corriente de información que el receptor esta leyendo actualmente, hay información urgente. *TCP* no intenta definir que hace el usuario específicamente, después de que se notifica la información urgente que esta pendiente, pero la noción general es que, el proceso receptor tomará acción para procesar la información urgente rápidamente.

3.2.1.9 Precedencia y Seguridad

El *TCP* hace uso del campo tipo de servicio del protocolo *INTERNET* y la opción de seguridad para proveer precedencia y seguridad en una base de conexión a usuarios de *TCP*. No todos los módulos *TCP* funcionarán necesariamente en un entorno de seguridad multinivel; alguien puede ser limitado a uso no clasificado solamente, y otros pueden operar solamente a un nivel y compartimento de seguridad. Por consiguiente, algunas instrumentaciones y servicios de *TCP* a usuarios pueden ser limitados a un subconjunto del seguro multinivel.

Los módulos de *TCP* que operen en un entorno de seguridad multinivel, tienen que marcar adecuadamente los segmentos de salida con la seguridad, compartimento, y la precedencia. Tales módulos de *TCP* también tienen que proveer a sus usuarios o sus protocolos de nivel superior como *Telnet* o *THP*, una interfaz para permitirles especificar el nivel de seguridad deseado, compartimento, y la precedencia de conexiones.

3.2.1.10 La terminología

El mantenimiento de una conexión de *TCP* requiere el recordar algunas variables. Estas variables son almacenadas en un registro de conexión llamado un Bloque de Control de Transmisión o ***TCB*** (*Transmission Control Block*). Entre las variables almacenadas en el *TCB*, se encuentran los números de *socket* locales y remotos, la seguridad y precedencia de la conexión, apuntadores para los *buffers* de envío y recepción del usuario, apuntadores a la cola de retransmisión y al segmento actual. Además las variables referentes a los números de secuencia emisores y receptores son almacenados en el *TCB*.

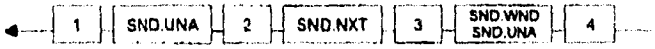
Variables de Secuencia de Envío:

SND.UNA	envía no reconocimiento;
SND.NXT	envía siguiente;
SND.WND	envía ventana;
SND.UP	envía apuntador urgente;
SND.WL1	número de secuencia del segmento utilizado para la última actualización de ventana;
SND.WL2	número de reconocimiento del segmento utilizado para la última actualización de ventana;
ISS	número de secuencia de envío inicial.

Variables de Secuencia de Recepción

RCV.NXT	recibe siguiente;
RCV.WND	recibe ventana;
RCV.UP	recibe apuntador urgente;
IRS	número de secuencia de recepción inicial.

Las siguientes figuras pueden ayudar para relacionar algunas de estas variables con el espacio de secuencia.



1 = viejos números de secuencia que han sido reconocidos
2 = números de secuencia no reconocidos
3 = números de secuencia permitidos para la nueva transmisión de datos
4 = números de secuencia futuros que todavía no están permitidos

Figura 3-8. Espacios de Secuencia de Envío

La ventana enviada es la porción del espacio de secuencia marcado como 3 en la figura 3-8.

1 = viejos números de secuencia que han sido reconocidos
2 = números de secuencia permitidos para la nueva recepción
3 = números de secuencia futuros que todavía no están permitidos

Figura 3-9. Espacios de Secuencia de Recepción

La ventana recibida es la porción del espacio de secuencia marcado como 2 en la figura 3-9. Hay también algunas variables utilizadas frecuentemente que toman sus valores de los campos del segmento actual.

Variables de Segmento Actuales

SEG.SEQ	número de secuencia de segmento
SEG.ACK	número de reconocimiento de segmento
SEG.LEN	longitud de segmento
SEG.WND	ventana de segmento
SEG.UP	apuntador urgente de segmento
SEG.PRC	valor de precedencia de segmento

Una conexión progresa a través de una serie de estados durante su tiempo de vida. Los estados son: *LISTEN, SYN-SENT, SYN-RECEIVED, ESTABLISHED, FIN-WAIT-1, FIN-WAIT-2, CLOSE-WAIT, CLOSING, LAST-ACK, TIME-WAIT*, y el estado ficticio *CLOSED*. *CLOSED* es ficticio, porque representa el estado cuando no hay *TCB*, y por lo tanto, ninguna conexión. Los significados de los estados son:

LISTEN	Representa la espera de una solicitud de conexión de cualquier *TCP* y puerto remotos.
SYN-SENT	Representa la espera de una solicitud de conexión coincidente despues de haber enviado una solicitud de conexión.
SYN-RECEIVED	Representa la espera de una solicitud de reconocimiento de confirmación de conexión, después de haber tanto recibido como enviado la solicitud de conexión.
ESTABLISHED	Representa una conexión abierta, la información recibida puede ser entregada al usuario. El estado normal para la fase de transferencia de datos de la conexión.
FIN-WAIT-1	Representa la espera de una solicitud de terminación de conexión del *TCP* remoto, o un reconocimiento de la solicitud de terminación de conexión previamente enviada.
FIN-WAIT-2	Representa la espera de una solicitud de terminación de conexión del *TCP* remoto.
CLOSE-WAIT	Representa la espera de una solicitud de terminación de conexión del usuario local.
CLOSING	Representa la espera de un reconocimiento de la solicitud de terminación de conexión del *TCP* remoto.
LAST-ACK	representa la espera de un reconocimiento de la solicitud de terminación de conexión previamente enviada al TCP remoto (la cual incluye un reconocimiento de su solicitud de terminación de conexión).
TIME-WAIT	Representa la espera por tiempo suficiente para asegurarse que el *TCP* remoto recibió el reconocimiento de su solicitud de terminación de conexión.
CLOSED	No representa un estado de conexión

TABLA 3-1. Significado de los estados.

Una conexión *TCP* progresa de un estado a otro en respuesta a eventos. Los eventos son llamadas de usuario, *OPEN, SEND, RECEIVE, CLOSE, ABORT,* y *STATUS*; los segmentos entrantes, particularmente aquellos que contienen las banderas *SYN, ACK, RST* y *FIN*; y tiempos de espera.

El diagrama de estado en la figura 3-10 ilustra solamente cambios de estado, junto con los eventos causantes y las acciones resultantes, pero no se especifican ni direcciones, ni condiciones de error, ni acciones que no están relacionadas con los cambios de estado.

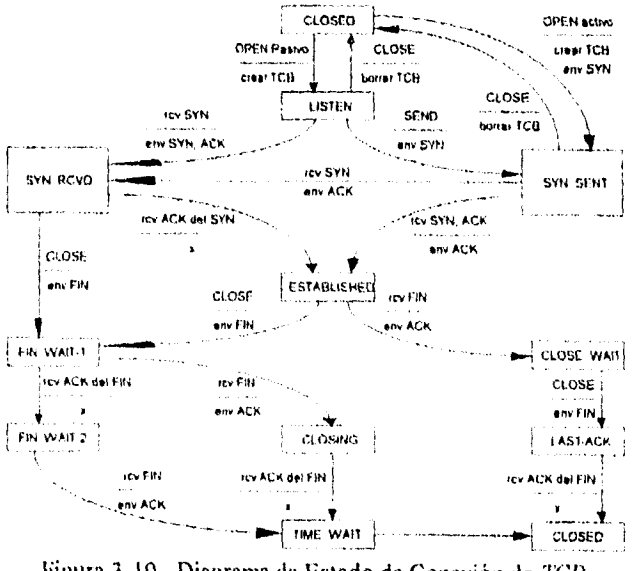


Figura 3-10. Diagrama de Estado de Conexión de *TCP*.

3.2.1.11 Números de Secuencia.
Un aspecto fundamental en el diseño, es que cada octeto de datos que es enviado en una conexión *TCP*, tiene un número de secuencia, ya que cada octeto tiene una secuencia, cada uno de ellos puede ser reconocido. El número de secuencia es acumulativo, de modo que el reconocimiento de un número *X* indica que se han recibido todos los octetos anteriores a *X*. Este mecanismo permite detectar envíos directos duplicados, cuando se presenta la retransmisión. La numeración de los octetos contenidos en un segmento se realiza asignando al primer octeto de datos que sigue al encabezado, el número más chico, y los siguientes octetos se numeran consecutivamente.

El rango real para el número de secuencia es de 0 a 232-1. Ya que el espacio es finito, toda aritmética que trate con los números de secuencia deben desarrollar el módulo 232. Esta aritmética sin signo, preserva la continuidad de los números de secuencia, de tal forma que habiendo llegado al máximo valor, salta a cero repitiendo el ciclo.

3.2.1.12 Selección del Número de secuencia Inicial
El protocolo no pone restricción en una conexión particular que se utiliza una y otra vez. Una conexión es definida por un par de *sockets*. Nuevas instancias de una conexión serán referidas como "encarnaciones" del la conexión. El problema que surge a partir de esto, es "¿Como identifica *TCP* a los segmentos duplicados de encarnaciones previas de la conexión?". Este problema viene a ser aparente si la conexión se abre o cierra en una rápida sucesión, o si la conexión se rompe con perdida de memoria y entonces es restablecida.

Para evitar confusión, se tiene que evitar que segmentos de una encarnación de una conexión sean utilizados mientras los mismos números de secuencia pueden estar presentes aun en la red de una encarnación anterior. Aunque un *TCP* "colisione" y pierda todo conocimiento de los números de secuencia que ha estado usando; cuando nuevas conexiones son creadas, un generador de número de secuencia inicial (*ISN*) es empleado, el cual selecciona un nuevo *ISN* de 32 *bits*. El generador es redondeado a un reloj de 32 *bits* (posiblemente ficticio), del cual su bit menos significativo es incrementado aproximadamente cada 4 microsegundos. Esto es, el *ISN* cumple un ciclo aproximadamente cada 4.55 horas. Ya que se asume que los segmentos estarán en la red no más que el Máximo Tiempo de Vida del Segmento (*MSL*) y debido a que el *MSL* es menor que 4.55 horas, se puede asumir razonablemente que los *ISN*s serán únicos.

Para cada conexión, hay un número de secuencia de envío, y un número de secuencia de recepción. El número de secuencia de envío inicial (*ISS*) es seleccionado por el envío de información de *TCP*; y el número de secuencia de recepción inicial (*IRS*) es aprendido durante el procedimiento de establecimiento de conexión.

Para una conexión que es establecida o inicializada, los dos *TCP*s deben sincronizarse con el número de secuencia inicial del otro. Esto se realiza con un intercambio de segmentos de establecimiento de conexión que transportan un bit de control llamado *SYN* (sincronizar) y el número de secuencia inicial. Como una taquigrafía, los segmentos que transportan el bit *SYN*, también son llamados *SYN*s. He aquí, que la solución requiere un mecanismo apropiado para recoger un número de secuencia inicial y un protocolo de apretón de manos ligeramente involucrado para intercambiar los *ISN*s.

La sincronización requiere que cada lado envíe su propio número de secuencia inicial y reciba una confirmación de él en reconocimiento del otro lado. Cada lado debe también recibir del otro lado, el número de secuencia inicial y enviar una confirmación de reconocimiento.

1. A → B SYN (B, mi número de secuencia es X)
2. A ← B ACK (A, tu número de secuencia es X)
3. A ← B SYN (A, mi número de secuencia es Y)
4. A → B ACK (B, tu número de secuencia es Y)

Debido a que los pasos 2 y 3 pueden ser combinados en un solo mensaje, todo el conjunto es llamado el protocolo de apretón de manos de tres vías (o tres mensajes). El protocolo de tres vías es necesario porque los números de secuencia no están atados a un reloj global en la red, y los *TCP*s pueden tener diferentes mecanismos para recoger los ISNs. El receptor del primer SYN no tiene forma de saber cuando el segmento fue un segmento viejo retrasado o no, a menos que recuerde el último número de secuencia utilizado en la conexión (lo cual no siempre es posible), de modo que tiene que preguntar al emisor para verificar este *SYN*.

3.2.1.13 Cuando permanecer Quieto

Para estar seguro que un *TCP* no puede crear un segmento que transporta un número de secuencia, el cual puede estar duplicado por un segmento viejo que permanece en la red, el *TCP* debe permanecer quieto por un tiempo de vida máximo de un segmento (*MSL*) antes de asignar cualquier número de secuencia a partir del inicio de operaciones, o la recuperación de una quiebra en que la memoria de los números de secuencia en uso fue perdida. Para esta especificación el *MSL* tiene una duración de dos minutos. Esta es una elección de diseño, y puede ser cambiada si la experiencia indica que es necesario. Se observa que si el *TCP* es reinicializado en algún sentido, todavía retiene la memoria de los números de secuencia en uso, de tal forma, que no necesita esperar todo; solo tiene que asegurarse que usa los números de secuencia mayores que aquellos utilizados recientemente.

3.2.1.14 Estableciendo una conexión

El protocolo de apretón de manos de tres vías, es el procedimiento utilizado para establecer una conexión. Este procedimiento es iniciado normalmente por un *TCP* y obtiene respuesta del otro *TCP*. El procedimiento también funciona si dos *TCP*s inician el procedimiento simultáneamente. Cuando existe el intento simultáneo, cada *TCP* recibe un segmento *SYN* que no transporta un reconocimiento después que ha enviado un *SYN*. Por supuesto, la llegada de un viejo segmento *SYN* duplicado, potencialmente puede aparentar, al receptor, que un inicio de conexión simultáneo esta en progreso. El uso apropiado de segmentos *reset* pueden eliminar estos casos.

Algunos ejemplos de inicio de conexión siguen a continuación. Aunque estos ejemplos no muestran la sincronización de conexión utilizando segmentos de transporte de datos, esto es perfectamente válido, mientras tanto, el *TCP* receptor no entrega los datos al usuario, hasta que queda claro que los datos son validos (i.e. los datos son almacenados en un *buffer* del receptor hasta que la conexión alcanza el estado ESTABLECIDO). El protocolo de apretón de manos de tres vías reduce la posibilidad de conexiones falsas. Esta es la instrumentación de una compensación de factores entre la memoria y los mensajes para proveer información para este chequeo.

El ejemplo más simple del protocolo de tres vías es mostrado en la figura 3-11. La figura podría interpretarse en la siguiente forma. Cada línea es numerada para propósitos de referencia. Los puntos (...) indican un segmento, el cual aún se encuentra en la red (retrasado o demorado). Unas equis (XXX) indican un segmento el cual esta perdido o es rechazado. Los comentarios aparecen en paréntesis. Los estados *TCP* representan el estado después de la partida o llegada del segmento (cuyo contenido se muestra en el centro de cada línea).

El contenido del segmento es mostrado con abreviaturas, un número de secuencia, banderas de control, y el campo *ACK*. Los campos tales como ventana, direcciones, longitudes y texto se dejaron fuera para ahondar en la claridad.

	TCP A				TCP B
1.	CERRADO				ESCUCHANDO
2.	ENVIO-SYN	→	<SEQ=100><CTL=SYN>	→	SYN-RECIBIDO
3.	ESTABLECIDO	←	<SEQ=300><ACK=101><CTL=SYN,ACK>	←	SYN-RECIBIDO
4.	ESTABLECIDO	→	<SEQ=101><ACK=301><CTL=ACK>	→	ESTABLECIDO
5.	ESTABLECIDO	→	<SEQ=101><ACK=301><CTL=ACK><DATOS>	→	ESTABLECIDO

Figura 3-11. Protocolo de 3 vías para sincronización de conexión.

En la línea 2 de la figura 3-11, el *TCP* A inicia enviando un segmento *SYN* indicando que utilizará números de secuencia, iniciando con el número 100. En la línea 3, el *TCP* B envía un *SYN* y un reconocimiento (*ACK*) del *SYN* recibido del *TCP* A. Obsérvese que el campo de reconocimiento indica que el *TCP* B esta esperando al número de secuencia 101, al reconocer el *SYN* que ocupó el número 100.

En la línea 4, el *TCP* A responde con un segmento vacío que contiene un *ACK* para el *SYN* del *TCP* B; y en la línea 5, el *TCP* A envía algunos datos. Nótese que el número de secuencia del segmento en la línea 5, es el mismo que en la línea 4, debido a que el *ACK* no ocupa un espacio en el número de secuencia (si lo hiciera, entonces se enviaría un reconocimiento [*ACK*] por cada *ACK*). El inicio simultáneo es ligeramente más compleja, como se muestra en la figura 3-12. Cada *TCP* circula del estado CERRADO al ENVIO-*SYN* al *SYN*-RECIBIDO al ESTABLECIDO.

	TCP A				TCP B
1.	CERRADO				CERRADO
2.	ENVIO-SYN	→	<SEQ=100><CTL=SYN>	...	
3.	SYN-RECIBIDO	←	<SEQ=300><CTL=SYN>	←	ENVIO-SYN
4.		...	<SEQ=100><CTL=SYN>	→	SYN-RECIBIDO
5.	SYN-RECIBIDO	→	<SEQ=100><ACK=301><CTL=SYN,ACK>	...	
6.	ESTABLECIDO	←	<SEQ=300><ACK=101><CTL=SYN,ACK>	←	SYN-RECIBIDO
7.		...	<SEQ=101><ACK=301><CTL=ACK>	→	ESTABLECIDO

Figura 3-12. Sincronización de Conexión Simultánea.

La razón principal para el protocolo de 3 vías es prevenir que viejos inicios de conexión duplicados provoquen confusión. Para tratar esto, un mensaje especial de control, *reset*, ha sido ideado. Si el *TCP* receptor esta en un estado no sincronizado (i.e. ENVIO-*SYN*, *SYN*-RECIBIDO), este regresa a ESCUCHANDO al momento de recibir un reset aceptable. Si el *TCP* se encuentra en uno de los estados sincronizados, (ESTABLECIDO, FIN-*WAIT*-1, FIN-*WAIT*-2, CLOSE-*WAIT*, *CLOSING*, *LAST-ACK*, *TIME-WAIT*), aborta la conexión e informa a su usuario. Este ejemplo se discute en conexiones *half - open* (entre abrir)

Como un simple ejemplo de recuperación de viejos duplicados, considere la figura 3-13. En la línea 3, un viejo *SYN* duplicado llega al *TCP* B. El *TCP* B no puede decir que este es un viejo duplicado, de modo que responde a él normalmente (línea 4). El *TCP* A detecta que el campo *ACK* es incorrecto y regresa un *RST* (*reset*) con el campo *SEQ* seleccionado para hacer el segmento creíble.

El *TCP* B, al recibir el *RST*, regresa al estado *LISTEN*. Cuando el *SYN* original finalmente llega (linea 6), la sincronización procede normalmente. Si el *SYN* de la linea 6 llega antes del *RST*, un intercambio más complejo podria ocurrir con *RST*s enviados en ambas direcciones.

	TCP A				TCP B
1.	CERRADO				ESCUCHANDO
2.	ENVIO-SYN	→	<SEQ=100><CTL=SYN>	...	
3.	(Duplicado)	...	<SEQ=90><CTL=SYN>	→	SYN-RECIBIDO
4.	ENVIO-SYN	←	<SEQ=300><ACK=91><CTL=SYN,ACK>	←	SYN-RECIBIDO
5.	ENVIO-SYN	→	<SEQ=91><CTL=RST>	→	ESCUCHANDO
6.		...	<SEQ=100><CTL=SYN>	→	SYN-RECIBIDO
7.	ENVIO-SYN	←	<SEQ=400><ACK=101><CTL=SYN,ACK>	←	SYN-RECIBIDO
8.	ESTABLECIDO	→	<SEQ=101><ACK=401><CTL=ACK>	→	ESTABLECIDO

Figura 3-13 Recuperación de un viejo SYN duplicado.

3.2.1.15 Conexiones entreabiertas y otras anomalías.

Se dice que una conexión establecida es "entre abierta" si uno de los *TCP*s ha cerrado o abortado la conexión en su extremo, sin el consentimiento del otro, o si los dos extremos de la conexión han sido desincronizados debido a una ruptura que da como resultado la pérdida de memoria. Tales conexiones, automáticamente serán restablecidas si un intento es hecho para enviar datos en cualquier dirección. Aún así, las conexiones entreabiertas pueden ser inusuales, y el procedimiento de recuperación está envuelto superficialmente.

Si en el sitio A, la conexión ya no existe, entonces un intento por el usuario en el sitio B para enviar cualquier dato en el, podria resultar en que el *TCP* del sitio B reciba un mensaje de control de reset. Tal mensaje, indica al *TCP* del sitio B, que algo está mal, y que espera que aborte la conexión.

Supóngase que dos procesos de usuario, A y B, están comunicándose entre si cuando ocurre una ruptura, provocando la pérdida de memoria del *TCP* de A. Dependiendo del sistema operativo que soporta el *TCP* de A, es como existe algún mecanismo de recuperación de error. Cuando el *TCP* es levantado nuevamente, es probable que A comience de nuevo desde el principio, o de algún punto de recuperación. Como resultado, probablemente A tratará de ABRIR (*OPEN*) la conexión nuevamente, o tratará de ENVIAR (*SEND*) en la conexión que cree abierta. En el último caso, recibe el mensaje de error "conexión no abierta" del *TCP* local (A en este caso). En un intento para establecer la conexión, el *TCP* de A enviará un segmento conteniendo un *SYN*. Este escenario se refiere al ejemplo mostrado en la figura 3-14. Después de que el *TCP* A quiebra, el usuario intenta reabrir la conexión. Mientras tanto, el *TCP* B piensa que la conexión esta abierta.

	TCP A				TCP B
1.	(Crash)				(envío 300, recibe 100)
2.	CERRADO				ESTABLECIDO
3.	ENVIO-SYN	→	<SEQ=400><CTL=SYN>	→	(¿?)
4.	(¡!)	←	<SEQ=300><ACK=100><CTL=ACK>	←	ESTABLECIDO
5.	ENVIO-SYN	→	<SEQ=100><CTL=RST>	→	(¡Aborta!)
6.	ENVIO-SYN				CERRADO
7.	ENVIO-SYN	→	<SEQ=400><CTL=SYN>	→	

Figura 3-14. Descubrimiento de conexión entreabierta.

Cuando el *SYN* llega en la linea 3, el *TCP* B, que se encuentra en un estado sincronizado, y el segmento entrante afuera de la ventana, responde con un reconocimiento indicando cual es la secuencia que espera (*ACK* 100). Entonces, el *TCP* A observa que ese segmento no reconoce nada de lo que ha enviado, y está desincronizado, enviando un reset (*RST*) ya que ha detectado una conexión entreabierta. *TCP* aborta la conexión en la linea 5. El *TCP* A continuará intentando establecer la conexión; el problema se ve reducido al protocolo de 3 vias de la figura 3-11.

Un caso alternativo interesante, ocurre cuando el *TCP* A quiebra, y el *TCP* B intenta enviar datos en lo que el cree que es una conexión sincronizada. Esto se ilustra en la figura 3-15 . En este caso, los datos que llegan al *TCP* A del *TCP* B (linea 2) son inaceptables debido a que tal conexión no existe, de tal modo que el *TCP* A envia un RST. El RST es válido y el *TCP* B lo procesa y aborta la conexión.

En la figura 3-16, existen dos *TCP*s, A y B, con conexiones pasivas esperando un *SYN*. Un viejo duplicado llega al *TCP* B (linea 2) y pone a B en acción.

	TCP A			TCP B
1.	(Crash)			(envio 300, recibe 100)
2.	(¿?)	← <SEQ=300><ACK=100><DATA=10><CTL=ACK>	←	ESTABLECIDO
3.		→ <SEQ=100><CTL=RST>	→	(¡Aborta!)

Figura 3-15. Extremo activo descubre conexión entreabierta.

Un *SYN-ACK* es regresado (linea 3) y ocasiona que el *TCP* A genere un *RST* (el *ACK* de la linea 3 no es aceptable). El *TCP* B acepta el reset y regresa a su estado pasivo de escucha (*LISTEN*).

	TCP A			TCP B
1.	ESCUCHANDO			ESCUCHANDO
2.		... <SEQ=300><CTL=SYN>	→	SYN-RECIBIDO
3.	(¿?)	← <SEQ=100><ACK=301><CTL=SYN,ACK>	←	SYN-RECIBIDO
4.		→ <SEQ=301><CTL=RST>	→	(regresa a ESC.)
5.	ESCUCHANDO			ESCUCHANDO

Figura 3-16. Viejo SYN duplicado inicia un Reset en dos *sockets* pasivos.

Una variedad de otros casos son posibles, los cuales son mencionados por las siguientes reglas para la generación y procesamiento del *RST*.

3.2.1.16 Generación del Reset.

Como regla general, el *reset* (*RST*) debe ser enviado siempre que un segmento que llegue, aparentemente no sea para la conexión actual. Un reset no debe ser enviado si no esta convencido de que este es el caso. Existen tres grupos de estados:

1. Si no existe la conexión (CERRADO - *CLOSED*) entonces un reset es enviado en respuesta a cualquier segmento entrante, excepto otro reset. En particular, los *SYNs* direccionados a conexiones no existentes, son rechazados por estos medios. Si el segmento entrante tiene un campo *ACK*, el *reset* toma el número de secuencia del campo *ACK* del segmento, de otro modo el reset tiene un número de secuencia cero y el campo *ACK* es puesto con la suma del número de secuencia y la longitud del segmento entrante. La conexión permanece en el estado *CLOSED*.

2. Si la conexión se encuentra en cualquier estado no sincronizado (*LISTEN, SYN-SENT, SYN-RECEIVED*), y el segmento entrante reconoce algo todavía no enviado (el segmento lleva un *ACK* inaceptable), ó si un segmento entrante tiene un nivel de seguridad, ó compartición, el cual no coincide con el nivel y compartición solicitada para la conexión, entonces el *reset* es enviado.

   Si el *SYN* propio no ha sido reconocido y el nivel de precedencia del segmento entrante es más alto que el nivel de precedencia solicitado, entonces, ya sea que se eleve el nivel de precedencia local (si es permitido por el usuario y el sistema) ó envía un *reset*; ó si el nivel de precedencia del segmento entrante es menor que el nivel solicitado, entonces se continúa como si los niveles coincidieran exactamente (si el *TCP* remoto no puede elevar el nivel de precedencia para hacerlo coincidir con el propio, esto será detectado en el siguiente segmento que envíe, y entonces la conexión será terminada). Si el *SYN* propio ha sido reconocido (quizás en este segmento entrante) el nivel de precedencia del segmento entrante debe coincidir con el nivel local de precedencia, si no, un reset debe ser enviado.
   Si el segmento entrante tiene un campo *ACK*, el *reset* toma el número de secuencia del campo *ACK* del segmento, de otro modo, el resto tiene el número de secuencia cero y el campo *ACK* es puesto con la suma del número de secuencia y longitud de segmento del segmento entrante. La conexión permanece en el mismo estado.

3. Si la conexión se encuentra en un estado sincronizado, (*ESTABLISHED, FIN-WAIT-1, FIN-WAIT-2, CLOSE-WAIT, CLOSING, LAST-ACK, TIME-WAIT*), cualquier segmento inaceptable (fuera del número de secuencia de la ventana, o número de reconocimiento inaceptable) debe sacar solo un segmento de reconocimiento vacío que contiene el número de secuencia de envío actual y un reconocimiento indicando el siguiente número de secuencia esperado en la siguiente recepción, la conexión permanece en el mismo estado.
   Si un segmento entrante tiene un nivel de seguridad, o compartición o precedencia, las cuales no coinciden con los correspondientes nivel de seguridad, compartición y precedencia solicitadas para la conexión, un *reset* es enviado y la conexión cambia al estado CERRADO-*CLOSED*. El reset toma el número de secuencia del campo *ACK* del segmento entrante.

3.2.1.17 Procesamiento del Reset.

En todos los estados, excepto en *SYN-SENT*, todos los segmentos de *reset* (*RST*) son validados al checar sus campos *SEQ*. Un reset es válido si su número de secuencia se encuentra en la ventana. En el estado *SYN-SENT* (un *RST* recibido en respuesta a un *SYN* inicial), el *RST* es aceptado si el campo *ACK* reconoce el *SYN*.

El receptor de un *RST*, primero lo valida, después cambia el estado. Si el receptor estaba en el estado *LISTEN*, lo ignora. Si el receptor estaba en el estado *SYN-RECEIVED* y había estado previamente en el estado *LISTEN*, entonces el receptor regresa al estado *LISTEN*, si no, el receptor aborta la conexión y cambia al estado *CLOSED*. Si el receptor estaba en cualquier otro estado, aborta la conexión e informa al usuario y cambia al estado *CLOSED*

3.2.1.18 Cerrando una conexión

CERRAR-*CLOSE* es una operación que indica "No tengo más datos para enviar". El significado de cerrar una conexión *full-duplex* esta sujeta a interpretación ambigua, ya que puede no ser obvio como tratar el lado receptor de la conexión. Se ha elegido tratar a *CLOSE* en un modo simple. El usuario que CIERRA, puede continuar recibiendo hasta que este dicho que el otro lado también ha cerrado.

Esto es, un programa podría transmitir varios *SENDs* seguidos por un *CLOSE*, y después, continuará RECIBIENDO hasta que detecte que ha fallado una RECEPCIÓN debido a que el otro lado ha cerrado. Se asume que el *TCP* indicará al usuario, aunque las RECEPCIONES no sean sobresalientes de que el otro lado ha cerrado, así el usuario puede terminar su lado satisfactoriamente. *TCP* suministrará todos los *buffers* de *SEND* antes de que la conexión fuera CERRADA, así, un usuario que no espera información de regreso, solo necesita que le informen que la conexión ha CERRADO exitosamente para conocer que toda su información fue recibida en el *TCP* destinatario. Los usuarios tienen que mantener conexiones de lectura que ellos cierran para el envío, hasta que el *TCP* indica que no hay más datos.

Esencialmente existen tres casos:

1. El usuario local inicia el proceso de cierre de conexión.
   En este caso, un segmento FIN puede ser construido y puesto en la salida de la cola de segmento. Más *SENDs* del usuario no serán aceptados por el *TCP*, y este cambia al estado FIN-*WAIT*-1. Los segmentos de RECEPCIÓN son permitidos en este estado. Todos los segmentos que preceden al fin e incluso este, serán retransmitidos hasta su reconocimiento. Cuando el otro *TCP* ha reconocido el FIN, y enviado un FIN de su propiedad, el primer *TCP* puede reconocer (*ACK*) este FIN. Nótese que un *TCP* que recibe un FIN podría reconocer (*ACK*) pero no enviar su propio FIN hasta que su usuario ha cerrado (*CLOSED*) también la conexión.

2. El *TCP* recibe un FIN de la red.
   Si un FIN no solicitado llega procedente de la red, el *TCP* receptor puede reconocerlo (ACK) e indicarle al usuario que la conexión esta cerrándose. El usuario deberá responder con un CLOSE, de tal forma que el *TCP* puede enviar un FIN al otro *TCP* después de enviar la información restante. Entonces, el *TCP* espera hasta que su propio FIN es reconocido, con lo cual elimina la conexión. Si un *ACK* no viene hacia adelante, después del tiempo de espera del usuario, la conexión es abortada y el usuario es informado.

3. Ambos usuarios cierran simultáneamente.
   Un CIERRE-*CLOSE* simultáneo por los usuarios en ambos extremos de la conexión provoca que segmentos FIN sean intercambiados. Cuando todos los segmentos que preceden a los FIN han sido procesados y reconocidos, cada *TCP* puede reconocer (*ACK*) el FIN que ha recibido. Después de que ambos *TCP* reciban los *ACKs*, borrarían la conexión

	TCP A				TCP B
1.	ESTABLECIDO				ESTABLECIDO
2.	(Cerrar)				
	FIN-WAIT-1	→	<SEQ=100><ACK=300><CTL=FIN,ACK>	→	CLOSE-WAIT
3.	FIN-WAIT-2	←	<SEQ=300><ACK=101><CTL=ACK>	←	CLOSE-WAIT
4.					(Cerrar)
	TIME-WAIT	←	<SEQ=300><ACK=101><CTL=FIN,ACK>	←	LAST-ACK
5	TIME-WAIT	→	<SEQ=101><ACK=301><CTL=ACK>	→	CERRADO
6	(2 MSL)				
	CERRADO				

Figura 3-17. Secuencia Normal para cerrar una conexión

	TCP A				TCP B
1.	ESTABLECIDO				ESTABLECIDO
2.	(Cerrar)				(Cerrar)
	FIN-WAIT-1	→	<SEQ=100><ACK=300><CTL=FIN,ACK>	...	FIN-WAIT-1
		←	<SEQ=300><ACK=100><CTL=FIN,ACK>	←	
		...	<SEQ=100><ACK=300><CTL=FIN,ACK>	→	
3.	CERRANDO	→	<SEQ=101><ACK=301><CTL=ACK>	...	CERRANDO
		←	<SEQ=301><ACK=101><CTL=ACK>	←	
		...	<SEQ=101><ACK=301><CTL=ACK>	→	
4.	TIME-WAIT				TIME-WAIT
	(2 MSL)	←	<SEQ=300><ACK=101><CTL=FIN,ACK>	←	(2 MSL)
	CERRADO				CERRADO

Figura 3-18. Secuencia Simultánea para cerrar una conexión.

3.2.1.19 Precedencia y Seguridad

El motivo de la precedencia y seguridad es para que la conexión sea permitida solo entre aquellos puertos con los mismos valores de seguridad y compartición en el nivel de precedencia más alto solicitado por los dos puertos.

Los parámetros de precedencia y seguridad utilizados en *TCP* son los mismos definidos en el protocolo *Internet*. A través de esta especificación de *TCP* el término "seguridad/compartición" pretende indicar los parámetros de seguridad utilizados en *IP*, incluyendo seguridad, compartición, grupo de usuario, y restricciones de manejo.

Un intento de conexión con valores incorrectos de seguridad/compartición o un valor de precedencia más bajo tiene que ser rechazado enviando un *reset*. Rechazar una conexión debido a una precedencia demasiado baja solo ocurre después de que un reconocimiento del *SYN* ha sido recibido.

Es necesario observar que los módulos *TCP* que operan solo con el valor de precedencia por defecto, aun tendrá que verificar la precedencia de los segmentos entrantes y posiblemente aumenten el nivel de precedencia que utilizan en la conexión.

Los parámetros de seguridad pueden ser usados aun dentro de un ambiente no-seguro (los valores podrían indicar información no-clasificada), estos *host*, en los entornos no-seguros tienen que estar preparados para recibir los parámetros de seguridad aunque no necesiten enviarlos.

3.2.2 Protocolo de datagramas de usuario (UDP)

El Protocolo de Datagramas de Usuario (*UDP*) es definido para hacer posible un modo de datagrama de la comunicación de una computadora de conmutación de paquetes en ambiente de redes de computadoras interconectadas. Este protocolo asume que el protocolo IP se utiliza como el protocolo de nivel inferior.

Este protocolo provee un procedimiento para que los programas de aplicación puedan enviar mensajes a otros programas con un mínimo de mecanismo del protocolo. El protocolo esta orientado a la transacción de datos; la protección para la entrega y la duplicación no están garantizadas. Las aplicaciones que requieren la entrega de una trama de datos digna de confianza deberán usar el *TCP*

CAPITULO 4 Conexión con máquinas remotas

Para iniciar este capítulo, imagine que un administrador es responsable de las operaciones de una computadora central. Esta computadora tiene que realizar diversas tareas y funciones, así como soportar varias terminales que poseen diferentes características. Por ejemplo; un usuario en una terminal *DEC*, necesita comunicarse con un usuario en una terminal *HP*. Esto no es fácil; ambos dispositivos usan diferentes controladores de pantalla, caracteres de control de teclado, y una línea de protocolos para el manejo del tráfico en las comunicaciones.

Si la computadora anfitrión soporta una gran variedad de terminales, preciados recursos son consumidos en los ciclos de *CPU* de la máquina para resolver los diferentes protocolos, el diseño, y el soporte de los diferentes códigos de protocolos de *software* para envío. El administrador de las operaciones del centro deberá consumir una gran cantidad de tiempo y considerables recursos en el desarrollo o adquirir sistemas que proporcionen facilidad de transmisión entre las diferentes máquinas.

4.1 EL NIVEL DE APLICACIÓN INTERNET

El nivel de aplicación es el nivel más alto del grupo de protocolos *INTERNET*. El grupo no necesita subdividir el nivel de aplicación; no obstante algunos de los protocolos *Internet* del nivel de aplicación contienen algunos subniveles internos. El nivel de aplicación *Internet* combina las funciones de los dos niveles más altos Presentación y Aplicación del modelo de referencia *OSI*. Se distinguen dos categorías de los protocolos del nivel de aplicación: Los protocolos de usuario que proveen servicio a los usuarios directamente, y los protocolos de soporte que proveen funciones de sistema comunes. Los protocolos de usuario *Internet* más comunes son :

- *TELNET* (acceso remoto)
- *FTP* (transferencia de archivos)
- *SMTP* (entrega de correo electrónico)

Estos no son los únicos, ya que existe un gran número de protocolos de usuario estandarizados, y muchos protocolos de usuario privados. Los protocolos de soporte, usados para el mapeo de nombres de los anfitriones, inicialización (*booting*) y administración, incluyen a los protocolos *SNMP, BOOTP, RARP*, y el *Domain Name System (DNS)*.

4.1.1 Conexión a una máquina remota con telnet

La comunicación y transferencia de archivos así como las sesiones en una máquina remota, pueden resultar una tarea difícil y muy cara. *Telnet*, proporciona algunas soluciones para estos problemas. Por ejemplo, define un procedimiento que permite a computadoras anfitrión averiguar acerca de las características de las terminales ligadas a otros anfitriones con las que ellas se comunican, y proporcionan convenciones para la negociación de un número de funciones y servicios para una sesión de terminal basada entre dos máquinas. Esto se aproxima a una mejora del protocolo cambiando el problema, porque la negociación de máquinas tiene la opción de no usar un servicio que no puede soportar una u otra máquina. El protocolo *TELNET*, permite a un programa en una máquina anfitrión (llamado "*cliente telnet*") usar los recursos de la otra máquina (llamada "*servidor telnet*").

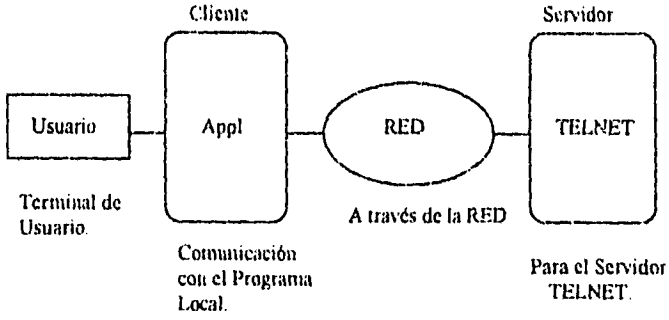


FIGURA 4-1. El modelo *TELNET*.

Debido a que *TELNET* soporta una gran variedad de dispositivos remotos con una máquina remota, lo han llegado a llamar sesión remota (*remote login*). Este se encuentra construido sobre tres ideas principales:

1. El concepto de una *Terminal Virtual de Red*, o *NVT*.
2. El principio de *opciones negociadas*.
3. Una *vista simétrica de terminales y procesos*.

TERMINAL DE RED VIRTUAL

El término virtual es usado porque un *NVT* no existe, este es un dispositivo imaginario que presenta características de una terminal. La idea es liberar a las computadoras anfitrión de mantener las características acerca de todas las terminales con las que hay comunicación. Con el *TELNET*, ambos usuarios y dispositivos servidores son necesarios para representar las características de su terminal en relación de una terminal virtual. El resultado final es que los dispositivos parecen ser comunicados con la terminal virtual de red, porque ellos están asumiendo que ambas partes se proveen un mapeo complementario. El *NVT* es intencionado para compensar la balanza entre ser excesivamente restringido (no ofreciendo un vocabulario suficientemente rico para localización en sus conjuntos locales de caracteres), y ser excesivamente generalizados (penalizando a usuarios con terminales modestas).

EL PRINCIPIO DE OPCIONES NEGOCIADAS.

Se basa en el hecho de que muchas computadoras proveerán servicios adicionales sobre y alrededor de aquellos disponibles dentro de un *NVT*, es posible que muchos usuarios tengan terminales sofisticadas y quisieran tener elegancia, en vez de servicios mínimos independientes, pero estructurados con el protocolo *TELNET*, para tal caso existen diversas "opciones" que serán inhabilitadas y podrán ser utilizadas con la estructura "*DO, DON'T, WILL, WON'T*" (discutidas abajo) para permitir a un usuario y a un servidor estar de acuerdo con un conjunto de convenciones más elaborado para su conexión. Tales opciones pueden incluir cambiar el conjunto de caracteres, el modo, etc.

La estrategia para establecer el uso de opciones es que cada computadora (o ambas) inicie una solicitud. Entonces la otra podrá ser capaz de aceptarla o rechazarla. Si la solicitud es aceptada, la opción inmediatamente se realiza, si fuera rechazada la conexión permanece como una *NVT*. La figura 4-2 muestra como pueden ser negociadas las opciones entre dos anfitriones

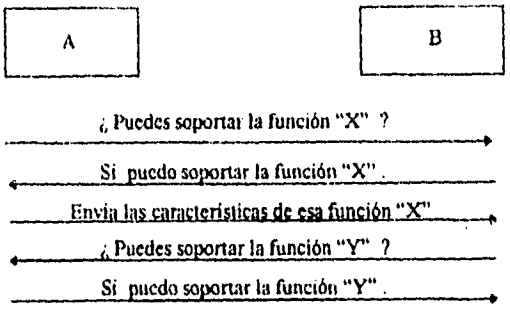


FIGURA 4-2. Negociación de las opciones

UNA VISTA SIMÉTRICA DE TERMINALES Y PROCESOS

TELNET no realiza ninguna conversión entre diferentes máquinas, determina las características para que las terminales puedan comunicarse e inter-colaborar, una con la otra para intercambiar datos. Además que ambos sistemas deben estar de acuerdo en los comandos que transmiten y reciben en ambas terminales virtuales. el programa *telnet* posee dos modos de ejecución:

1. Si *telnet* es ejecutado sin argumentos, entra el modo comando, que es indicado por el cursor, *telnet*. En este modo, *telnet* acepta y ejecuta los comandos listados en la tabla 4-1.
2. Si fuera ejecutado con argumentos, ejecuta un comando de apertura (ver tabla 4-2) con estos argumentos. Una vez abierta una conexión *telnet* entra en *modo input*. El *modo input* puede ser *caracter-por-caracter* (*CpC*) o *linea-por-linea* (*LpL*), dependiendo del sistema remoto.

Comando	Descripción
`Open host [puerto]`	Abre una conexión al *host* nombrado.
`Close`	Cierra una sesión *TELNET* y regresa el modo comando.
`Quit`	Cierra cualquiera sesión *TELNET* y abandona el programa.
`^Z`	Suspende *telnet*. Este comando solamente trabaja cuando el usuario está utilizando el `csh`.
El modo `type`	Puede ser de la forma *CpC* o *LpL*. El *host* local pregunta al host remoto que modo puede usar una; vez respondida la pregunta el *host* remoto entra al modo solicitado.
`Status`	Despliega la condición actual.
`Display [argumento]`	presenta todo, o una parte del conjunto.
`? [ comando ]`	Acceso ayuda en línea. Sin argumentos, telnet imprime un sumario de la ayuda. Si un comando es especificado, *telnet* imprime la información de ayuda para ese comando.
`Send argumento (s)`	Envía uno o más secuencias de caracteres especiales al *host* remoto.

TABLA 4-1. Comandos disponibles en *telnet*.

En modo *caracter-por-caracter*, el texto es enviado a la máquina remota tal y como es mecanografiado. En cambio en el modo *linea-por-linea*, el texto es repetido localmente y solamente estando las líneas completas se envían a la máquina remota.

Aunque una conexión *TELNET* a través de la red, es básicamente *full-duplex*, el *NVT* es visto como un dispositivo *half-duplex* operando en un modo de línea. Esto es, a menos que ocurra lo contrario, cuando las operaciones son negociadas.

NOMBRE	CÓDIGO	DESCRIPCIÓN
escape		Envía un caracter de *escape* a *telnet*
synch		Causa al sistema remoto descartar la entrada previa que no ha sido leída. Esta se envía como información urgente a *TCP* y no trabaja en un sistema 4.2 de *BSD*.
Brk (*break*)	243	Para interrumpir el comando en el sistema remoto.
Ip (Proceso de Interrupción)	244	Que causa que el sistema remoto aborte el proceso que esta corriendo actualmente.
Ao (Aborta Salida)	245	Nivela toda salida del sistema remoto a la terminal del usuario.
Ayt (Está Ud. Allí).	246	El sistema remoto puede o no responder.
Ec (Erase Caracter)	247	Que causa al sistema remoto borrar el último caracter ingresado.
El (Erase Línea)	248	Que causa al sistema remoto borrar la línea actual.
Ga (Adelanta).	249	Frecuentemente esta secuencia no tiene significado para el sistema remoto.
Nop (Ninguna operación).	241	No realiza ninguna operación.
?		Imprime la información de ayuda para el comando de envía.
Off		Apaga el valor de la función asociada con la variable.

TABLA 4-2. Argumentos que pueden ser especificados en *telnet*

4.1.1.1 Estructura de los comandos telnet

La unidad de datos *telnet* es llamada comando, y su formato es representado en la figura 4-3. El comando se encuentra formado por tres *bytes*, el primer *byte* es un interprete (*IAC*). Este es un código reservado para *telnet*; también es un caracter de escape, porque es usado por el receptor para detectar si el tráfico de entrada no es un dato o un comando *telnet*. El siguiente *byte* es el código del comando; su valor es usado con el *byte IAC* para describir el tipo de comando. El tercer *byte* es conocido como la opción de negociación. Este es usado para definir un número de opciones para ser usadas durante la sesión, por supuesto las colisiones de *bytes* de datos con valores de comando reservados son minimizados, todas las colisiones presentan este inconveniente.

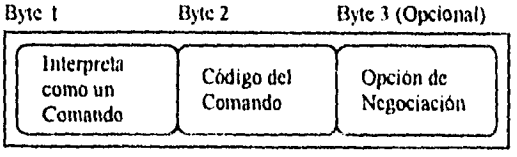


FIGURA 4-3. El formato de comandos *TELNET*.

Cuando se desea que el primer *byte* sea interpretado como un caracter de escape y además como interprete, solamente se duplica el *IAC* para ser enviado como comando, y los otros 255 códigos pueden ser pasados transparentemente. La tabla 4-3 lista las 6 funciones *telnet*.

NOMBRE	CÓDIGO	SIGNIFICADO
SE	240	Fin de sub-negociación de parámetros.
Data Mark	242	La porción de flujo de datos de un *Synch*. Esta debe siempre estar acompañado por una notificación urgente de *TCP*
SB	250	Indica que la siguiente sub-negociación de opción indicada.
WILL	251	Indica el deseo comenzar ejecutar, o confirmación que usted está ahora ejecutando, la opción indicada.
WON'T	252	Indica el rechazo para desempeñar, o continuar ejecutando, la opción indicada.
DO	253	Solicitud que el otro miembro lo desempeñe, o confirmación de espera.
DON'T	254	Indica la demanda que el otro miembro necesita para seguir o confirmacion que no esta esperando más tiempo miembro.
IAC	255	*Byte* de Datos 255.

TABLA 4-3. Códigos y funciones de *telnet*.

4.1.2 Conexión a una máquina remota con rlogin

A veces se puede necesitar hacer una sesión remota en una computadora con *UNIX* sobre una red *TCP/IP* y llevar acabo algunas tareas, esto puede ser realizado utilizando la orden *rlogin*. Esta orden es utilizada para abrir sesión en otras máquinas remotas como si fuera un usuario local, este comando lleva el nombre de la máquina remota como argumento, por ejemplo:

```
$ rlogin sysul2
```

Por omisión, la orden *rlogin* establece un entorno en la máquina remota análogo al entorno que se tendría si se abriese la sesión en aquella máquina directamente; inclusive se tiene acceso al directorio raiz del usuario en la máquina remota. Suministrándole el *ID* del usuario y el tipo de terminal que se está utilizando mediante el envío del valor de la variable *TERM*. Durante una sesión *rlogin*, se transfieren caracteres en ambos sentidos entre los dos sistemas, ya que durante la sesión se permanece conectado al anfitrión original. También se puede utilizar *rlogin* para presentarse en un sistema remoto utilizando un *ID* de usuario diferente. Para hacer esto, utilice la opción *-l* seguida del *ID del usuario*. Ejemplo:

```
$ rlogin -l armando sysul3
```

Bajo ciertas circunstancias, puede utilizar *rlogin* para presentarse en la máquina remota sin necesidad alguna de introducir la contraseña en esa máquina. Esto ocurre cuando se tiene autorizado (ver apéndice C.1), tras una breve espera aparecerá el cursor de la máquina remota y podrá entonces ejecutar órdenes en ese sistema.

```
vax> rlogin sysul3
sysul3[1]
```

Si no se dispone de una presentación en la máquina remota, *rlogin* solicitara una contraseña. Esta petición de contraseña se puede evitar introduciendo una nueva línea y rlogin solicitara una nueva presentación.

```
vax> rlogin sysul3
Password: <enter>
login:
```

Finalmente, bajo ciertas circunstancias; se le denegara el acceso cuando intente presentarse a una máquina remota, si no existe entrada para el usuario en la base de datos de contraseñas del sistema remoto, no podrá presentarse en absoluto. Pero si el nombre de su máquina local esta en el archivo `/etc/hosts.equiv` en la máquina remota, puede presentarse en la máquina remota sin introducir una contraseña. Esto ocurre debido a que la máquina remota confía en su máquina local.

En caso contrario, cuando se tiene una entrada en la base de datos de contraseñas de la máquina remota, pero el nombre de su máquina no esta en el archivo `/etc/hosts.equiv` del anfitrión remoto y no existe en el archivo .rhosts en el directorio principal del usuario en la máquina remota, solicitara una contraseña. Sin embargo en este caso puede presentarse, pero no podrá ejecutar procesos remotos tales como *rsh* o *rcp*. Esto impide que pueda tener presentación múltiple a otra maquina.

Mientras la sesión remota este activa, todas las órdenes serán introducidas en la máquina remota y la sesión local esperara hasta que se termine con el sistema remoto. Si se esta utilizando un *shell* con control de trabajos, se puede suspender temporalmente la sesión remota introduciendo una tilde (~) seguido del caracter de suspensión, *CTRL-Z*. De este modo se detiene la sesión *rlogin* hasta que sea reinicializada con el control de trabajos. Si se introduce *CTRL-Z* sin tilde, *rlogin* transmitirá el caracter *CTRL-Z* a la máquina remota, en donde el shell intentara suspender un trabajo alli. Hay que agregar algo muy importante las órdenes *telnet* y *rlogin* son muy similares, pero *rlogin*, solo funciona en sistemas con *UNIX*, mientras que el *telnet* lo hace en situaciones en donde *rlogin* es inadecuado.

4.1.2.1 Estructura del rlogind (servidor de sesión remota)
El *daemon rlogind* es usado por el programa *rlogin*; el *daemon* provee un *login* remoto facilitado con la autenticidad basada en los números de puerto privilegiados. Este a su vez es invocado por *inetd* cuando recibe una conexión en el puerto indicado para una sesión. Cuando una solicitud de servicio es recibida, el siguiente protocolo es inicializado:

1. Revisa el puerto fuente del cliente. Si el puerto no esta en el rango de 0 a 1023, el servidor aborta la conexión.
2. Revisa la dirección fuente del cliente y solicita el nombre del host correspondiente. Si el nombre-del-*host* no puede ser determinado, se usa la notación-punto.
3. Legaliza al usuario de acuerdo con los siguientes pasos:
   a) El nombre del usuario remoto es buscado en el archivo de contraseñas y hace un *chdir* al directorio principal del usuario, si esto falla la conexión es abortada.
   b) Si el usuario no es el *super user* el archivo `host.equiv` o `host.pv` son revisados por si hay una equivalencia.
   c) Si la búsqueda falla o si el *super user* busca en el archivo `.rhosts`, y también llega a fallar, la conexión es abortada.

El proceso login padre, es decir quien inicia la sesión (la máquina local), manipula el lado maestro de la pseudo terminal, operando como un intermediario entre el proceso *login* y el programa login cliente. En la operación normal, el protocolo *Pty* (configura la terminal remota) es invocado para proveer señales de interrupción (^*Q* y ^*S*) a los programas remotos.

El proceso *login* cliente (la máquina remota) indica a la terminal el tipo de velocidad de transmisión en *baudios*, así como el tipo de terminal a través de la variable de entorno *TERM*. El tamaño de pantalla o ventana de la terminal es solicitado por el cliente, y cualquier cambio en el tamaño de la ventana del cliente es enviado a la pseudo terminal.
El procedimiento de autenticidad usado aquí supone la integridad de cada máquina cliente y el medio conector. Esto es inseguro, pero es útil en un entorno abierto.

4.1.3 Transferencia de archivos con el comando rcp
La orden *rcp* se utiliza para copiar archivos en y desde máquinas remotas sobre una red *TCP/IP*. Actúa de forma muy similar al comando *cp* normal; para copiar un archivo en un directorio especificado, dando al archivo el mismo nombre que tiene en el sistema remoto. La diferencia radica en que este comando, como la mayoría de las órdenes r*, cuando reciben una solicitud, llaman al servidor de sesión remota *rlogin*. Utilice el siguiente formato .

```
$ rcp «máquina»: «nombre-camino»  «directorio»
```

Por ejemplo, para copiar el archivo de nombre `/santiago/gato.txt` en la máquina *sysul1* al directorio `/yopo/datos/santiago/gato.txt` :

```
$ rcp  sysul1:/santiago/gato.txt      /yopo/datos/santiago
```

En el apéndice C.2, se presentan una serie de ejemplos, con mayor dificultad así como una explicación detallada de cada uno de ellos.

4.1.4 Transferencia de archivos con ftp

Para hacer transferencia de archivos con el protocolo *FTP*, la máquina local debe estar conectada al *Internet* y el usuario debe saber la dirección *Internet*; además debe tener una cuenta en el servidor *FTP*. En algunos *host* remotos se permite el uso del programa *FTP* (*File Transfer Protocol*) para trasladar archivos sobre el *Internet*. El programa permite hacer una sesión en otra máquina conectada al *Internet*. Por propósitos didácticos, llamaremos a cada sitio, máquina, o *host* remotos como servidores de *FTP*.

Una vez hecho el enlace a través de *FTP*, es posible revisar los directorios, y el envió de archivos a la computadora local; a este proceso se le denomina bajar o carga-baja (*downloading*). Si por el contrario, la cuenta que usted está usando en el servidor *FTP*, permite enviar archivos de su máquina local al servidor *FTP*, el proceso se denomina subir o carga-alta (*uploading*).

4.1.4.1 Uso del FTP

El programa *FTP* puede ser usado desde Europa (132.248.59.10); una vez inicializada la sesión en su màquina local, teclee después del *prompt* `C:\>` ftp y en seguida la dirección del servidor *FTP;* por ejemplo, si se quisiera conectar de Europa a la micro *VAX* del IFUNAM seria :

```
C:\> FTP 132.248.7.1
```

Después de un pequeño desplegado de mensajes del *ftp*, este le pedirá el nombre de la cuenta (*account*). Para comenzar una sesión *ftp* se puede hacer de dos formas; la printera es con el programa se proporciona la orden `open` y el nodo así:

```
$ ftp
ftp> open hidalgo
```

La segunda es más directa, al especificar el nodo, después de la orden *ftp*.

```
$ ftp hidalgo
```

Una vez aceptada su entrada al sistema aparecerá, dependiendo de la versión de *FTP*, su prompt respectivo, `FTP>`.

4.1.4.2 ¿Qué hay disponible en el servidor de FTP...?

Se usa el `dir` o el `ls` para listar archivos en el directorio del servidor *FTP*. Dependiendo de la versión de *FTP*, el comando `ls`, permite un listado de los archivos que se encuentran en el directorio local y puede ser usado en función de la versión de *FTP* que se tenga. El comando `cd` es usado para cambiarse entre directorios del servidor *FTP*, mientras que el `LCD` se usara para cambiarse entre directorio del *host* local.

Si se necesita más información sobre un comando el *FTP*, se cuenta con un programa de ayuda (help) sobre el uso de los comandos. Para poder hacer un *shell* de la máquina sin salir del *ftp*, solo es necesario colocar el signo de admiración y la orden a ejecutar, ejemplo:

```
ftp> !dir a:
```

También es posible incluir referencias o una lista a una fuente de documentación que provee ayuda con *FTP*, para poder hacer esto es necesario que se contacte con una persona que pueda ayudar (Mike de Leo en DGSCA). Para traer un archivo del servidor *FTP* a una computadora local se debe usar el comando GET de la siguiente forma:

get <nombre del archivo> [nuevo nombre]

si desea que el archivo conserve el mismo nombre, solo escriba el "<nombre del archivo>" y dé enter. En caso de querer cambiar el nombre, entonces escriba el "nombre del archivo" y después como desea nombrarlo "[nuevo nombre]", este cambio aparecerá en su máquina local. Para traer varios archivos del servidor *FTP* a la máquina local, se usa el comando mget y el metacaracter o comodín (*); por ejemplo la siguiente combinación puede traer todos los archivos con extensión .zip del directorio en el que se encuentra haciendo la sesión remota

```
mget *.zip
```

Para enviar un archivo de la máquina remota, se usa el comando PUT. Por ejemplo, el siguiente comando envía el archivo (con el mismo nombre).

```
put document.txt
```

Para enviar varios archivos, se usa el comando MPUT con un metacaracter en la especificación del archivo. Por ejemplo:

```
mput *.txt
```

Envía a la máquina remota todos los archivos con extensión .txt del directorio presente. Nota: al bajar los archivos a la máquina local hay que ser prudente con el formato de estos, ya que existen distintos tipos de formato para transferencia (*ASCII* y *BINARIO*). Si se baja un archivo binario en formato *ASCII*, este programa no correrá correctamente. Antes de copiar un archivo hay que asegurarse de establecer correctamente el tipo de transferencia, escribiendo *ASCII* para los archivos en este formato o *binary* para los archivos ejecutables. Una vez fijo el tipo de transferencia, pueden utilizarse las órdenes de transferencia.

Cada versión de *FTP* usa comandos distintos para cambiar el formato del archivo que va a ser traído. Por ejemplo, algunas versiones tienen un comando para cambiar a *BINARIO* o *ASCII*. Con *CMU/TEK FTP* para *VAX/VMS*, el comando TYPE puede ver el contenido de un archivo a través del servidor *FTP*, mientras que el *SET TYPE ASCII* (es el formato por *default*) o el *SET TYPE BINARY* (algunas veces el *IMAGE* es usado en lugar del *BINARY*) estos comandos son usados para cambiar el formato de archivo.

Si se incluye una línea al final del párrafo, de la siguiente forma: Use el comando SET TYPE para cambiar al tipo apropiado del archivo, después teclee el get o el mget y archivo(s). Cambia SET type, solo si es necesario para el comando apropiado. Puesto que es importante trasladar archivos de diferentes formatos correctamente, a continuación se proporciona algunas extensiones de archivos, así como su formato en la tabla 4-4.

BINARY	ASCII
.COM	.TXT
.EXE	.DOC
.ARC	.UUE
.ZIP	.EPS
.LZE	

TABLA 4-4. Extensiones de archivos y su formato.

Hay que agregar que los archivos que no poseen extensión, como son el caso de *INDEX* o *README*, son archivos en formato *ASCII*. Si se tienen dudas acerca del tipo de formato del archivo, se recomienda que primero se envie el archivo en formato *ASCII*, en caso de no ser este el formato correcto, intente después con el *BINARY* o *IMAGE*. Nota: algunas de las extensiones descritas en la tabla 4-4 por lo regular son comprimidas en la red, para reducir el tamaño del archivo y el tiempo de transmisión a través del Internet. Para cerrar la conexión con el servidor *FTP* y salir del programa *FTP*, hay que teclear `quit` o `bye`.

4.1.4.3 Requerimientos especiales para diferentes servidores de FTP

Varios servidores de *FTP* se ejecutan bajo diferentes sistemas operativos. Algunos de estos son muy sensitivos. En estas ocasiones los comandos `login`, `cd`, `dir`, `get` y `put` deben ser escritos en letras minúsculas. Algunos otros requieren que el nombre de los archivos sea escrito igual que en el directorio remoto. Como es muy difícil saber qué comandos o qué tipos de comandos trabaja con un específico servidor *FTP*, se recomienda que los comandos sean escritos en letras minúsculas.

4.1.4.4 Encontrando qué sitios de FTP se encuentran disponibles

Hay alrededor de 700 sitios con *FTP* anónimo en la Internet. Dos de los sitios más comúnmente usados son `Sumex-aim.stanford.edu` y `wsmr-simtel20.army.mil`. Una gran colección de programas para *Macintosh* se encuentran en disponibles el *Sumex-aim*, en cambio el *Simtel20* posee una colección de programas para *MS-DOS*. Como estos programas son *copyright*, en algunos casos es necesario pagar una pequeña cantidad para usarlos. Para encontrar los sitios que proporcionan servicios *FTP*, existe una lista *FTP* (llamados `anon.ftp.list`) que es disponible en `unsvax.nevada.edu`. Otra lista similar se encuentra en el *FTP* anónimo de `pilot.njin.net` (128.6.7.38) que es actualizada mensualmente en el *USENET*.

La lista de sitios *FTP* contiene el nombre del sitio *FTP*, la dirección *Internet*, la fecha de la última modificación, información sobre la zona de tiempo, disponibilidad e información sobre el *login* y *password*. La información sobre la zona de tiempo es basada en la diferencia entre el meridiano de Greenwich y el tiempo local. Es importante recordar que el *FTP* anónimo es un privilegio, y no un derecho. Muchos sitios con *FTP* restringen el horario para efectuar *FTP* anónimos. Algunos otros limitan el número de usuarios que pueden realizar el *FTP* anónimo, tiempo de sesión a un usuario, número de archivos para cargar o bajar del host en una sesión.

4.1.4.5 Entendiendo y listando directorios UNIX

No es necesario conocer qué marca de computadora o qué sistema operativo es usado por el host remoto. Sin embargo, la información que proporciona el `dir` o `ls` abajo puede ayudar a moverse libremente a través de los directorios. Cuando se usa el comando `dir` o `ls`, las máquinas con sistema operativo *UNIX* despliegan algo muy similar a lo siguiente:

```
-r--rw-r--      1     nica  consult    1316  Jun 21      10:49 ABSTRACTS
-r--rw-r--      1     nica  consult    1044  Apr 18      1991  README
drwxr-xr-x  2   root  system           512   Jun 26      10:43 bin
dr-xr-xr-x  2   nica  ftp              512   Nov  9      1991  pub
```

La información en este listado puede llegar a ser de gran ayuda, la primera columna indica si es archivo con una "-" o directorio a través de la letra "*d*". Los siguientes nueve caracteres en la columna de información representan el grado permitido de acceso por varios usuarios a los archivos. Las primeras tres posiciones (de la 2 a la 4) son tipos de acceso permitido por el propietario del directorio, los siguientes tres son para su o sus grupos, y los últimos tres son para los otros usuarios. El código "*wrx*" permite escribir, leer y ejecutar. Una "-" en alguna de estas posiciones indica un acceso negado ya sea a la lectura, escritura o ejecución. La última columna indica su nombre respectivo. Los primeros dos renglones del ejemplo son archivos y los dos últimos direcciones. Nota: el permiso de "lectura" (*read*) para un directorio permite desplegar su contenido, el de "ejecutar" (*execute*) permite el acceso a los archivos en el directorio. El permiso "lectura" en un archivo permite el uso del comando *get*. Por último los números que están antes de la fecha abreviada, indican el tamaño en *bytes*.

Existe una gran variedad de *software* de dominio público que está disponible en la *Internet*. El *ftp* anónimo es el modo más habitual de distribuir estos programas.

4.1.4.6 Comprimiendo y descomprimiendo archivos

Para tener mayor espacio en los directorios de acceso público, se usan algunos programas de compresión. Cada compresión a un archivo es usualmente identificada por el tipo de extensión usada. Algunas de las extensiones más que comunes son incluidas son las siguientes:

`.arc, .zip, .uue y .tar.`

El mapa de abajo provee información del nombre y la técnica de compresión o programa usado para cada una. Los programas para compactación y descompresión de las extensiones listadas abajo, se encuentran disponibles como programas (*software*) de dominio público. Por lo regular los archivos grandes están disponibles en el formato `.tar` comprimido. Para la transferencia de tales archivos, debe primero utilizarse la orden `binary`, después de recibir el archivo, usar `uncompress` y al final `tar` para recuperar el archivo original. Un ejemplo de una conexión *FTP* se encuentra en el apéndice C.4.

Extensión	Descripción	Herramienta para descompresión
.arc	Formato del archivo. Los archivos *Arc* son almacenados como binarios (*image*).	creado con PKARC, descompresión con PKXARC.
.zip	Formato *Zipped*. Los archivos *ZIP* son almacenados como binarios (*image*).	creado con PKZIP, descompresión con PKUNZIP
.uue	*Uuencoded* . Los archivos son almacenados como binarios (*image*).	creado con uuencoded y descompresión con uundecoded
.tar	Formatos de *UNIX* `tar` (*tape archive*)	utileria `tar`

TABLA 4-5. Nombre y técnica de compresión.

4.2 CONEXIÓN AUTOMÁTICA DE FTP POR MEDIO DEL ARCHIVO .netrc

En algunas ocasiones los usuarios llegan a olvidar la dirección de un *host* remoto, o a confundir comandos de otros sistemas con comandos de *ftp*, o inclusive se resisten a conocer más sobre nuevos sistemas concretándose solo a lo que conocen , pero desean obtener los beneficios de *ftp*. Este trivial problema fue solucionado con un archivo de personalización, conocido como *.netrc* ; este archivo contiene instrucciones para transferencias, así como definiciones personales de comandos *ftp*. Para poder obtener los beneficios de este archivo se deben cumplir las siguientes restricciones:

a) El archivo se debe encontrar en el directorio principal del usuario que inicia la transferencia de archivos.
b) Si el archivo incluye contraseñas, debe ser protegido contra lectura y escritura para el mundo y el grupo solo el dueño puede leerlo o modificarlo.

El archivo *.netrc* utiliza el siguiente formato:

- Cada línea define opciones para una máquina específica.
- Una línea escrita puede ser del tipo `machine` o por `default` dejarla en blanco.
- La línea por `default` tiene que ser la primera en el archivo si existe.
- El orden de las líneas debe ser el siguiente:
  - por `default` (el comando y el nombre del servidor)
  - por `machine` (`machine`, nombre del servidor y la opción).

Opción	Parámetro	Por default	Descripción
`machine`	nombre de la máquina	ninguno	Identifica el nombre de la máquina remota
`login`	nombre	nombre local	Identifica a un usuario en una máquina remota
`password`	*password*	ninguno	Contraseña para la sesión remota
`account`	*password*	ninguno	Contraseña de la cuenta Adicional
`macdef`	nombre del macro	ninguno	Define un macro como el comando `ftp macdef`

TABLA 4-6. Opciones válidas para una línea de *machine*.

Ejemplo: Este es un ejemplo de un archivo *.netrc*:

```
machine cactus login máquina smith
machine nic.ddn.mil login anonymous password anonymous
machine palm.statteu.edu login smith password ualonxcjdjk
macdef byenow
quit
macdef ls
dir
```

La primera línea permite a *ftp* identificar a Smith en la máquina cactus, después de preguntar por el *password*, como se muestra en el siguiente ejemplo:

```
                          $ ftp cactus
Connected to cactus.tech.edu
                          FTP server lready.
Password required for smith
(En este momento el usuario teclea su palabra clave)
                          User logged in
                          ftp>
```

La segunda línea del archivo permite al usuario desempeñar un *ftp* anónimo transfiere después de teclear este comando:

```
                          $ ftp nic. ddn. mil
```

La tercera línea permite a Smith identificarse en la máquina `palm.stateu.edu`. Al usuario no se le solicitara una contraseña porque esta línea de máquina incluye información de contraseña. Debido a que el archivo `.netrc` incluye la contraseña, no debe tener permiso para ser leído por el mundo ni por el grupo. Las líneas `macdef` son definiciones de macros, que operan mucho como un alias. Una línea en blanco tiene que seguir cada definición macro para señalar el fin del macro. La primera definición de macro define `byenow` como un alias para el comando *quit*.

4.3 USO DE FTP ANÓNIMO

Un grupo de personas de la universidad de McGill en Canadá crearon en conjunto, un sistema de consulta conocido como ***ARCHIE***, originalmente fue hecho para una forma rápida y fácil de conocer el contenido de los muchos sitios de *FTP* anónimo, que son mantenidos alrededor de la red. Con el tiempo, el *ARCHIE ha* incluido otros servicios muy valiosos.

Las bases de datos *ARCHIE*, son accesibles a través de una sesión interactiva *TELNET*, consultas por Correo electrónico (*Email*), clientes de *X-Windows* y comandos de línea. Las respuestas del correo electrónico pueden ser usadas con servidores correo *FTP* para aquellos que no se encuentran dentro *INTERNET*. Actualmente *ARCHIE* contiene aproximadamente mil sitios de *FTP* anónimo, lo cual representa casi un millón de archivos almacenados en *INTERNET*.

En conjunto estos archivos representan un promedio de 50 *GigaBytes* de información; con nuevos archivos añadidos diariamente, el servidor de *ARCHIE*, actualiza automáticamente la lista de información de cada lugar, aproximadamente cada mes. Esta actualización constante podría desperdiciar recursos de red, pero asegura que la información en cada sitio *FTP* esta renovada. A continuación se presenta una pequeña lista de servidores *ARCHIE* :

`archie.sura.net`	(Maryland, USA)
`archie.unl.edu`	(Nebraska, USA)
`archie.mcgill.ca`	(El primer servidor ARCHIE en Canada)
`archie.au`	(Australia)

Ejemplo de sesión *ARCHIE*; realice una conexión *TELNET* con el servidor *ARCHIE*:

```
$ telnet archie.sure.net
Trying 128.167.254.179...
Connecteds to archie.sure.net
Escape character '^['

SunOS UNIX (nic.sura.net)
login: archie
```

En el prompt *login*, teclee `archie` (no se requiere de una palabra clave), mostrará un mensaje detallando la información acerca del ambiente, en el cual trabaja el sistema; finalmente, aparecerá el prompt `archie>` en el cual se podrán introducir los comandos, *prog* para realizar una búsqueda, *mail* para correo, etc.

4.4 ESTADO DE LA RED CON netstat

En algunas ocasiones es importante conocer el estado de la red para el administrador del sistema, esto se puede obtener mediante el comando `netstat` que despliega la información de protocolos y *sockets* de la red en estructuras simbólicas. Dependiendo de las opciones suministradas, hay una variedad de formatos de salida. Si no hay opciones especificadas, `netsat` despliega el estado de todos los *sockets* activos utilizados en cualquiera de los protocolos listados en `/etc/protocols`.

La primera forma, para *sockets* activos	Despliega una lista de sockets activos, así como las direcciones locales y remotas, el tamaño de las colas de transmisión y recepción (en *bytes*), protocolo, y, opcionalmente, el estado interno del protocolo.
La segunda forma	Despliega el contenido de una de las otras estructuras de datos de red de acuerdo con la opción seleccionada.
La tercera forma	Con un intervalo específico, despliega continuamente información pausando entre intervalos de segundos antes de refrescar la pantalla, sobre el tráfico de paquetes en las interfaces de red.
La cuarta forma	Despliega la Arquitectura de Red Digital (*DNA*) los contadores de la Capa de Enlace de Información Ethernet para una interfaz de Ethernet, o la Fibra de Información (*FDDI*) los contadores de la Capa de Enlace de Información para una interfaz de *FDDI*. También despliega el estado del adaptador de *FDDI* y características de la interfaz *FDDI*.

TABLA 4-7. Formatos de salida más de *netstat* comunes

Los formatos de dirección son de la forma *host.puerto* o *red.puerto*; las direcciones de *hosts* y red son desplegadas simbólicamente de acuerdo a las bases de datos `/etc/hosts` y `/etc/networks`, respectivamente. Si un nombre simbólico para una dirección es desconocido, o si la opción *-n* es especificada, la dirección es impresa en el formato punto. Las direcciones y puertos sin especificar aparecen como un asterisco (*).

La interfaz provee un despliegue en forma de tabla de las estadísticas acumulativas considerando paquetes transferidos, errores y las colisiones, así como la dirección de red de la interfaz y la máxima unidad de transmisión (*mtu*). De esta forma una tabla de ruteo indica las rutas disponibles y su condición. Cada ruta consiste de un hosts o red destino y un *gateway* para usar la retransmisión de paquetes.

El campo de banderas despliega el estado de la ruta (por ejemplo, *U* si esta arriba), si es un *gateway* (G), y si fue creada dinámicamente por un *redirect* (*D*). Hay que recordar que las rutas de dirección son creadas por cada interfaz incluyendo al hosts local. Los protocolos de conexión orientada normalmente sostienen una ruta única durante la conexión, mientras que los protocolos con una conexión pequeña obtienen una ruta mientras envían al destino.

Cuando *netstat* es invocado con un intervalo en un argumento, despliega una serie de estadísticas relacionadas con la interfaz de red. Este despliegue consiste de una columna para la interfaz principal (la primera interfaz encontrada durante la autoconfiguración), y una columna resumida de información para todas las interfaces. La interfaz principal puede ser reemplazada con otra interfaz con la opción *-I*. La primera línea de cada pantalla de información contiene un resumen desde que el sistema inicializa. Las líneas posteriores de la salida muestran valores sobre el intervalo anterior. Todas opciones de este comando, así como algunos ejemplos son mostrados en el apéndice C 5.

4.5 ESTADO DE LAS MÁQUINAS REMOTAS CON ruptime

Este comando muestra por cada línea el estado para cada máquina en la red local. Si el número de la máquina es dado, solo el estado de esa máquina es proporcionado en pantalla. Estas líneas son formadas por la **emisión de paquetes** en cada *host* en la red por un minuto. Por lo tanto las máquinas para las que no haya informe a los 5 minutos entonces estarán apagadas (abajo) siempre y cuando estas hayan aparecido en otra coacción.

Hay que tomar en cuenta que este comando puede llegar a tener problemas, debido a que el *daemon* rwhod envía su información a través de emisión de paquetes, por lo tanto crea una gran cantidad de tráfico en la red. En redes grandes el tráfico extra puede ser problemático, y en algunos sistemas los administradores pueden llegar a deshabilitarlo o inclusive a no activarlo, y así evitar el tráfico de datagramas. Como consecuencia la máquina puede aparecer como apagada, cuando se utiliza el comando `ruptime`.

La cantidad de usuarios exhibidos en pantalla por el comando `ruptime` puede llegar a ser incorrecto, debido a que los usuarios que hacen un *login* fallido a la máquina, incrementan la cuenta de usuarios. Esto se debe al tamaño máximo de un paquete *Ethernet* que es 1500 *bytes*, y el hecho que el *daemon* tiene que emitir su información en un paquete único. El apéndice C.6, muestra una tabla de las opciones de este comando así como algunos ejemplos.

CAPITULO 5 El subsistema *UUCP*

Además de las muchas herramientas de comunicación ya discutidas, el sistema *UNIX* incluye una facilidad de comunicación muy potente y sofisticada que permite la transferencia de archivos entre máquinas, sin demasiada atención por parte del usuario. El subsistema *UUCP* (copia de *UNIX* a *UNIX* [*UNIX to UNIX copy*]) es un paquete completo de movimiento de datos que puede transferir archivos *ASCII* y binarios entre máquinas, También puede controlar la ejecución de órdenes sobre una máquina remota, encargándose de colocar en cola trabajos para su posterior transferencia y de reintentar automáticamente una transferencia cuando falle por alguna razón El subsistema *UUCP* incluye muchas órdenes y funciones de usuario, e inclusive facilidades de adaptación para diferentes redes de comunicación. Además, el subsistema *UUCP* ofrece fuertes medidas de seguridad, herramientas completas de registro, depuración y varios protocolos diferentes de transferencia de datos con chequeo de errores adecuados al tipo de red.

5.1 ¿ PORQUE USAR EL SUBSISTEMA UUCP[1] ?

1. *Disponibilidad.* Es la forma de interconexión más usada por sistemas *UNIX*, porque el *software* de comunicación viene en forma estándar en cada versión de sistema operativo *UNIX*.
2. *Interconexión económica.* Es muy barato instrumentar *UUCP*, ya que el *software* viene incluido en todas las versiones de sistema operativo y si se desea interconectar dos sistemas *UNIX*, solo se requiere un cableado entre los puertos series *RS-232*, en caso de querer hacer una conexión remota, solo necesita un *modem*, instalación telefónica y cable
3. *Transportación confiable.* Los mayores problemas residen en la interconexión y no en el *software*. Los archivos son transferidos en unidades llamadas paquetes, si un paquete no es recibido correctamente será retransmitido hasta que sea bien recibido. El ruido puede llegar a alterar la comunicación, pero no evita la pérdida de la información a no ser que el enlace sea perdido completamente.
4. *Un común denominador.* El sistema puede ser usado para interconectar varias máquinas de diversas marcas, tipos e inclusive diferentes versiones de sistemas operativos *UNIX* así como *MS-DOS*.
5. *Accesos a la comunidad mundial USENET.* Al instalar *UUCP*, se tiene la posibilidad de hacer accesos a la red *USENET* y poder intercambiar archivos, programas e inclusive noticias actualizadas.

---

[1] El subsistema *UUCP* data de 1976, cuando fue concebido y desarrollado por primera vez por Mike Lesk en los laboratorios Bell. Una versión mejorada y reescrita del subsistema, conocida como *UUCP* Versión 2, fue distribuida con la versión 7 *UNIX System* y fue la versión estándar hasta 1983. Sin embargo, el uso extensivo del Sistema *UUCP* en un amplio rango de facilidades de comunicación, hizo necesario aumentar sus capacidades y prestaciones. Una nueva versión fue desarrollada por **Peter Honeyman, David Nowitz y Brian Redman** en los Laboratorios Bell en 1983, esta soporta un rango más amplio de conexiones en red, proporciona facilidades administrativas y corrige deficiencias de la *UUCP* Versión 2. Esta versión, conocida como HoneyDanBer (por los nombres de presentación [*login*] de sus creadores, honey, dan y ber). Sin embargo la documentación de *AT&T* refiere a esta nueva versión como "*Basic Networking Utilities*", incorporada a *UNIX* Sistema V Versión 2

5.2 Los programas más importantes de UUCP

Para determinar que tipo de versión que se posee en un sistema *UNIX*, solo tiene que ver en su directorio `/usr/lib/uucp` , si tiene un archivo `L.sys`, esta corriendo *UUCP* Versión 2, en caso que tenga un archivo llamado *Systems*, tendrá *UUCP HoneyDanBer*.

El subsistema *UUCP* permite las siguientes facilidades :

- Sesión remota a un sistema (con *UNIX* o sin el) a través de *cu*.
- Transferencia de archivos entre sistemas *UNIX* con *uucp*.
- Ejecución de comandos en un sistema remoto con *uux*.
- Transferencia de archivos con *mail*.
- Transferencia de archivos con la red *USENET*

Cada uno de los postulados los veremos con más detalle en secciones posteriores.

El sistema provee archivos para transferencias de archivos, copias, ejecución de comando, etc. La tabla 5-1, lista los programas más importantes, separándolos por categorías.

Especifica solicitudes de trabajo	
`cu`	Llama por teléfono a un sistema remoto.
`uucp`	Solicita copia de archivos entre sistemas remotos.
`uux`	Solicita ejecución de comandos entre sistemas remotos
`uuto`	Solicitud de copia de archivos, más sencilla.
`uupick`	Recupera archivos copiados con `uuto`
Checa el estado y control de trabajos.	
`uulog`	Examina la información de sesión de la *UUCP*.
`uustat`	Control de trabajos, chequeo del estado previo de Comandos y accesibilidad a un sistema remoto
`uucheck`	Del sistema HoneyDanBer interpreta el archivo de permisos de seguridad
Herramientas	
`uuencode`	Convierte de código binario a *ASCII* para enviar por correo
`uudecode`	Convierte de *ASCII* a binario
Accesos a sistemas	
`uugetty`	Del sistema HoneyDanBer, es una variación del getty que soporta un acceso *dial-in dial-out* al mismo puerto.
`Uutry`	Del sistema HoneyDanBer un `shell` de escritura que invoca al `uucico` para actualizar.
Mantenimiento	
`uuclean`	Retira archivos obsoletos de los directorios *UUCP*
`uucleanup`	Del sistema HoneyDanBer, una versión mejorada del uuclean
`uudemon`	`Shell` de escritura para mantenimiento del *UUCP*.
"Daemons"	
`uucico`	Contacta máquinas remotas y transfiere archivos
`uuxqt`	Ejecuta peticiones de comandos para un sistema remoto.
`uuschd`	Del sistema HoneyDanBer, *"daemon"* de transferencia de archivos de catalogo que se inicializa al levantar `uucico`.

TABLA 5-1. Comandos más importantes en *UUCP*

Ahora veamos en forma rápida el funcionamiento de cada uno de los comando de nuestro interés.

- ***cu.*** Permite trabajar en un sistema remoto, mientras se encuentra en sesión en la máquina local. La entrada del teclado local es enviado al sistema remoto y cualquier salida de ese sistema es desplegado en la pantalla local. Permite la transferencia de archivos, sin embargo no hay ninguna comprobación de error (*cheksum*).
- ***uucp.*** Crea primero un *job* o un archivo de comando, que contiene instrucciones para transferir la información. Dependiendo de la versión y de los argumentos *uucp* puede copiar un archivo fuente aun directorio público o hacerlo directamente al directorio deseado. Después que se ha realizado una solicitud el *uucp* invoca al "*daemon*" *uucico* para llamar al sistema remoto y poder transferir.
- ***uux.*** Realiza una ejecución remota y si son necesarios archivos de datos en sistema remoto, invoca uucico para transferirlos al sitio remoto. Se ejecuta en la máquina remota el archivo que contiene instrucciones para la ejecución del comando. El *uucico*, en la máquina remota, llama al "*daemon*" *uuxqt*, para realizar la ejecución.
- ***uuto.*** Es un programa que invoca a *uucp* para transferir archivos a un directorio público `/usr/spool/uucppublic` en el sistema remoto.
- ***uupick.*** Esta diseñado para recuperar los archivos copiados por *uuto*, en el directorio público de la máquina local.
- ***uulog.*** Es utilizado para monitorear el progreso de las transacciones realizadas por *uucp*. En la versión 2, uulog despliega la información a través del archivo `/usr/spool/uucp/LOGFILE`; escribiendo varios mensajes de estado de la sesión en cada transacción. En el caso de HoneyDanBer, el *uulog* es un *shell* que permite examinar esos archivos que son llamados después del proceso que los escribe y se encuentran en el directorio `/usr/spool/uucp/.Log` .
- ***uustat.*** Se utiliza este comando para determinar la accesibilidad a sistemas remotos, conectados a su máquina local. Para los usuarios que no tienen privilegios, el *uustat* les permite cancelar o informar acerca de la condición de los trabajos .
- ***uucode*** y ***uudecode.*** Permite la conversión de archivos binarios a *ASCII* y viceversa. Por ejemplo si se necesita enviar un archivo binario a otro sistema remoto, primero hay que cambiarlo a *ASCII* y después transmitirlo por correo. Cuando es recibido en el sistema remoto, se convierte de *ASCII* a binario. El *uucode* controla también los permisos originales y modos de operación.
- ***uugetty.*** El programa estándar *getty* fue mejorado por el sistema HoneyDanBer de modo que ahora permite entradas y salidas en una línea.
- ***uutry.*** Este *shell* de escritura solo es soportado por los sistemas HoneyDanBer, invoca a *uucico* para actualizar y guardar su salida a un archivo temporal: es importante para seguir la ruta de conexión entre sistemas remotos.
- **El *shell* de escritura *uudemon.*** Diseñado para mejorar periódicamente el mantenimiento del sistema *UUCP*, existen tres versiones separadas de este "*daemon*" en la Versión 2, todos son llamados uudemon y tienen un prefijo que indica la periodicidad en que deben ejecutarse `.hr` para cada hora; `.day` para hacerlo del diario y `.wk` semanalmente. En el sistema HoneyDanBer se ha añadido otro para soportar el monitoreo del sistema remoto. Los otros tres "*daemons*" fueron reescritos y renombrados. El `uudemon.hour` invoca al *uushed* para procesar una solicitud *uucp*

  El `uudemon.poll`, y el `uudemon.admin` envia información sobre el estado del *uucp* al administrador. Por ultimo el `uudemon.cleanu` elimina de los directorios de *uucp* los archivos de sesión. Ambas versiones uudemon son invocadas por *cron*.

- ***uuclean.*** La Versión 2 de *uuclean* provee un espacio en disco para ser usado por el subsistema *UUCP*, cuando es invocado por el `uudemon.hr` o `uudemon.wk`; Un registro de archivos borrados es mantenido en `/usr/spool/uucp/LOGDEL`. en la versión HoneyDanBer *uncleanup* invoca al `uudemon.cleanu` y mantiene un diario en `/usr/spool/uucp/.Admin/cleanup`.
- ***uucico.*** [*UNIX to UNIX copy-in copy-out*] Este programa generalmente es invocado en forma automática por el sistema *UUCP* en *backgroun*, transporta archivos que han sido encolados por el *uucp*, *uux* y los programas de *mail*. Existe la posibilidad de invocarlo en forma manual para iniciar una transferencia o para poner al subsistema *UUCP* en condiciones óptimas. La opción `debug`, posee diferentes niveles de `debugging` para salidas voluminosas.

  El programa es inicializado por el sistema con un papel de MAESTRO cuando inicializa una conexión en un sistema remoto. Así, el programa o usuario que invoque *uucico* especificando la opción `-rl` lo utilizará en el papel de MAESTRO; y otro uucico será inicializado con el papel de ESCLAVO en el sistema remoto. Desde luego debe tener permiso para ser ejecutado en la maquina remota y no debe ser ejecutado con la opción `-rl`. Los dos "*daemons*" inician de esta forma una conversación, manteniendo un control de información y envío de esta.
- ***uuxt.*** Este "*daemon*" es invocado por uucico en el sistema remoto, para responder a una solicitud de servicio a un comando por el programa *uux* en la máquina local. Primero busca un directorio de *spool* para depositar en un archivo la solicitud; al terminar verifica que todos los archivos de datos se encuentren disponibles y el permiso para ejecutar la petición de la solicitud.

La figura 5-1 muestra "la jerarquía de comandos" de *uux* a *uucico* MAESTRO al llamar al *uucico* ESCLAVO y por ultimo al *uuxqt* en el sistema remoto.

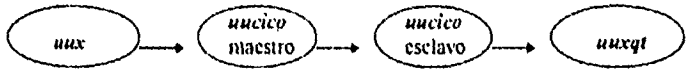


FIGURA 5-1. La jerarquía de los comandos

Al igual que en *uucico* se puede especificar la opción de debug cuando se invoca *uuxqt* para poner al sistema en diferentes opciones de optimización. El sistema *UUCP* usa varios directorios. La tabla 5-2 muestra los directorios usados por el sistema *UUCP* y una breve descripción de ellos.

Directorio	Descripción
`/usr/bin`	Directorio de Comandos de usuario
`/usr/spool/uucp`	
`/usr/spool/locks`	Directorio de
`/usr/lib/uucp`	Directorio de librerías
`/usr/spool/uucppublic`	Directorio *spool*- solicitudes y archivos de sesion (solo para el HoneyDanBer)

TABLA 5-2. Directorios usados por el sistema *UUCP*

UUCP utiliza gran cantidad de archivos unos muy importantes, otros un poco menos y otros dependiendo de las circunstancias. Sería ideal poder describir en esta tesis cada uno de ellos sin embargo en el apéndice D.1 se presentan dos tablas con algunos de los más importantes de la Versión 2 y de la honeydanber.

5.3 CONCEPTOS DE UUCP

La red de máquinas *UUCP* no tiene una administración central, esto significa que no existe computadora que gestione la red entera. El usuario se une a la red negociando con otra máquina que ya esté en la red *UUCP* y que este de acuerdo en ser su vecina, en el sentido de que esa computadora consienta en añadir información de configuración para establecer una comunicación con su maquina.

Una vez que ha encontrado un vecino, puede entonces conectarse a través de esa máquina a todas las otras máquinas que quieran conectarse con ella; para comunicarse con una computadora remota no conocida por su propia computadora, necesita una ruta hasta esa computadora. El problema de encontrar tal ruta está parcialmente mitigado por la existencia del mapa *UUCP*, que especifica conexiones entre varias computadoras.

El mapa *UUCP* es mantenido por el proyecto de red *UUCP*, que remite este mapa cada mes en el grupo de noticias `comp.mail.maps` sobre *USERNET* (discutida posteriormente en este capitulo). Cada nodo principal de la red *UUCP* paga los costes de los enlaces de conexión con las otras máquinas.

Sin embargo, el subsistema *UUCP* globalmente es mucho más potente, en parte debido a que permite ejecución remota de órdenes arbitrarias. En consecuencia, existen extensas protecciones de seguridad dentro del subsistema *UUCP*; muchas de las órdenes y acciones discutidas aquí estarán prohibidas en algunas máquinas, pero permitidas en otras.

Es importante tener en cuenta que muchas de las características más potentes de *UUCP*, pueden estar desactivadas en sistemas o redes especificas por razones de seguridad. A lo largo de los años, la seguridad *UUCP* se ha hecho más y más restrictiva como resultado de amargas experiencias dentro de la comunidad del sistema *UNIX*.

Solo a los entornos de máquinas más amistosos, en donde no hay acceso a lineas telefónicas publicas o *modems* de llamadas, pueden confiárseles las capacidades más potentes de *UUCP*.

Después de conocer acerca de los componentes del sistema *UUCP*, vamos a ver a través de un ejemplo como trabajan todos en una aplicación en común al enviar una correspondencia a un sistema remoto.

La siguiente figura es un diagrama esquemático de los componentes y su interacción para enviar correspondencia aun sitio remoto. Por comodidad lo analizaremos a través de 20 pasos, cabe hacer notar que el número de los pasos es arbitrario y depende del nivel de detalle que se desee considerar.

Este ejemplo será analizado tanto para la Versión 2, como para la HoneyDanBer. Antes de iniciar la explicación recordemos que los archivos *system*, *devices* y *dialcodes*, pertenecen al sistema Versión 2 y los mensajes son escritos en el directorio `/usr/spool/uucp/LOGFILE`, mientras que los `L.sys`, `L-devices` y `L-diacodes`, pertenecen al HoneyDanBer y los mensajes son escritos en el directorio `/usr/spool/uucp/.Log`.

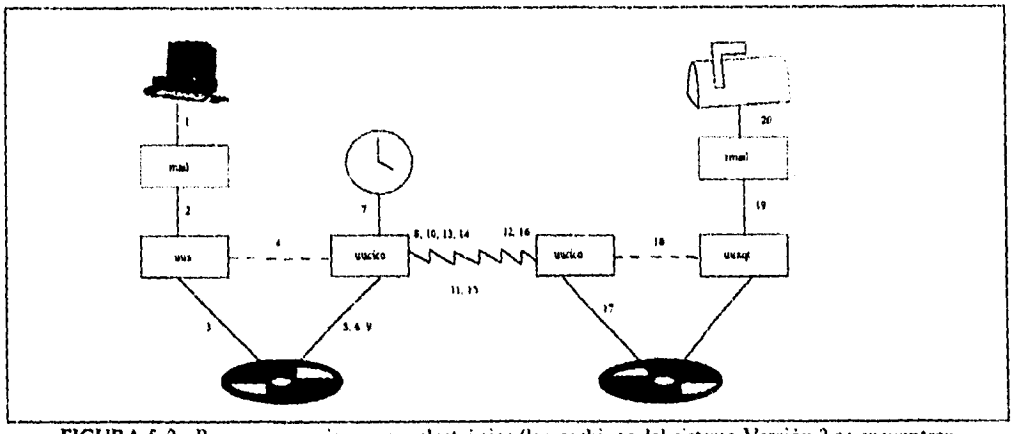


FIGURA 5-2. Pasos para enviar correo electrónico (los archivos del sistema Versión 2 se encuentran encerrados por paréntesis).

1. El usuario local invoca el *mail*.
2. El *mail* invoca a su vez al *uux*.
3. El *uux* local encola la solicitud.
4. El *uux* local se inicializa como MAESTRO.
5. El *uucico* local busca el archivo de trabajo y
6. Crea un archivo por sitio remoto.
7. Verifica el archivo `L.sys` (*system*) para el tiempo de la conexión.
8. Revisa el archivo `L-devices` (*devices*) para determinar el puerto.
9. Crea un archivo de bloque para la comunicación del puerto.
10. Contacta o llama al sistema remoto.
11. Inicia la secuencia de identificación entre ambos sistemas.
12. El *uucico* remoto se inicializa como ESCLAVO.
13. Inicia la secuencia de intercambio entre *uucicos*.
14. Se inicia el protocolo de negociación entre *uucicos*.
15. El *uucico* MAESTRO envía archivos de solicitud.
16. El MAESTRO y el ESCLAVO intercambian sus papeles.
17. Son enviadas las solicitudes pendientes del sistema remoto al local.
18. El *uucico* remoto inicializa el *uuxqt*.
19. El *rmail* remoto es invocado para entregar el mensaje.
20. El mensaje es entregado en su *mailbox*.

Una explicación más detallada de cada uno de estos pasos se encuentra en el apéndice D.2, se sugiere que sea leído para una mayor comprensión de cada uno de los pasos.

5.4 TRANSFERENCIA DE ARCHIVOS CON uuto

Los archivos pequeños en *ASCII* pueden ser enviados más rápidamente como correo. Sin embargo, es molesto recibir mensajes o archivos grandes utilizando *mail*. Cuando un mensaje es grande, se desplaza a través de la pantalla antes de que el usuario tenga la oportunidad de salvarlo o borrarlo. Los largos memoranda, manuscritos o artículos no pueden ser manejados de este modo.

Es mucho más fácil enviar el mensaje de otra forma y enviar una notificación a través de *mail*. El subsistema *UUCP* proporciona varias facilidades para copiar archivos entre computadoras. El modo más fácil de mover archivos entre máquinas es con la orden **uuto** (de *UNIX* a *UNIX* [*UNIX to UNIX*]), y la orden *mupick* (recoger de *UNIX* a *UNIX* [*UNIX* to *UNIX pickup*]) para recuperar archivos enviados desde otra máquina. La orden uuto toma una lista de archivos y una dirección remota como argumentos, como se muestra aquí:

```
$ uuto datos1 datos2 remota!identif
```

La dirección es el último argumento; va a continuación de todos los nombres de archivos. La dirección toma la misma forma aquí que para mail: un nombre de máquina remota y un id de presentación separados por un carácter *!* (admiración). A diferencia de *mail*, *uuto* solo permite un único destino, de modo que no puede enviarse el mismo archivo a varios receptores con una única línea de orden. La orden *uuto* puede ir seguida de tantos nombres de archivos como se necesite, pero la dirección debe ser el último argumento.

Es necesario aclarar que la orden *uuto* no transfiere realmente archivos, sino que los pone en cola para que sean transferidos por el sistema *uucp* cuando este pueda. El subsistema *uucp* puede demorar la transferencia durante horas o días, de modo que los archivos pueden no alcanzar su destino durante algún tiempo; si la transferencia es colocada en la cola correctamente, uuto vuelve al shell tras un segundo o dos y se puede continuar la sesión. La orden *uuto* puede aceptar dos opciones de línea. La opción `-m` (correo [*mail*]) hace que *uuto* envíe correo electrónico al usuario remitente y al destinatario, cuando se complete la transferencia de datos. Por ejemplo,

```
$ uuto -m report jersey!abby
```

envía el archivo `report` (sobre el sistema local arizona) al usuario *abby* en *jersey* y envía por último el correo cuando esta transferencia se completa, además de enviar la notificación al usuario *abby*. El correo tendrá el siguiente aspecto:

```
$ mail >From uucp sun Feb 4 oo: ss EST 1990 remote from arizona REQUEST:
arizona ! khr/report --> jersey ! -/receive/abby/arizona/ (abby) (SYSTEM
jersey) copy  succeeded
```

La opción `-p` hace que *uuto* copie el archivo durante el proceso de puesta en cola (espera). Si no se utiliza la opción `-p`, uuto requiere que el archivo retenga el mismo nombre y no sea suprimido hasta después de que se complete la transferencia. Si el archivo se modifica entre el momento en que se pone en cola y el momento en que se envía, se enviara la versión modificada. Si se utiliza la opción `-p`, el contenido del archivo en el momento en que fue puesto en cola será el que se transmita, y el archivo original puede ser trasladado, suprimido o modificado. Puesto que la opción `-p` hace que el archivo sea duplicado, utiliza cierto espacio de disco adicional. Esta puede ser una consideración si se están transfiriendo archivos muy grandes.

A diferencia de la orden *mail*, *uuto* no permite direccionamiento multipaso. Si no existe una ruta directa entre las máquinas fuente y destino, se utiliza *uuto* para pasar los archivos a una tercera máquina que conozca a ambas y tenga un amigo allí para reexpedir los archivos a su destino final. Por ejemplo:

```
$uuto memo sysul3!armando
```

En el ejemplo anterior uuto transfiere el archivo memo al usuario armando en el sistema *sysul3*, en el directorio raíz del usuario. También puede encaminar archivos a través de una serie de sistemas intermedios si se conoce la ruta a un sistema remoto.

5.5 RECEPCIÓN DE ARCHIVOS CON uupick

Cuando los archivos han sido transferidos, el sistema *uucp* de la máquina receptora enviará correo electrónico al receptor, anunciando que han llegado archivos procedentes de la otra maquina. Cuando recibe ese correo, el receptor puede usar la orden *uupick* para copiar los archivos desde el directorio en donde *uucp* los ha dejado, al directorio actual del receptor. La orden *uupick* se utiliza generalmente sin argumentos. Esta orden localizará los archivos enviados via *uuto*, y luego mostrara un inductor solicitando la disposición a tomar con cada archivo del modo siguiente:

```
$ uupick
from system uname: file datos1 ?
```

Se visualizan el *uname* de la máquina remota junto con el nombre del archivo `datos1` y luego *uupick* espera una orden. Por ejemplo, la ejecución de *uupick* puede dar

```
$ uupick from system mozart: file fugue ?
```

5.5.1 Algunas opciones de uupick

La orden * (asterisco) hará que *uupick* muestre un resumen de las órdenes disponibles. La más útil es m (mover), que hace que el archivo designado sea trasladado a otro directorio. La orden m puede tomar un nombre de directorio como argumento, en cuyo caso el archivo es trasladado a ese directorio en lugar de al directorio actual,

```
$ uupick
from system uname: file datos1 ? m /tmp
642 blocks
from system uname: file datos2 ?
```

después se visualiza el tamaño del archivo y luego *uupick* continúa con el resto de los archivos. Si todos los archivos han sido transferidos, *uupick* terminara regresando al *shell*. Se utiliza `m.` (m punto) para trasladar un archivo al directorio actual. La orden a (todos [*all*]) es análoga en función a m, pero hace que todos los archivos procedentes de la máquina remota sean trasladados de una vez. Esto puede ser más rápido que m si se han transferido muchos archivos. La orden d (suprimir [delete]) hace que el archivo sea suprimido en vez de trasladado. La orden p (imprimir [*print*]) hace que el archivo sea visualizado, pero debe ser utilizada con cuidado, ya que pueden haberse transferido archivos binarios con *uuto*.

La orden *q* (abandonar [*quit*]) hace que *uupick* finalice; los archivos que no hayan sido trasladados o suprimidos permanecerán en el directorio de *spool* hasta que se ejecute *uupick* de nuevo. Finalmente, la orden !orden (admiración orden) es un escape al *shell* que ejecutara la orden en un *subshell* y luego regresará a *uupick*. Esta orden suele utilizarse para crear directorios o modificar permisos antes de trasladar los archivos. Por ejemplo, la sesión *uupick* podria parecer como esta,

```
$ uupick
from system mozart: file fugue ? m /music
4 blocks
from system mozart: file symphony ? m
38 blocks
from system mozart: file opera ?
from system beethoven: file symphony ?
from system vivaldi: file concerti ? a
116 blocks
$
```

Por último la orden uupick puede tener una opción de linea `-s` (sistema), seguida de un nombre de sistema especifico. Esto restringe la búsqueda de *uupick* a los archivos transferidos desde la máquina designada. Tanto *uuto* como *uupick* son generalmente guiones *shell*, de modo que pueden ser inspeccionados en `/usr/bin` si se desea obtener más información sobre su modo de funcionamiento. Por ejemplo, para ver solo aquellos archivos que le han sido enviados mediante *uuto* desde el sistema *mozart*, utilice

```
$ uupick -s mozart
```

5.6 TRANSFERENCIA DE ARCHIVOS CON uucp

La orden *uuto* proporciona el acceso más sencillo al subsistema *UUCP*, para la mayoría de las transferencias de archivos. Sin embargo, la orden *uucp* puede proporcionar más control sobre transferencia de datos en situaciones especificas. A diferencia de uuto, la orden *uucp* permite solicitar un archivo de una máquina remota para copiarlo en la máquina local. El formato de la orden *uucp* es :
$ uucp <archivo.fuente> <archivo.destino>

La orden *uucp* copiará la lista de argumentos `archivo.fuente` al `archivo.destino`. Estos archivos son realmente direcciones en el formato familiar *máquina!destino*, excepto que el destino no es un *id* de presentación de usuario, sino el nombre del directorio a copiar. En la sintaxis de *uucp*, la parte *maquina!* puede omitirse si se refiere a la máquina local. Por ejemplo,

```
$ uucp /etc/profile susis!/var/spool/uucppublic/profile
```

El ejemplo anterior copiará el archivo profile de la máquina local al directorio `var/spool/uucppublic` en la máquina `susis`. También se permite copias de múltiples archivos si el destino es un directorio.

```
$ uucp $HOME/* susis!/var/spool/uucppublic/xfer.dir
```

En este ejemplo, todos los archivos del directorio $HOME se transferirán al directorio /var/spool/uucppublic/xfer.dir en la máquina susis. Además, *uucp* puede copiar archivos desde la máquina remotu hasta la máquina local si se incluye la parte maquina! en la lista de archivos fuente, como se muestra aqui:

```
$ uucp "uname!/tmp/hola/*" /var/spool/uucppublic/xfer
```

Esto moverá todos los archivos del directorio /tmp/hola en el sistema remoto al directorio /var/spool/uucppublic/xfer en el sistema local. La lista fuente esta acotada para impedir que el shell local expanda el carácter *; este será expandido en el sistema remoto. También puede incluirse la parte maquinal tanto en la lista de archivos fuente como en la lista de archivos destino del modo siguiente:

```
$ uucp maquina1!archivo maquina2!archivo
```

En este caso la orden *uucp* realizara una transferencia a través de terceros de una máquina a otra. Ninguna de ambas máquinas necesita ser local suponiendo que la seguridad *uucp* este administrada para permitirlo. También se puede utilizar *uucp* para enviar archivos a una máquina que no está conectada directamente a la local, si se conoce una ruta que conecte a ambas. Por ejemplo, cuando usted ejecuta la orden

```
$ uucp memo alpha!beta!gamma!ferdie!/var/home/dan/memo
```

uucp contacta con *alpha*, y le proporciona el archivo después se contacta con gamma y con ferdie e intenta poner el archivo memo en el directorio /vur/home/dan. Cada máquina solo necesita conocer a la siguiente máquina de la cadena.

5.6.1 Nombres de caminos lógicos

Generalmente la seguridad *uucp* está administrada de modo que las copias de archivos solo se permiten desde algunos directorios públicos específicos como /var/spool/uucppublic. Por tanto, el mecanismo de nombre de camino no funcionará generalmente como se describió con anterioridad. Para obviar estas limitaciones de seguridad, la orden *uucp* permite algunas formas de denominación lógica de caminos. Si la porción de nombre de archivo es **~usuario** (tilde usuario), donde usuario es un id de presentación en la máquina destino, *uucp* creara el archivo en el directorio propio de ese usuario. Por ejemplo, muchos sistemas permiten que cada usuario cree un directorio público de nombre `rje` (por entrada remota de trabajos [*remote job entry*]) dentro de su directorio raiz. La orden *uucp*

```
$ uucp fich.Local mi_sis!~steve/rje
```

escribirá el archivo fich.local en el directorio rje del usuario steve en la máquina `mi_sis`. Un nombre de archivo que comience con ~/ (tilde diagonal), tal como ~/destino, escribirá el archivo en el directorio público *uucp* estándar en la máquina remota. Este directorio es generalmente `/var/spool/uucppublic`, de modo que la orden

```
$ uucp archivo.local mi_sis!~/steve
```

copiara `fich.local` a `/var/spool/uucppublic/steve` en la máquina `mi_sis`. Esto funciona para la transferencia de un archivo, pero si se va a transferir más de un archivo, el `~/destino` debería finalizar con una / al final, que instruya a *uucp* para que cree un directorio con el nombre indicado y para que copie todos los archivos fuente en ese directorio, reteniendo sus nombres originales. Por ejemplo:

```
$ uucp $HOME/* sistema!~/steve/
```

Realmente esta orden es muy parecida a la operación *uuto*, excepto que uuto utiliza un convenio de denominación más complejo para el directorio destino, construido a partir del nombre de la máquina local y del *id* de presentación del receptor sobre el sistema destino.

5.6.2 Opciones en línea para la orden uucp

uucp permite muchas opciones en la línea de orden que modifican su función. Muchas de ellas son utilizadas principalmente por expertos en *uucp* para la operación de transferencia de datos, pero otras son de utilidad general. La opción `-m` (correo [*mail*]) instruye a *uucp* que envíe un mensaje al peticionario cuando se complete la transferencia. La opción `-n` (notificar) toma un id de presentación como argumento sobre la máquina remota y envía correo a ese usuario en la máquina remota cuando se completa la copia. La opción `-C` (copiar) copia un archivo en el directorio público de *uucp* antes de que sea transferido, de modo que sea posible eliminar o modificar el archivo original. Si no se emplea esta opción, el archivo no puede ser suprimido ni renombrado hasta que la transferencia se complete.

La opción `j` (trabajo *rjob*]) hace que *uucp* muestre un número de identificación de trabajo único cuando coloca el trabajo en la cola. Este número de identificación es útil para seguir la pista al éxito o fracaso de la transferencia. La opción `-x` lleva un dígito entre 1 y 9 como argumento y hace que se visualice información de depuración en la terminal mientras mayor sea el numero, más amplia será la salida de depuración que aparecerá. Finalmente, puede evitarse que *uucp* intente enviar inmediatamente un trabajo después de que haya sido puesto en cola con éxito. Si no se desea que la conexión con la máquina remota comience inmediatamente, se utiliza la opción `-r`; hará que el trabajo sea puesto en cola, pero no iniciará la transferencia. En vez de ello, *uucp* intentará enviar el archivo durante su siguiente ciclo de planificación regular, que generalmente sucede alrededor de cada hora. La opción `-r` podría ser utilizada cuando se sepa que la máquina destino no está operando, o que la conexión es inoperable por alguna razón.

5.7 EJECUCIÓN DE COMANDOS EN MÁQUINAS REMOTAS CON uux

La orden más potente disponible dentro del subsistema *UUCP* es *uux* (ejecutar de *UNIX* a *UNIX* [*UNIX to UNIX execute*]). La orden *uux* permite la generación de líneas de orden que se ejecutarán en una máquina remota. Puede recolectar archivos designados en la línea de orden procedentes de varias máquinas y luego ensamblar la orden en la máquina destino y ejecutarla allí. Debido a estas notables capacidades, *uux* es muy peligroso y su uso esta generalmente limitado en la mayoría de los sistemas. Cuando está permitido, puede ser muy útil en entornos de red en donde el movimiento de archivos entre máquinas es rápido y eficiente.

La orden *uux* lleva una línea de orden normal como argumento, con una excepción: el nombre de la orden y cualesquiera nombres de archivos especificados como argumento, pueden ir precedidos con el prefijo *maquina admiración*. Este prefijo instruye a *uux* a ejecutar la orden en la máquina especificada o a recoger los archivos indicados de la máquina designada. Por ejemplo, la orden

```
$ uux "sis1!cat sis2!/etc/profile sis3!/etc/rc2 > !/tmp/salida"
```

recogerá el archivo `/etc/profile` de la máquina *sis2* y el archivo `/etc/rc2` de la máquina *sis3* y los trasladará a la máquina *sis1*, en donde se ejecutará la orden `cat`. La salida esta redirigida al archivo `/tmp/salida`, sobre la máquina local.

Cualquier parte de la línea de orden que solo tenga una !, sin designación de maquina, se interpreta como referida al sistema local. La línea de orden entera esta entrecomillada para impedir que el shell local interprete el carácter de redirección > antes de que *uux* lo vea. En líneas de cauce multiorden, las órdenes que vienen tras la primera no pueden llevar el prefijo `máquina!`, ya que *uux* requiere que todas las partes de la línea de cauce sean ejecutadas en la misma máquina. Por ejemplo, dada la orden

```
$ uux "sis1!cat !/etc/profile | grep HOME > !/tmp/sal"
```

uux tomará el archivo `/etc/profile` del sistema local y lo trasladará a la máquina *sis1*, en donde se ejecutara la linea de cauce `cat | grep`. La salida será redirigida al archivo `/tmp/sal` en la máquina local. Puede utilizarse el carácter especial (menos) dentro de la orden *uux* como parte de un nombre de archivo, en cuyo caso *uux* leerá su entrada estándar en lugar de ese archivo. La orden anterior también podria haber sido escrita del modo siguiente:

```
$ cat /etc/profile | uux "sis1!cat I grep HOME > !/tmp/sal"
```

En las órdenes *uux* se prefiere el uso de nombres de caminos completos, pero los operadores especiales `~usuario` y `~!camino` funcionaran en las partes de nombres de archivos de las órdenes *uux* del mismo modo que lo hacen en las órdenes *uucp* discutidas anteriormente. La orden *uux* fallará si alguno de los archivos no es encontrado donde se espera, o si las órdenes no están permitidas dentro de las restricciones de las reglas de seguridad *uucp* en la máquina destino. Cuando *uux* falla, envia correo al usuario que solicitó la acción, en la máquina original en donde uux fue ejecutado.

La orden *uux* puede llevar varias opciones en la linea de orden. La opción `-c` (copiar) hace que *uux* copie los archivos locales al directorio de `spool`. Esto permite mover o suprimir los archivos designados antes de que se complete la transferencia de datos. La opción `-n` (notificar) hace que *uux* no envíe correo de notificación si la orden falla. Esto es útil para trabajos automáticos o subordinados, tales como la recolección regular de algunos datos procedentes de las máquinas de una red de área local.

La opción `j` (trabajo [*job*]) hace que uux muestre un *id* de trabajo cuando ponga en cola la acción, y la opción `-r` hace que el trabajo sea puesto en cola pero no sea iniciado inmediatamente. La opción `-z` envia correo a quien originó el trabajo; si el trabajo tiene éxito normalmente uux permanecerá en silencio. Finalmente, la opción `-x` toma un digito entre 1 y 9 como argumento y hace que aparezca el correspondiente nivel de salida de depuración cuando el trabajo sea puesto en cola. Por ejemplo, si tiene acceso a un sistema con una impresora de alta calidad, puede imprimir un archivo utilizando la impresora remota con la orden

```
$ uux @jersey!lp -dlaser !memo@
```

que envía el archivo memo desde el sistema local para ser impreso en *jersey* utilizando la orden `lp` con la opción *d*laser.

5.8 ESTADO Y CONTROL DE JOBS CON uustat

Puesto que el subsistema *uucp* efectúa sus actividades de transferencia de archivos en modo subordinado, no hay modo inmediato de conocer si las transferencias van teniendo éxito o fallan. Por tanto, la orden *uustat* (*status* de *UNIX* a *UNIX*) está disponible para informar del estado de la cola de transferencia de *uucp*, la orden *uustat* proporcionará un breve resumen de los trabajos por enviar, o del estado de las comunicaciones de las máquinas con las cuales la local interacciona, entre otras posibilidades. Por omisión, *uustat* informa solamente de los trabajos creados por el usuario que la ejecuta, ejemplo :

```
$ uucp /etc/profile mi_sis!~/root
$ uustat
mi_sisN74d6 06/27-18:58 S mi_sis root 4290 /etc/profile
```

En este caso, el trabajo `mi_sis` tiene el *id* N74d6 y fue puesto en cola a las 6:58 P.M. el 27/6, está dirigido a la máquina `mi_sis` y fue creado por el usuario *root*. El tamaño es de 4290 *bytes*, y el archivo a transferir es `/etc/profile`. La letra `S` significa que será enviado como correo, también es posible una `R`, que indica que el archivo tiene que ser recibido. Un trabajo en cola puede tener más de una tarea bajo el mismo id de trabajo, como en este ejemplo:

```
$ uustat
mi_sisN74d7 06/29-17:47 S mi_sis root 62 D.tune2317f15
      06/29-17:47 S mi_sis root rmail jim
$
```

el ejemplo anterior es la respuesta de una orden *mail* con un destinatario remoto. De hecho, la orden *mail* envía el mensaje de correo como un archivo de datos, que es el primer archivo en la salida de *uustat*. El segundo archivo es una orden `uux` para ejecutar `rmail jim` en la máquina destino para entregar el mensaje. Si se envían múltiples archivos vía uuto o *uucp*, aparecerán múltiples archivos por cada id de trabajo. Si se pone en cola más de una orden *uucp*, habrá más de un id de trabajo en la salida de uustat.

Cuando se completa la transferencia de datos, el *id* de trabajo es suprimido de la lista *uustat*, y si no quedan trabajos, *uustat* terminara silenciosamente. Por tanto, la ausencia de salida de *uustat* son buenas noticias; dice que todos los trabajos de la cola han sido ya enviados.

uustat, por omisión informa únicamente de los trabajos puestos en cola por el usuario que inicia la orden; la opción `-a` (todos [*all*]) ordena a *uustat* para visualizar todos los trabajos en cola de la máquina, no importa quien los formó. Además, la opción `-u` (usuario) toma un *id* como argumento y produce un listado de los trabajos en cola planificados por el usuario nombrado solamente. Por ejemplo:

```
$ uustat -u steve
```

5.8.1 Información sobre máquinas específicas

La orden *uustat* puede mostrar la cola para una máquina específica con la opción `-s` (sistema). Es necesario introducir un *id* de máquina detrás de la opción `-s`, como se muestra aquí:

```
$ uustat -s mi_sis
```

Esta orden producirá el listado de los trabajos en cola para la máquina *mi_sis*. La orden *uustat* también puede producir un informe sobre el estado de las transferencias desde la máquina local hasta cualesquiera máquinas remotas que hayan sido contactadas recientemente. Para este propósito se utiliza la opción `-m` (máquina) del modo siguiente:

```
$ uustat -m
mi_sis 1C 06/29-17:47 SUCCESSFUL
sis2 1C 06/29-18:20 TALKING
$
```

En este ejemplo, dos máquinas han sido contactadas recientemente. La máquina *mi_sis* fue contactada la última vez a las `17:47` del `29/6` y la conexión se completo con éxito. La máquina *sis2* esta actualmente conectada con la máquina local y se están transfiriendo datos.

Finalmente, la orden *uustat* permite suprimir un trabajo de la cola *uucp* no enviado. Esto podría ser necesario en caso de cambiar de idea respecto a una transferencia de datos, o cuando una conexión no se esté comportando correctamente. Todas las transferencias *uucp* están disponibles desde *uustat*, incluyendo las creadas por *mail* o *uux*. Para suprimir un trabajo, debe conocerse su *id*, el cual se visualiza cuando el trabajo se pone originalmente en la cola o mediante *uustat*. Para eliminar un trabajo se utiliza la opción `-k` (eliminar [*kill*]). Esta opción lleva el *id* de trabajo como argumento. Por ejemplo:

```
$ uustat -k mi_sisN74d7
Job: mi_sisN74d7 successfully killed
$
```

Sólo se permite un id de presentación en cada invocación de la orden uustat `-k`. Si un trabajo se ha completado, no puede ser eliminado y *uustat* devolverá un error, como se muestra aquí:

```
$ uustat -k mi_sisN74d7
Can't find Job mi_sisN74d7; Not killed
$
```

Además, uustat no puede eliminar un trabajo que está en progreso.

5.8.2 Otras opciones uustat

Si desea ver cuanto tiempo tardara en enviarse algo a un sistema remoto, utilice la orden *uustat* con las opciones `-t` [sistema] y `-c`. La opción `-t` [sistema], permite ver la tasa de transferencia [*transfer rate*] o el tiempo de cola [*queue time*] para un sistema especifico; la opción comprueba el tiempo de cola. Por ejemplo:

```
$ uustat -twong -c average queue time to wong for last 60 minutes: 0.07
minutes data gathered from 01:20 to 02:20 GMT
```

Para comprobar la tasa de transferencia de datos, utilice `-t sistema` sin la opción `-c`. Por ejemplo:

```
$ uustat -twong
average transfer rate with wong for last 60 minutes: 206200.00 bytes/sec
data gathered from 01:26 to 02:26 GMT
```

5.9 ESTADO DE UUCP CON uulog

La orden *uulog* visualiza un registro de información para todos los sistemas que han sido contactados desde la última limpieza semanal de datos de *uucp*, esta información es una historia completa de todos los accesos a la máquina. Esta información es almacenada en base a cada máquina, de modo que la orden *uulog* requiere un nombre de sistema como argumento, como en la Figura 5-2. Este ejemplo muestra el registro de la máquina local *mi_sis* para tráfico con la máquina remota susis.

En la figura 5-2 se han representado tres llamadas, las dos primeras iniciadas por susis. La primera falló (*<<CAN'T ACCESS DEVICE>>*) debido a que el *modem* no estaba conectado. Las otras tuvieron éxito, y cada una de ellas dio lugar a la transferencia de dos archivos. La llamada comienza con *<<SUCCEEDED>>*, que indica que la conexión fue efectuada correctamente; luego *<<startup>>* indica que el programa *uucico* está comunicando. La llamada finaliza en *<<conversation complete>>*. La tercera llamada de la Figura 5.3 fue una llamada procedente de la máquina remota a la máquina local. En este caso, la línea *SUCCEEDED* no aparece y el registro comienza con la línea <<startup>>. La máquina remota solicita el envío de dos archivos a la máquina local (*<<REMOTE REQUESTED>>*).

```
$ uulog susis
uucp susis (4/11-18:56:13,357,0) TIMEOUT (generic open)
uucp susis (4/11-18:56:13,357,0) CONN FAILED (CAN'T ACCESS DEVICE)
uucp susis (4/11-19:20:32,381,0) SUCCEEDED (call to susis process job grade Z )
uucp susis (4/11-19:20:36,381,0) OK (startup)
root susis susisZ4d9b (4/11-19:20:37,381,0) REQUEST (mi_sis!D.mi_si4d9dfe9
--> susis!D.mi_si4d9dfe9 (root))
root susis susisZ4d9b (4/11-19:20:38,381,1) REQUEST (mi_sis!D.susis4d9be23
--> susis!X.susisA4d9b (root))
uucp susis (4/11-19:20:40,381,2) OK (conversation complete ttyO0s 57)
uucp susis (4/12-18:15:05,321,0) OK (startup)
uucp susis (4/12-18:15:06,321,0) REMOTE REQUESTED (susis!D.susisOd75e46
--> mi_sis!D.susisOd75e46 (root))
uucp susis (4/12-18:15:07,321,1) REMOTE REQUESTED (susis!D.mi_si2636c80
--> mi_sis!X.mi_sisN2636 (root))
uucp susis (4/12-18:15:08,321,2) OK (conversation complete ttyO1s 4)
$
```

FIGURA 5-3. Salida típica de la orden *uulog*

El registro puede ser muy largo si dos máquinas conversan activamente, pero es suprimido durante la limpieza semanal de *uucp*. Con suerte, un registro largo de una máquina sospechosa puede proporcionar gran cantidad de información muy útil para comprender por que los archivos no alcanzaron su destino como, era de esperar.

5.10 CONEXIÓN A UNA MÁQUINA REMOTA CON cu

La orden cu (llamar a *UNIX*) (*call UNIX* permite presentarse en otro sistema y utilizar el sistema remoto mientras aún se tiene sesión abierta en el sistema propio. Cuando se utiliza cu, el sistema local entra al sistema remoto utilizando una línea de comunicación dedicada, una red de área local o un *modem*, dependiendo de la configuración de ciertos archivos).

Cuando se utiliza cu, primero se invoca al sistema remoto y se crea una conexión con él; los sistemas remotos conocidos al sistema propio pueden ser invocados por nombre. También se puede conectar a un sistema remoto instruyendo al sistema sobre el modo de realizar la conexión.

Usted puede invocar a un sistema conocido por el suyo mediante nombre (listado en la salida de la orden *uuname*) utilizando la orden cu con el nombre del sistema como argumento. No necesita conocer como están conectados los sistemas, ya que su computadora tiene un archivo, `/etc/uucp/Systems`, que mantiene información referente a las conexiones de su sistema con el resto de sistemas. Si hay muchas conexiones entre sistemas, `cu` tiene capacidad para elegir entre varios medios de establecer la conexión. Por ejemplo, puede utilizar cu para conectarse al sistema *alpha*, conocido por su computadora, utilizando la orden siguiente:

```
$ cu alpha
Connected login:
```

La utilización de cu para contactar con el sistema *alpha* hace que se compruebe el archivo `Systems` y se establezca una conexión de acuerdo con la configuración del archivo `Systems`. Una vez que el sistema se ha conectado con *alpha*, aparecerá el mensaje `Connected`, seguido del indicador `login:` y el procedimiento normal de presentación sobre alpha. Si trata de contactar con un sistema que no está incluido en el archivo `Systems` obtendrá un mensaje como este:

```
$ cu beta
Connection failed: SYSTEM NOT IN Systems FILE.
```

En la Versión 4 la orden cu ha sido mejorada para que reconozca caracteres de 8 *bits* y caracteres *multibyte*. Si se especifica un número telefónico como argumento de `cu`, se seleccionará una unidad de llamada automática (*ACU*) *[automatic calling unit]* y el sistema marcará el número telefónico.

Un número telefónico válido para cu es una secuencia formada por dígitos del `0` al `9`, los símbolos `*` y `#` (del teclado telefónico) y los simbolos `=` y `-`. El símbolo = instruye a `cu` para que espere un tono de marcar secundario antes de marcar el resto de la secuencia: el símbolo `-` indica una pausa (de cuatro segundos) antes de seguir marcando. Supongamos que usted marca el 9, espera un tono de marcar para una línea externa y luego marca `12015551234` para conectar con *alpha*. Usted llama a *alpha* utilizando la orden:

```
$ cu 9=12015551234
```

Si necesita marcar un `@9` antes de alcanzar una línea externa, utilice la orden siguiente:

```
$ cu "@9=12015551234@
```

Los signos de acotación son necesarios para que el `@` no sea interpretado por parte del *shell*. Si marca el número de un sistema pero no logra crear una conexión, obtendrá un mensaje de error. Por ejemplo:

```
$ cu 9=12015551234
Connect failed: CALLER SCRIPT FAILED
```

-s velocidad	Especifica la velocidad de transmisión (300, 1.200, 2.400, 4.800 o 9.600 bits por segundo) a utilizar para la conexión. El valor por omisión es "*Any*" ("Cualquiera"), en el que el valor dependerá de la velocidad registrada en el archivo /ele/uucp/Devices
c tipo	Especifica la red de área local a utilizar. El valor de tipo se toma del primer campo del archivo /etc/uucp/Devices. (En la Versión 4 la opción -c también puede ser utilizada para especificar una clase de líneas.)
línea	Especifica el dispositivo a utilizar como línea de comunicaciones. Esta opción anula la selección de dispositivo desde el archivo Devices establece paridad par. Establece paridad impar.
b	Establece *semi-duplex*.
@ n	Establece *n (7 u 8 bits)* caracteres.
-n	Solicita un número telefónico. Esto proporciona una seguridad adicional al usuario, ya que el número telefónico no forma parte de la línea de orden y, por tanto, no se visualiza en respuesta a una orden ps t.
-t	Llama a un terminal con un modem autor respuesta

TABLA 5-5 Opciones de *cu* para llamar a otro sistema

Para llamar a *alpha* utilizando 2.400 *baudios*, marca 9 para obtener una línea exterior y luego 12015551234, utilice la orden siguiente.

```
$ cu -s2400 9=12015551234
```

Para llamar a través de un modem asociado a /dev/term/04, utilice la siguiente línea:

```
$ cu -lterm/04 9=12015551234
```

Después de crear una conexión con el sistema remoto, cu ejecuta dos procesos separados, un proceso de transmisión y un proceso de recepción. ~rproceso de transmisión lee la entrada estándar (normalmente el teclado) y la transmite al sistema remoto, excepto las líneas que comienzan con una ~ (tilde). El proceso de recepción acepta entrada procedente del sistema remoto, excepto las líneas que comienzan con una (tilde), transfiere esa entrada a la salida estándar.

Una vez que se ha abierto una sesión en el sistema remoto, puede hacer cualquier cosa normalmente posible sobre ese sistema. Por ejemplo, supongamos que el usuario se presenta desde *jersey* hasta nevada tecleando la siguiente orden en *jersey*:

```
$ cu nevada
Connected login:
```

tras presentarse con éxito suministrando su nombre de presentación y contraseña, puede ejecutar órdenes en nevada, tales como.

```
$ who
npm          term/17   Feb 2 14:45
ddr          term/04   Feb 2 09:06
khr          term/01   Feb 2 12:07
greq         term/12   Feb 2 12:53
```

5.10.1 Órdenes utilizadas con cu

La razón para utilizar *cu* es que soporta sencillas transferencias de archivos *ASCII*, incluso sobre computadoras que no disponen del Sistema *UNIX*. (Las opciones de paridad y semi-duplex pueden ser necesarias para otros sistemas; las computadoras que ejecutan el Sistema *UNIX* normalmente no las requieren). Siempre que el sistema remoto disponga las órdenes *stty, echo y cat*, es posible que *cu* intercambie archivos.

Se puede utilizar la orden `cu ~%take` para copiar archivos desde el sistema remoto hasta la computadora local. La forma general de esta orden es

```
~ %take gato [aqui]
```

La orden [*take*] copia el archivo gato en el archivo aquí sobre el sistema local. Para copiar archivos desde la computadora local hasta la computadora remota se utiliza la orden `cu ~ %put`. La forma general de esta orden es

```
~ %put gat02 [alli]
```

[*put*] copia el archivo gato2 sobre el sistema remoto con el nombre allí. Observe que tras escribir `%`, cu escribirá el nombre del sistema local entre la ~ y el %.

Por ejemplo, supongamos que usted se ha presentado en nevada mediante la ejecución de una sesión cu desde *jersey*. Puede tomar el archivo memo en nevada y copiarlo en el archivo de nombre newmemo sobre *jersey* utilizando la siguiente orden:

```
~[jersey]%take memo newmemo
```

Análogamente, puede colocar el archivo de nombre data desde *jersey*, llamándole al archivo copiado `data.new`, sobre nevada utilizando la siguiente orden:

```
~[jersey]%put data data.new
```

Instalando estas órdenes cu bajo sistemas operativos tales como `DOS`, `TSO` y `GCOS`, los Sistemas *UNIX* son capaces de comunicarse con una gran variedad de computadoras.

Para ejecutar una orden en el sistema local durante una sesión `cu`, se utiliza la secuencia ~ !orden. Para ejecutar la orden localmente, pero enviar la salida al sistema remoto, se utiliza la secuencia ~ `$orden`. Para enviar linea (tilde linea) `@ máquina` remota se teclea `~~linea` (tilde tilde linea).

Se puede escapar temporalmente de la sesión `cu` a un *shell* interactivo en el sistema local tecleando ! (tilde admiración). Tras ejecutar las órdenes en el sistema local, se termina este shell en la máquina local del modo normal (utilizando éxito). Al terminar este *shell*, se regresa a la sesión en la máquina remota. cu hace el eco de una ! (admiración). Por ejemplo, supongamos que usted ha iniciado una sesión en *arizona* ejecutando una orden cu sobre *jersey*:

```
$ ~[jersey]!
$ pwd
/var/home/khr
$ éxito
```

Para cambiar de directorios en el sistema local durante una sesión *cu* se utiliza la secuencia tilde `~%cd` seguido del nombre de un directorio. (El nombre del sistema local aparecerá tras la tilde una vez que se haya tecleado el `%`.) Este cambio persistirá durante la sesión de conexión remota. Por ejemplo, supongamos que usted ha llamado a nevada desde el sistema *jersey*. Utilice la secuencia tilde, para cambiar al directorio `/home/khr/tools` en *jersey* durante la sesión *cu*.

```
~[jersey]%cd /home/khr/tools
```

Se puede preguntar por que es necesaria una orden `cu` especial para cambiar de directorio en el sistema local. La razón es que la orden `cu ~ !cd` no cambia el directorio en el sistema local; falla porque las órdenes en `cu` son ejecutadas por un *subshell*, de modo que el cambio de directorio no persiste después de terminar el *subshell*. Una vez que usted ha utilizado la orden `cu` para presentarse en una segunda computadora, puede utilizar `cu` de nuevo para presentarse en una tercera computadora. Por ejemplo, mientras tiene abierta sesión en *jersey* puede utilizar `cu` para abrir sesión en *nevada*. Una vez abierta sesión en nevada, puede utilizar cu para presentarse en *arizona*. Puede ejecutar la orden uname en *arizona*, *jersey* y *nevada*, respectivamente, del modo siguiente:

```
uname arizona ~[jersey]!uname jersey ~~[nevada]!uname nevada
```

La sesión en el sistema remoto se termina tecleando `~.` (tilde-punto). Cuando se teclea el . (punto), `cu` pone el nombre del sistema remoto entre corchetes tras la tilde. Por ejemplo, para terminar una conexión con *jersey*:

```
$ ~.
Disconnected
```

`'~%b`	Envía una ruptura (*break*) al sistema remoto.
`~ %d`	Entra y sale del modo de depuración.
`~ t`	Imprime los valores de la estructura term para el terminal de usuario.
`~ l`	Imprime los valores de la estructura term para la línea de comunicación. Cambia entre control de flujo de entrada DC3/DC1 (CTRL-S/CTRL-Q) y ningún control de flujo de entrada. Esto es útil cuando el sistema remoto no responde a los caracteres de control de flujo.
`~%OCF`	Cambia entre control de flujo de salida DC3/DC1 (CRTL-S/CRTL-Q) y ningún control de flujo de salida. Esto permite que el control de flujo sea gestionado por el sistema remoto.
`~%old`	Cambia al estilo antiguo (previo a la Versión 4) de sintaxis `c~`. Esto se utiliza para recibir archivos en el UNIX Sistema V Versión 4 desde un sistema anterior.

TABLA 5-6. Órdenes adicionales

5.10.2 La orden ct

La orden *ct* (llamar a terminal [*call terminal*]) es un modo conveniente de instruir al sistema para que haga una llamada a una terminal asociada a un *modem* de autorrespuesta. La orden

```
$ ct -s1200 9=5551234
```

utiliza la misma sintaxis que `cu` e instruye al sistema para que utilice una línea de 1.200 *baudios* para llamar; marque luego un 9, espere un tono de marcar secundario y marque 5551234. La orden *ct* es utilizada con frecuencia con un trabajo *at* para llamar automáticamente a un terminal. Por ejemplo, el guión

```
at 8:00pm ct -s1200 9=5551234
```

puede ser utilizado para llamar a una terminal propio a las 8:00 P M, ahorrando el tiempo y el gasto de que la terminal llame al sistema.

CAPITULO 6 *Sockets*

Las primitivas de entrada y salida (*I/O*) siguen un modelo referido como *open-read-write-close*. Antes de que un usuario pueda desarrollar operaciones *I/O*, llama a la función `open` (abrir) para especificar el archivo o dispositivo a ser utilizado, y obtener el permiso. La llamada a `open` regresa un entero corto que es el descriptor de archivo (el termino descriptor de archivo surge porque en *UNIX*, todos los dispositivos son considerados como archivos) que el proceso utiliza para desempeñar las operaciones *I/O* en el archivo o dispositivo abierto. Una vez que el objeto ha sido abierto, el proceso de usuario puede hacer una o más llamadas a la función `read` (leer) o a la función `write` (escribir) para transferir datos. `Read` transfiere datos al proceso de usuario; `write` transfiere datos del proceso de usuario al archivo o dispositivo. Ambas funciones necesitan tres argumentos que especifiquen el descriptor de archivos a utilizar, la dirección de un buffer, y el número de bytes a transferir. Después de que todas las operaciones son completadas, el proceso de usuario llama a la función `close` (cerrar) para informar al sistema operativo que se ha terminado de utilizar el archivo (el S.O. cierra automáticamente todos los objetos abiertos si un proceso termina sin hacer una llamada a la función `close`).

6.1 UNIX y TCP/IP

Las primeras instrumentaciones de *TCP/IP* bajo *UNIX* utilizaban el modelo *open-read-write-close* con el archivo especial `/dev/tcp`. Sin embargo, el grupo que añadió los protocolos de red al *UNIX 4BSD* decidió que los protocolos de red eran más complejos que los dispositivos *I/O* convencionales, la interacción entre los procesos de usuario y los protocolos de red son más complejos que la interacción entre los procesos de usuarios y las servicios de *I/O*. En particular, la interfaz del protocolo debe permitir crear servidores que esperen conexiones pasivamente, así como clientes que formen conexiones al activarse. Más aún, los programas de aplicación que envían los datagramas, querrían especificar la dirección de destino con cada datagrama en vez de unir destinos al tiempo que llaman a la función `open`. Para manejar estos casos, los diseñadores decidieron no utilizar el modelo tradicional *open-read-write-close* y añadir varias llamadas al sistema, así como nuevas rutinas de librerías. Añadir los protocolos de red a *UNIX* incrementaron la complejidad de la interfaz *I/O* substancialmente.

La complejidad surge en la interfaz de protocolos de *UNIX*, debido a que los diseñadores intentaron preparar un mecanismo general para acomodar muchos protocolos. Por ejemplo, la generalidad hace posible tener *software* para los protocolos *TCP/IP*, así como para los protocolos Internet de *Xerox (XNS)*, permite que los programas de aplicación puedan utilizar cualquier protocolo o ambos, si así lo desea. Como consecuencia, el programa de aplicación no puede proporcionar una dirección y esperar que el sistema lo interprete correctamente. La aplicación debe especificar explícitamente que la dirección es una dirección *IP*.

6.1.1 Direcciones

En el dominio de *UNIX*, la dirección de un programa es especificada utilizando el nombre y su ruta estándar *UNIX*. En el dominio *INTERNET*, esto no es viable por dos razones. Primero, la ruta estándar no provee algún método para especificar en que computadora se localiza el programa. Segundo, no todas las computadoras conectadas a la red, utilizan el sistema operativo *UNIX*.

Para resolver este problema, se pueden utilizar dos números; el primer número es el número de puerto, un número de 16 *bits*. Cada proceso del nivel de aplicación que utiliza los protocolos *TCP/IP* debe identificarse a si mismo por un número de puerto. El número de puerto es algo similar a la ruta y nombre utilizada en el dominio *UNIX*, para identificar el programa. Este número es utilizado entre los 2 *host* para identificar que programa de aplicación recibirá la información entrante.

El segundo número es una dirección *IP* de 32 *bits*, de la computadora en la cual se tiene acceso al proceso. Cada máquina en la red, dondequiera que se encuentre, tendrá un número único que la distinga de los demás. Debe notarse que la dirección de la maquina, no es lo mismo que el *hostname* de la maquina. Un *hostname* usualmente es una cadena de texto (`intrepid.ecn.purdue.edu`, `sysul3.ifisicacu.unam.mx`, etc.), y no es fácilmente utilizado como una dirección de red, ya que no proporciona por si sola alguna información de como acceder a la máquina en cuestión. Además, ya que un mismo *host* puede estar en más de una red, es posible, que tenga varios números de red o direcciones asociadas; y cada número de red especifica al sistema operativo como alcanzar la maquina, utilizando diferentes rutas de red.

Una notación para las direcciones *IP* de los *host* divide los 32 *bits* en 4 campos de 8 *bits* y especifica el valor para cada campo como un número decimal con cada campo separado con un punto. Por ejemplo, la dirección *INTERNET* de *ISIF* es *010.020.000.052*, o la de *sysul3*`.ifisicacu.unam.mx` es *132.248.7.4*.

6.1.2 Números de puertos

El uso de números de puerto, también provee una capacidad de multiplexaje al permitir que múltiples programas de usuario se comuniquen concurrentemente con un programa de aplicación, así como con *TCP*. Los números de puertos son utilizados para identificar esas entradas de la aplicación. El concepto es muy similar al punto de servicio de aplicación (*Service Application Point* [*SAP*]) en el modelo *OSI*.

En *INTERNET*, algunos de los números de puertos ya están preasignados. Estos son llamados "puertos perfectamente conocidos" y son utilizados para identificar ampliamente las aplicaciones comúnmente utilizadas, llamadas como "servicios perfectamente conocidos". Estos números de puertos ocupan valores en el rango de 0 a 255. Una organización no deberá usar los números de este rango debido a que están reservados. Si desea asignar un número de puerto a cualquier aplicación especifica, deberá utilizar un número mayor a 255.

Debido que el mapeo de los puertos a los procesos de alto nivel pueden ser manejados internamente por cada *host*, *INTERNET* publica números utilizados frecuentemente por los procesos de alto nivel. Las tablas E-1, E-2, E-3, E-4 y E-5 que se encuentran en el apéndice E.1, muestran los números de puerto comúnmente utilizados con sus nombres y descripciones.

Aunque *TCP* establece los números de puerto mayormente utilizados, los números o valores por arriba del 255 son permitidos para uso privado. Los números de puerto previamente asignados tienen todos los *bits*, con excepción de los 8 *bits* de bajo orden puestos en cero. El resto de los valores de números de puerto son permitidos para que cualquier organización los use de acuerdo a su conveniencia.

6.2 SOCKETS

En adición al uso de puertos, los protocolos basados en *TCP/IP* utilizan un identificador abstracto llamado *socket*. Se considera al *socket* como una generalización del mecanismo de acceso de archivos de *UNIX*, de tal forma que identifica y termina el proceso de comunicación. Así como en el acceso de archivos, los programas de aplicación solicitan al sistema operativo crear un *socket*, cuando este es necesitado. El sistema regresa un entero corto que el programa de aplicación utiliza para referenciar el *socket* recién creado. La gran diferencia entre los descriptores de archivos y los descriptores de *sockets*, es que el sistema operativo une un descriptor de archivo a un archivo específico o dispositivo, cuando la aplicación llama a la función `open`; pero puede crear *sockets* sin unirlos a direcciones específicas de destino. La aplicación puede escoger el proporcionar la dirección de destino cada vez que utiliza el *socket* (e.g., cuando envía datagramas), o puede escoger unir la dirección de destino al *socket* y evita que se repita la indicación de la dirección destino (e.g., cuando se realiza una conexión *TCP*).

Los *sockets* se manipulan de igual forma que los archivos o dispositivos de *UNIX*, de tal forma que pueden ser utilizados con operaciones tradicionales, como las funciones `read` y `write`. Por ejemplo, una vez que el programa de aplicación ha creado un *socket* y crea una conexión *TCP* del *socket* a una aplicación externa, el programa puede usar la función `write` para enviar datos a través de la conexión (mientras en el otro extremo se utiliza la función `read` para recibirlos). Para hacer posible el uso de primitivas como `read` y `write` con archivos y *sockets*, el sistema operativo asigna descriptores de *sockets* y de archivos desde el mismo conjunto de enteros y se asegura que si un número ya ha sido reservado como un descriptor de archivo, este no sea nuevamente asignado como un descriptor de *socket*. Debido a que los descriptores de archivos y los de *sockets* se mezclan sin distinción, generalmente se utiliza el descriptor de archivo para ambos casos.

La aplicación que se encuentra sobre el nivel de *TCP* en un *host* es identificada por un número de puerto. El número de puerto es concatenado con la dirección internet *IP* del *host* para formar un *socket*. Esta dirección debe ser única a lo largo de *INTERNET*, y un par de *sockets* identifican únicamente cada extremo de la conexión. Por ejemplo:

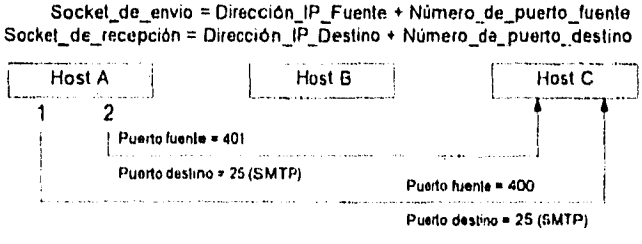


Figura 6-1. Estableciendo sesiones con un puerto destino

La figura 6-1, muestra como son asignados y manipulados entre dos *host*, los números de puertos. En el evento 1, el *host* A envía un segmento *TCP* al *host* C. Este segmento es una solicitud de una conexión *TCP* para comunicarse con un proceso de alto nivel. En este ejemplo, el puerto perfectamente conocido es igual a 25, que es el número asignado para *SMTP* (*Simple Mail Transfer Protocol*). El puerto de destino es preparado como 25. El identificador del puerto fuente es local

Un *host* escoge cualquier número conveniente a sus operaciones internas. En este ejemplo, el puerto fuente 400 es seleccionado para la primer conexión. La segunda conexión, denotada en el evento 2, también es destinada al *host* C para utilizar *SMTP*. Consecuentemente, el puerto de destino es igualmente 25. El identificador de puerto fuente es cambiado; en este momento es puesto al valor 401. El uso de dos números diferentes para el acceso *FTP* previene cualquier mezcla o interferencia entre las dos sesiones del *host* A y el C.

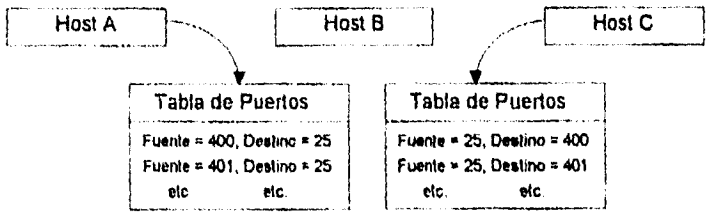


FIGURA 6-2. Relación con Tablas de Puertos

La figura 6-2, muestra el efecto de los dos segmentos que fueron utilizados para establecer la conexión en la figura anterior. Los *hosts* A y C generalmente almacenan la información de las conexiones en tablas. En este ejemplo las tablas son llamadas tablas de puertos. Obsérvese la relación inversa que existe entre las tablas; en el *host* A, los puertos origen son 400 y 401, y los puertos de destino son 25 para ambos casos; en cambio, en el *host* C, los puertos de origen son 25 y los puertos de destino son 400 y 401. En otras palabras, los números de puertos de origen y destino son invertidos por los módulos *TCP* para enviar y recibir entre los *hosts*.

Es posible que otro *host* pueda enviar una solicitud de conexión al *host* C, en el cual los números de puerto fuente y puerto destino sean igual a los valores que ya se están utilizando. Ciertamente no sería inusual que el puerto destino sea el mismo valor, ya que este es un puerto "perfectamente conocido", y por tanto es solicitado frecuentemente. Y ya que los identificadores de puertos fuente son seleccionados en forma local, puede coincidir el número de puerto, con algún otro puerto de algún otro *host*; la siguiente figura muestra que el *host* B ha seleccionado también el puerto fuente 400.

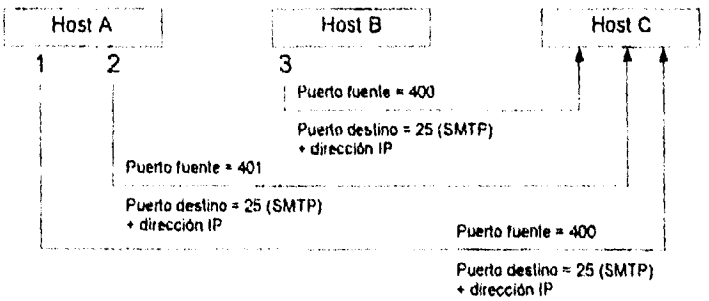


FIGURA 6-3. Distinción entre identificadores de puertos.

Sin algún tipo de identificador adicional, la primer conexión entre los *hosts* A y C, y la conexión reciente de B y C se encuentran en conflicto, ya que están utilizando los mismos números de puertos fuente y destino. En estos casos el *host* C puede diferenciar fácilmente las conexiones al utilizar las direcciones *IP* que se encuentran en el encabezado *IP* de los datagramas.

De esta forma, los números de puertos fuente pueden estar duplicados y la dirección *INTERNET* es utilizada para distinguir cada conexión. En adición a las direcciones *IP* y los números de puerto, muchos sistemas también identifican un *socket* con un valor de familia de protocolos; por ejemplo, *IP* es una familia de protocolos; *DecNET* es otra familia de protocolos. La manera en que cada familia de protocolos es identificada, depende del vendedor, y del sistema operativo.

6.2.1 Usando sockets para soportar multiplexaje

Debido a que los números de puertos pueden ser utilizados por más de un punto final de conexión, los usuarios pueden compartir un recurso de puerto simultáneamente. Esto es, múltiples usuarios pueden ser multiplexados simultáneamente a través de un puerto. En la figura anterior, tres usuarios están compartiendo el puerto 25 (*UDP* también soporta multiplexaje de puertos).

6.2.2 Las diferentes clases de Sockets

Las clases de *sockets* pueden ser clasificadas de acuerdo a tres atributos:

Dominios (domain)

El dominio de un *socket* indica donde pueden residir los *sockets* del cliente y servidor; los valores de dominio permitidos se encuentran en el archivo `<sys/socket.h>`. El dominio de *UNIX* es *AF_UNIX*; el dominio de *INTERNET* es *AF_INET*; los dominios que actualmente se soportan son:

- *AF_UNIX* (los clientes y el servidor deben estar en la misma maquina)
- *AF_INET* (los clientes y el servidor pueden estar en cualquier lugar de *INTERNET*)
- *AF_NS* (los clientes y el servidor pueden estar dentro de un sistema de red *XEROX*)

AF significa Familia de Direcciones (*Address Family*). Este es un conjunto similar de constantes que empiezan con *PF*, el cual proviene de Familia de Protocolos (*Protocol Family*) (i.e., *PF_UNIX*, *PF_INET*, etc.). Cualquier conjunto puede ser utilizado, ya que son equivalentes.

Tipos (type)

El tipo de un *socket* determina el tipo de comunicación que puede existir entre el cliente y el servidor; los tipos de *sockets* también son encontrados en el archivo `<sys/socket.h>`. Los dos tipos principales que se soportan actualmente son:

- *SOCK_STREAM*: secuenciable, confiable, conexión basada en dos direcciones, cadenas de *bytes* de longitud variable.
- *SOCK_DGRAM*: en una dirección a un tiempo, poco confiable, mensajes de longitud fija.

Otros tipos que son incluidos ya sea en las etapas de planificación o son instrumentados en algunos dominios son:

- *SOCK_SEQPACKET*: secuenciable, confiable, conexión basada en dos direcciones, paquetes de *bytes* de longitud de fisica.
- *SOCK_RAW*: provee acceso a protocolos e interfaces internos de la red.

Protocolos (protocol)
El valor del protocolo especifica los medios de bajo nivel por el cual el tipo del *socket* es instrumentado; ya que las familias de protocolos usualmente consisten de más de un protocolo, las llamadas al sistema que esperan un parámetro de protocolo aceptan 0 como "escoge el protocolo más adecuado". Muchos sistemas solo soportan protocolos diferentes a 0 como una opción extra.

El concepto del *socket*, es muy parecido al concepto de entradas/salidas (*I/O*) del *UNIX BSD*. Un *socket* no es más que un punto final en los procesos de comunicaciones. Sin embargo, a diferencia de los conceptos de *sockets* con archivos de entrada y salida, la especificación *TCP/IP* de *UNIX BSD* permite que un *socket* sea creado sin especificar una dirección destino. Una dirección destino en una llamada posterior al sistema seria utilizada para crear la unión final entre las direcciones de envío y recepción.

6.2.3 Servicios orientados a conexión
El proceso de comunicación vía *sockets* esta basado en el modelo cliente servidor. Un proceso, conocido como "proceso servidor", crea un *socket*, cuyo nombre es conocido por otros "procesos clientes". Estos procesos clientes pueden "hablarle" al proceso servidor a través de una conexión al *socket*; para hacer esto, un proceso cliente crea primero un *socket* sin nombre y entonces solicita que sea conectado al *socket* creado por el servidor. Una conexión exitosa regresara un descriptor de archivo al cliente, y uno al servidor, los cuales pueden ser utilizados para lectura y escritura; no como los *pipes*, los *sockets* son bidireccionales.

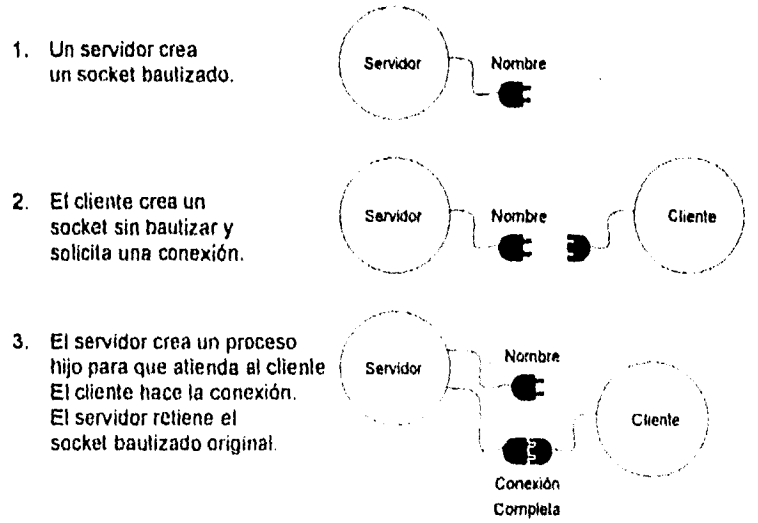


FIGURA 6-4. La conexión de un *socket*.

Una vez que la conexión del *socket* esta hecha, es bastante común para el proceso servidor, crear un proceso hijo dependiente que converse con el cliente, mientras que el proceso original continua aceptando otras conexiones de clientes.

Un ejemplo típico de esto, es una impresión remota. el proceso servidor acepta un cliente que desea enviar un archivo para impresión, y entonces crea un proceso hijo que desempeña la transferencia del archivo. El proceso original, mientras tanto, espera por más solicitudes de impresión de clientes.

6.3 ESCRIBIENDO PROGRAMAS DE SOCKETS

Cualquier programa en C que utilice *sockets* debe incluir los archivos `/usr/include/sys/types.h` y `/usr/include/sys/socket.h`. Archivos adicionales deben ser incluidos basados en el dominio del *socket* que se desea utilizar:

Dominio	Archivos adicionales
AF_UNIX	`/usr/include/sys/un.h`

AF_INET	`/usr/include/netinet/in.h`
	`/usr/include/arpa/inet.h`
	`/usr/include/netdb.h`

TABLA 6-1. Archivos adicionales para *sockets*

6.3.1 El Servidor

El servidor es el proceso que es responsable para crear un *socket* bautizado y aceptar conexiones a el. Para completar esto, debe utilizar las siguientes llamadas al sistema en el siguiente orden:

Nombre	Significado
`socket`	crea un *socket* sin nombre
`bind`	proporciona un nombre al *socket*
`listen`	especifica el máximo número de conexiones pendientes
`accept`	acepta de un cliente, una conexión al *socket*

TABLA 6-2. Llamadas al sistema (servidor).

Un proceso puede crear un *socket* utilizando la función `socket()`, la cual trabaja así:

```
int socket(int domain, int type, int protocol)
```

La función `socket(domain, type, protocol)` es la llamada al sistema que permite la creación de un *socket* sin nombre, y contiene tres argumentos. El campo `domain` describe la familia de protocolos (dominio) asociada con el *socket*; puede incluir a las familias *INTERNET*, *PUP*, *DEC*, *Appletalk*, etc. El campo `type` estipula el tipo de comunicaciones deseados en esta conexión. El programador puede establecer valores para especificar un servicio de datagramas, servicio de entrega confiable, o un simple *socket*. El tercer argumento permite al programador codificar el tipo de servicio que provee cada uno de los protocolos dentro de la familia de protocolos. Si se proporciona un cero, el sistema seleccionara el protocolo adecuado de acuerdo con el dominio. Si tiene éxito, regresa un descriptor asociado con el *socket* creado recientemente; si no regresa -1.

Se menciono anteriormente que *UNIX* permite que un *socket* sea creado sin preparar la dirección para el *socket* (llamado nombre en *UNIX V*). Las comunicaciones no pueden ocurrir hasta que las direcciones local y externa *INTERNET* son declaradas para la asociación entre las entidades que se comunican. En el dominio *UNIX*, la ruta local y externa de los nombres son utilizados.

Una vez que se ha creado un *socket* sin nombre, debe asociársele un nombre utilizando la función `bind()`, la cual trabaja así:

```
int bind(int fd, struct sockaddr* address, int addressLen)
```

Asocia el *socket* sin nombre con el descriptor `fd` con la dirección del *socket* almacenada en `address`. `addressLen` debe contener la longitud de la estructura. El tipo y el valor del la dirección entrante depende del dominio del *socket*. Si el *socket* se encuentra en el dominio de *AF_UNIX*, un apuntador a la estructura **`sockaddr_un`** debe ser referenciada a (**`sockaddr*`**) y pasado a la función como `address`. Esta estructura tiene dos campos que deben ser inicializados como:

Campo	Valor
`sun_family`	*AF_UNIX*
`sun_path`	La ruta completa del *socket* (absoluta o relativa), hasta una longitud de 108 caracteres

TABLA 6-3. Inicialización de la estructura `sockaddr_un`.

Si el *socket* bautizado *AF_UNIX* ya existe, ocurrirá un error, de tal forma, que es buena idea desligar el nombre antes de intentar asociarlo [`unlink()`]. Si el *socket* se encuentra en el dominio *AF_INET*, un apuntador a la estructura **`sockaddr_in`** debe ser referenciada a (**`sockaddr*`**) y pasado a la función como `address`. Esta estructura tiene cuatro campos que deben ser inicializados como:

Campo	Valor
`sin_family`	*AF_INET*
`sin_port`	El número de puerto del *socket INTERNET*
`sin_addr`	Una estructura del tipo `in_addr` que contiene la dirección *INTERNET*
`sin_zero`	Dejar vacío

TABLA 6-4. Inicialización de la estructura `sockaddr_in`.

Cuando un proceso servidor atiende a una conexión de un cliente, siempre es posible que otro cliente también solicite una conexión. La función `listen()` permite especificar el número de conexiones pendientes que pueden ser "*encoladas*", y trabaja así:

```
int listen(int fd, int queueLength)
```

Permite especificar el número máximo de conexiones pendientes en un *socket*. Si un cliente intenta una conexión a un *socket* en donde su cola esta llena, se le niega el acceso con una indicación de *ECONNREFUSED*. Otros códigos de error asociados con la función `listen()` son `EBADF` = `fd` es invalido; `ENOTSOCK` = `fd` no es un *socket*; `EONOTSUPP` = el *socket* no soporta una operación de escucha (*listen*).

Ya que el *socket* se ha creado, bautizado y especificado su cola de espera, el paso final consiste en aceptar las solicitudes de conexión de clientes. Para hacer esto, el servidor debe usar la función `accept()`:

```
int accept(int fd, struct sockaddr* address, int addressLen)
```

Escucha al *socket* bautizado por el servidor, referenciado por `fd` y espera hasta que una solicitud de conexión es recibida. Cuando esto ocurre, `accept()` crea un *socket* sin nombre con los mismos atributos del *socket* del servidor original, lo conecta al *socket* del cliente, y regresa un nuevo descriptor de archivo que puede ser utilizado para comunicarse con el cliente. El *socket* del servidor original, puede ser utilizado para aceptar más conexiones.

La estructura `address` es llenada con la dirección del cliente, y normalmente, solo es utilizada en conjunción con las conexiones *INTERNET*. El campo `addressLen` inicialmente deberá apuntar a un entero que contiene el tamaño de la estructura apuntada por *address*. Cuando una conexión es realizada, el entero que lo apunta, es puesto al tamaño actual en *bytes* de la dirección resultante.

Si `accept()` funciona, regresará un nuevo descriptor de archivo que puede ser utilizado para hablar con el cliente; si no, regresará un -1.

6.3.2 Dando servicio a un cliente

Cuando ocurre una conexión de un cliente, la secuencia de eventos en el programa es esta.

- El servidor genera un proceso hijo
- El proceso original cierra el archivo descriptor del cliente que se ha creado recientemente y ejecuta la llamada `accept()`, listo para aceptar nuevas solicitudes de cliente.
- El proceso hijo habla con el cliente, utilizando las funciones `read()` y `write()`. Cuando la conversación es completada, el proceso hijo cierra el archivo descriptor del cliente y termina.

Ejemplo de creación de un *socket* en un servidor:

```
serverFD = socket(AF_UNIX, SOCK_STREAM, DEFAULT_PROTOCOL);
serverUNIXAddress.sun_family = AF_UNIX;          ...Proporciona el tipo de dominio
strcpy(serverUNIXAddress.sun_path, "nombre_s");                ...da el nombre
unlink("nombre_s");                              ...remueve el archivo si ya existe
bind(serverFd, serverSockAddrPtr, serverLen);                  ...asocia el archivo

listen(serverFd, 5);                                ...establece máximo número de
                                                                ...conexiones pendientes
while (1)                                                      ...realiza ciclos "infinitos"
{                                                  ...acepta una conexión de un cliente
  clientFd = accept (serverFd, clientSockAddrPtr, &clientLen);
  if (fork () == 0)                                            ...crea un proceso hijo
  {
    write(clientFd, datos, sizeof(datos));                          ...envía los datos
    close(clientFd);                                               ...cierra el socket
    exit (0);                                     ...termina ejecución del proceso hijo
  }
  else
    close (clientFd);                                ...cierra el descriptor del cliente
}
```

6.3.3 El Cliente

Un cliente es un proceso que es responsable de crear un *socket* sin nombre y asociarlo al *socket* bautizado del servidor. Para cumplir con esto, tiene que hacer uso de las siguientes llamadas de sistema:

Nombre	Significado
`socket`	Crea un *socket* sin nombre
`connect`	Asocia un *socket* sin nombre de cliente a un *socket* bautizado del servidor

TABLA 6-5. Llamadas al sistema (cliente).

La forma que un cliente utiliza `socket()` para crear un *socket* sin nombre es la misma que utiliza el servidor. El dominio, tipo, y protocolo del cliente, deben coincidir con los del servidor. Para conectarse al *socket* del servidor, un proceso cliente debe llenar una estructura con la dirección del *socket* del servidor y después utilizar `connect()`:

```
int connect(int fd, struct sockaddr* address, int addressLen)
```

Intenta conectarse a un *socket* de servidor, cuya dirección se encuentra en una estructura apuntada por `address`. Si lo logra, `fd` puede ser utilizado para comunicarse con el *socket* del servidor. El tipo de estructura que apunta `address` debe seguir las mismas reglas descritas en `bind()`:

- Si el *socket* se encuentra en el dominio de *AF_UNIX*, un apuntador a la estructura **`sockaddr_un`** debe ser referenciada a (**`sockaddr*`**) y pasado a la función como `address`.
- Si el *socket* se encuentra en el dominio *AF_INET*, un apuntador a la estructura **`sockaddr_in`** debe ser referenciada a (**`sockaddr*`**) y pasado a la función como `address`.

`addressLen` debe ser igual al tamaño de la estructura `address`. Si la conexión se lleva a cabo, `connect()` regresa un 0. Si el *socket* del servidor no existe, o su cola de conexiones pendientes esta llena, `connect()` regresa un -1.

Ejemplo de creación y conexión de un *socket* en un cliente:

```
clientFd = socket(AF_UNIX, SOCK_STREAM, DEFAULT_PROTOCOL);
serverUNIXAddress.sun_family = AF_UNIX;          ...Proporciona el tipo de dominio
strcpy(serverUNIXAddress.sun_path, "nombre_s");          ...da el nombre

do                                                ...repite hasta que se realice una
{                                                                    conexión
  result = connect (clientFd, serverSockAddrPtr, serverLen)
  if (result == -1) sleep (1);                    ...espera y vuelve a intentarlo
}
while (result == -1);
```

6.3.4 Comunicación a través de sockets

Una vez que los *sockets* del cliente y del servidor se han conectado, sus descriptores de archivos pueden ser utilizados por `write()` y `read()`. El servidor utiliza `write()` para enviar sus datos al cliente.

```
linel = "datos....";
line2 = "mas datos...";
write (fd, linel, strlen (linel) + 1);          .. envia la primer línea
write (fd, line2, strlen (line2) + 1);          ...envia la segunda línea
```

Nombre	Significado
`socket`	Crea un *socket* sin nombre
`connect`	Asocia un *socket* sin nombre de cliente a un *socket* bautizado del servidor

TABLA 6-5. Llamadas al sistema (cliente).

La forma que un cliente utiliza `socket()` para crear un *socket* sin nombre es la misma que utiliza el servidor. El dominio, tipo, y protocolo del cliente, deben coincidir con los del servidor. Para conectarse al *socket* del servidor, un proceso cliente debe llenar una estructura con la dirección del *socket* del servidor y después utilizar `connect()`:

```
int connect(int fd, struct sockaddr* address, int addressLen)
```

Intenta conectarse a un *socket* de servidor, cuya dirección se encuentra en una estructura apuntada por `address`. Si lo logra, `fd` puede ser utilizado para comunicarse con el *socket* del servidor. El tipo de estructura que apunta `address` debe seguir las mismas reglas descritas en `bind()`:

- Si el *socket* se encuentra en el dominio de *AF_UNIX*, un apuntador a la estructura **`sockaddr_un`** debe ser referenciada a (**`sockaddr*`**) y pasado a la función como `address`.
- Si el *socket* se encuentra en el dominio *AF_INET*, un apuntador a la estructura **`sockaddr_in`** debe ser referenciada a (**`sockaddr*`**) y pasado a la función como `address`.

`addressLen` debe ser igual al tamaño de la estructura `address`. Si la conexión se lleva a cabo, `connect()` regresa un 0. Si el *socket* del servidor no existe, o su cola de conexiones pendientes esta llena, `connect()` regresa un -1.

Ejemplo de creación y conexión de un *socket* en un cliente:

```
clientFd = socket(AF_UNIX, SOCK_STREAM, DEFAULT_PROTOCOL);
serverUNIXAddress.sun_family = AF_UNIX;          ...Proporciona el tipo de dominio
strcpy(serverUNIXAddress.sun_path, "nombre_s");          ...da el nombre

do                                                  ...repite hasta que se realice una
{                                                                    conexión
  result = connect (clientFd, serverSockAddrPtr, serverLen)
  if (result == -1) sleep (1);                      ...espera y vuelve a intentarlo
}
while (result == -1);
```

6.3.4 Comunicación a través de sockets

Una vez que los *sockets* del cliente y del servidor se han conectado, sus descriptores de archivos pueden ser utilizados por `write()` y `read()`. El servidor utiliza `write()` para enviar sus datos al cliente.

```
line1 = "datos....";
line2 = "mas datos...";
write (fd, line1, strlen (line1) + 1);          ... envia la primer línea
write (fd, line2, strlen (line2) + 1);          ...envia la segunda línea
```

y el cliente utiliza `read()` para recibir los datos del servidor:

```
do                          ...lee caracteres hasta que encuentres un caracter nulo o fin de entrada
{
  n = read (fd, str, 1);
}
while (n > 0 && *str@ != NULL);
```

El servidor y el cliente deben ser cuidadosos para cerrar sus respectivos descriptores de archivo, una vez que estos ya no de utilizaran.

6.3.5 Sockets Internet

Un *socket INTERNET* es especificado por dos valores: una dirección de 32 *bits*, la cual especifica a un solo *host*; y un número de puerto de 16 *bits*, el cual especifica un puerto en particular dentro del *host*. Esto significa que un cliente *INTERNET* debe conocer no solo la dirección *IP* del servidor, sino que también el número de puerto del servidor.

Existen varios números de puerto estándares que están reservados para uso del sistema. Por ejemplo, el puerto 13 siempre es atendido por el proceso que manda la fecha y hora del *host* a cualquier cliente que lo solicite.

6.3.5.1 Los Clientes Internet

El procedimiento para crear un cliente *INTERNET*, es el mismo que en un cliente de *AF_UNIX*, excepto por la inicialización de la dirección del *socket*. Anteriormente, en la discusión de `bind()`, se menciono que la estructura de la dirección *INTERNET* del *socket* es del tipo `sockaddr_in`, y tiene cuatro campos:

- **`sin_family`**, el dominio del *socket*, el cual debe ser inicializado como *AF_INET*.
- **`sin_port`**, el número de puerto, que este caso es el número 13.
- **`sin_addr`**, el número de 32 *bits* del cliente/servidor.
- **`sin_zero`**, que es para relleno y no es inicializado.

Cuando se crea el *socket* del cliente, el único truco es determinar la dirección *IP* de 32 *bits*. Para esto, si se introduce una cadena que contiene el formato de dirección *INTERNET* de cuatro dígitos, `inet_addr()` es llamada para desarrollar la conversión. He aquí como trabaja:

```
unsigned long inet_addr(char* string)
```

Regresa la dirección *IP* de 32 *bits* que corresponde al formato A.B.C.D. La dirección *IP* esta en orden *network-byte*. El orden *network-byte* es un ordenamiento de *host* neutral de los *bytes* en la dirección *IP*. Esto es necesario, ya que el ordenamiento de *bytes* puede diferir de máquina a maquina, el cual podría hacer que las direcciones *IP* no fueran portables. La función contraria es `inet_ntoa()`:

```
char* inet_ntoa(struct in_addr address)
```

La función toma una estructura del tipo `in_addr` como argumento y regresa un apuntador a una cadena que describe la dirección *IP* en el formato A.B.C.D. Si se quiere indicar que la dirección es del *host* local, basta con obtener su nombre, y este se obtiene con la función `gethostname()`:

```
int gethostname(char* name, int nameLen)
```

`gethostname()` proporciona al apuntador de caracteres `name` con longitud `nameLen` la cadena que contiene el nombre del *host* local. Si es el *host* local, o se indica el nombre de algún *host* remoto, ambos se procesan de la misma forma, es decir, se observa si se encuentra dado de alta en el archivo de hosts de la red `/etc/hosts`. Esto lo ejecuta la función `gethostbyname()`:

```
struct hostent* gethostbyname(char* string)
```

Busca el archivo `/etc/hosts`, y regresa un apuntador a la estructura **`hostent`** que describe el archivo asociado con la cadena `name`. Si el nombre no es encontrado en el archivo, un valor `NULL` es regresado. La estructura `hostent` tiene varios campos, pero el más interesante es un campo del tipo (**`struct in_addr*`**) llamado **`h_addr`**. Este campo contiene el número *IP* asociado al *host* en un subcampo llamado **`s_addr`**.

Campo	Valor
`h_name`	Nombre del *host*
`h_aliases`	Lista de alias
`h_addrtype`	Tipo de dirección de *host*
`h_length`	Longitud de la dirección
`h_addr_list`	Lista de direcciones

TABLA 6-6. Valor y descripción de algunos campos.

A continuación se muestran tres formas de obtener la dirección *IP*.

```
dirIP = "132.248.7.4";                                  ...obtén dirección IP del
inetAddress = inet_addr(dirIP);                          string "132.248.7.4"

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

gethostname(hostName,100);                              ...obtén nombre host local
hostStruct = gethostbyname(hostName);                   ...busca en tabla local
if (hostStruct != NULL) {                                  ...si encontró
  hostNode = (struct in_addr*) hostStruct->h_addr;         ...obtén dirección IP
  inetAddress = hostNode->s_addr;
}

/*-------------------------------------------------------*/
hostName = "sysul3"                                     ...especifica nombre host
hostStruct = gethostbyname(hostName);                   ...busca en tabla local
if (hostStruct != NULL) {                                  ...si encontró
  hostNode = (struct in_addr*) hostStruct->h_addr;         ...obtén dirección IP
  inetAddress = hostNode->s_addr;
}
```

Una vez que la dirección *IP* ha sido determinada (almacenada en **`inetAddress`**), los campos de dirección del *socket* cliente son llenados:

```
bzero ((char*)&serverINETAddress,sizeof(serverINETAddress));
serverINETAddress.sin_family = AF_INET;                 ...utiliza Internet
serverINETAddress.sin_addr.s_addr = inetAddress;        ...indica dirección IP
serverINETAddress.sin_port = htons(13);                 ...indica puerto servidor
```

bzero() borra el contenido de la estructura de dirección del *socket* antes de que sus campos sean asignados.

```
void bzero(char* buffer, int length)
```

La función llena el arreglo buffer de tamaño length con ceros (caracter NULO). Al igual que la dirección *IP*, el número de puerto también es convertida al orden *network-byte* por la función hton():

```
unsigned long  htonl(unsigned long hostLong)
unsigned short htons(unsigned short hostShort)
unsigned long  ntohl(unsigned long networkLong)
unsigned short ntohs(unsigned short networkShort)
```

Cada una de estas funciones desarrolla una conversión entre números con formato de *host*, y números con formato de red. Por ejemplo, htonl() regresa el formato de red equivalente del formato de *host* almacenado en hostLong, y ntohs() regresa el formato de *host* equivalente del formato de red almacenado en networkShort. El paso final es crear el *socket* del cliente e intentar la conexión. El código para realizar esto, es el mismo que para los *sockets AF_UNIX*:

```
clientFd = socket (AF_INET, SOCK_STREAM, DEFAULT_PROTOCOL);
do                                    ...repite hasta que se realice una conexión con el servidor
{
  result = connect (clientFd,serverSockAddrPtr,serverLen);
  if (result == -1) sleep (1);                    ...espera 1 seg. y vuelve a intentar
}
while (result == -1);
```

6.3.5.2 Servidores Internet

Crear un servidor *INTERNET* actualmente es muy fácil. Los campos **sin_family**, **sin_port**, y **sin_zero** de la estructura de dirección del *socket* deben ser llenados como en el ejemplo del cliente. La única diferencia es que el campo s_addr debe contener el valor de la constante INADDR_ANY convertida al orden *network-byte*, lo cual significa "acepta cualquier solicitud de conexión del cliente." El siguiente código muestra como crear una dirección de *socket* de servidor:

```
int serverFd;                                          ...socket del servidor
struct sockaddr_in serverINETAddress;          ...dirección INTERNET del servidor
struct sockaddr* serverSockAddrPtr;          ...apuntador a la dirección del servidor
struct sockaddr_in clientINETAddress;          ...dirección INTERNET del cliente
struct sockaddr* clientSockAddrPtr;          ...apuntador a la dirección del cliente
int port = 13;                          ...dar el número de puerto que se desea servir
int serverLen;                            ...longitud de la estructura de la dirección

serverFd = socket(AF_INET, SOCK_STREAM, DEFAULT_PROTOCOL);      ...crear socket
serverLen = sizeof(serverINETAddress);                  ...longitud de la estructura

bzero((char*) &serverINETAddress, serverLen);                  ...borrar estructura
serverINETAddress.sin_family = AF_INET;                ...en dominio de INTERNET
serverINETAddress.sin_addr.s_addr = htonl(INADDR_ANY);         ...acepta a clientes
serverINETAddress.sin_port = htons (port);          ...número de puerto del servidor
```

Cuando la dirección es creada, el *socket* es unido a la dirección y el tamaño de su cola de espera es especificada de la manera usual

```
serverSockAddrPtr = (struct sockaddr*) &serverINETAddress)
bind (serverFd, serverSockAddrPtr, serverLen);
listen (serverFd, 5);
```

El paso final es aceptar conexiones de clientes. Cuando una conexión se realiza, la dirección de *socket* del cliente es llenada con la dirección *IP* del cliente y un nuevo descriptor de archivo es regresado

```
clientLen = sizeof (clientINETAddress);
clientSockAddrPtr = (struct sockaddr*) clientINETAddress;
clientFd = accePt (serverFd, clientSockAddretr, &clientLen);
```

6.3.5.3 Otras llamadas de sistema

Existen, además otras funciones de sistema con las que se obtienen más servicios para el usuario, entre ellas, para enviar y recibir datos.

```
send(socket, buffer, sizeofbuf, flags)        envia datos
recv(socket, buffer, sizeofbuf, flags)        recibe datos
```

La única diferencia entre estas llamadas y las funciones `write` y `read` es el campo extra `flags`. Este campo permite especificar algunas opciones al *enviar/recibir* datos de un *socket* conectado. Estas banderas son: `MSG_PEEK`, examina el siguiente mensaje sin leerlo; `MSG_OOB`, recibe datos urgentes (*out-of-Band*; `MSG_DONTROUTE`, envia datos, pero no realices el itinerario actual (para diagnósticos y mantenimiento).

```
sendto(socket, buffer, sizeofbuf, flags, destaddr, addrlen)
recvfrom(socket, buffer, sizeofbuf, flags, souraddr, addrlen)
```

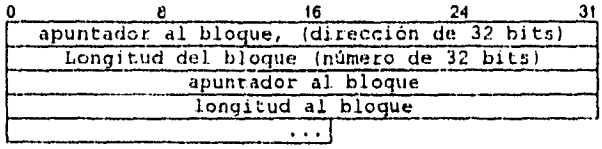
En las funciones anteriores, se añadieron dos parámetros más, el parámetro `destaddr` en **`sendto`** contiene la dirección a la que se dirigen los datos, el parámetro `souraddr` en **`recvfrom`** es la dirección de donde se obtienen los datos. El parámetro `addrlen` de ambas funciones indica la longitud de la dirección.

```
sendmsg(socket, message, flags)
recvmsg(socket, message, flags)
```

Las funciones anteriores permiten recibir o enviar a través de un *socket* desconectado; los parámetros son muy similares a los de las funciones `read` y `write`, con excepción del parámetro message, el cual define una estructura que incluye la dirección y el tamaño del mensaje y algunas entidades de autenticidad. La estructura tiene la siguiente forma:

0	8	16	24	31
apuntador a la dirección del socket				
longitud de la dirección				
apuntador a la lista IOVEC				
longitud de la lista IOVEC				
apuntador a la lista de derechos de acceso				
longitud de la lista de derechos de acceso				

La estructura *IOVEC* es de la forma:



La estructura *IOVEC* también se utiliza con las siguientes funciones:

```
writev(socket, iovector, vectorlen)
readv(socket, iovector, vectorlen)
```

Estas funciones funcionan en forma similar a las funciones `write` y `read`, excepto que utiliza una forma de escritura agrupada, haciendo posible para la aplicación escribir datos sin copiarlos en memoria contigua. En el apéndice E.1, tabla E-6 se presenta una lista con varias funciones y su descripción.

6.4 PROGRAMAS CLIENTE Y SERVIDOR EN EL DOMINIO UNIX

CHEF, es el programa servidor, el cual crea un *socket* llamado "receta" y escribe la receta a cualquier cliente que la solicita. La receta es una colección de cadenas de longitud variable terminadas con el caracter NULO. *AYUDANTE*, es el programa cliente, el cual se conecta al *socket* llamado "receta" y lee la receta del servidor. Este despliega la receta a la salida estándar como lo lee, y entonces termina. El proceso servidor corre en *background*. Cualquier proceso cliente que se conecte al servidor, provoca que el servidor genere un proceso hijo para manejar la transferencia de la receta, permitiendo al servidor original aceptar las otras conexiones entrantes. He aquí una muestra de la salida del ejemplo del *CHEF/AYUDANTE*:

```
armando> chef &                                   ...corre al servidor en background
[1] 5684
armando> ayudante                                 ...corre el cliente y despliega la receta
Nueces, peras, crema,
bate, muele, acitrona y cuece.
armando> ayudante                                 ...corre nuevamente el cliente
Nueces, peras, crema,
bate, muele, acitrona y cuece.
armando> kill %1                                  ...termina al proceso servidor
[1]    Terminated
armando> _
```

6.4.1 El programa servidor CHEF

El Listado completo del programa *Chef* se encuentra en el apéndice E.2. El proceso crea un *socket* sin nombre en la linea 30:

```
serverFD = socket(AF_UNIX, SOCK_STREAM, DEFAULT_PROTOCOL);
```

El servidor asigna los campos **sockaddr_un** y ejecuta la función `bind()` en las líneas 31..34:

```
serverUNIXAddress.sun_family = AF_UNIX;          ...Proporciona el tipo de dominio
strcpy(serverUNIXAddress.sun_path, "receta");    ...da el nombre
unlink("receta");                                ...remueve el archivo si ya existe
bind(serverFd, serverSockAddrPtr, serverLen);    ... asocia el archivo
```

El servidor especifica el máximo número de conexiones pendientes en la línea 35:

```
listen(serverFd, 5);          ...establece máximo número de conexiones pendientes
```

El servidor acepta la conexión en la línea 40:

```
clientFd = accept(serverFd,clientSockAddrPtr, &clientLen);
```

Servicio a un cliente. Cuando ocurre una conexión de un cliente, la secuencia de eventos en el programa es esta:

```
while (1)                                          ...realiza un ciclos "infinitos"
{
                                                   ...acepta una conexión de un cliente
  clientFd = accept (serverFd, clientSockAddrPtr, &clientLen);

  if (fork () == 0)                                ...crea un proceso hijo
  {
    writeRecipe (clientFd);                        ...envía la receta
    close (clientFd);                              ...cierra el socket
    exit (0);                                      ...termina la ejecución del proceso hijo
  }
  else
    close (clientFd);                              ...cierra el descriptor del cliente
}
```

Observe que el servidor escoge ignorar las señales `SIGCHLD` en la línea 21, de tal forma que el proceso hijo puede morir inmediatamente sin requerir que el padre acepte sus códigos de regreso.

6.4.2 El programa cliente AYUDANTE

El listado completo del programa Ayudante se encuentra en el apéndice E.3. El proceso cliente crea un *socket* sin nombre en la línea 22

```
21   /* crea un socket UNIX, bidireccional, protocolo más apropiado */
22   clientFd = socket (AF_UNIX, SOCK_STREAM, DEFAULT_PROTOCOL);
23   serverUNIXAddress.sun_family = AF_UNIX;          /* dominio del servidor */
```

El proceso cliente llama a `connect()` hasta que se realiza con éxito una conexión, esto es en las líneas 26..31:

```
do                                                 ...repite hasta que se realice una conexión
{
  result = connect (clientFd, serverSockAddrPtr, serverLen)
  if (result == -1) sleep (1);                     ...espera y vuelve a intentarlo
}
while (result == -1);
```

6.4.3 Comunicación a través de los sockets

El servidor utiliza write en las líneas 55..64 del listado del servidor (*CHEF*):

```
writeRecipe (fd)
  int fd;
{
  static char* line1 = " Nueces, peras, crema,";
  static char* line2 = " bate, muele, acitrona y cuece.";
  write (fd, line1, strlen (line1) + 1);          ..envia la primer linea
  write (fd, line2, strlen (line2) + 1);          ...envia la segunda linea
}
```

y el cliente utiliza `read()` en las líneas 53..69

```
readLine (fd, str)

  int fd;
  char* str;
                                        ...lee una cadena terminada con el caracter nulo
{
  int n;
  do                    ...lee caracteres hasta que encuentres un caracter nulo o fin de entrada
{
n = read (fd, str, 1);
}
while (n > 0 &&*str@ != NULL);
return (n > 0);                                 ...regresa falso se fue fin de entrada
}
```

6.5 PROGRAMAS DE SOCKETS A TRAVÉS DE INTERNET

6.5.1 Programa Internet Time

El siguiente programa muestra la conexión al puerto 13 de cualquier *host INTERNET* en el mundo y muestra la fecha y hora actual de esa máquina. El programa permite tres clases de direcciones *INTERNET*:

- Si se introduce "1", se indica que es el *host* local.
- Si se introduce números, se asume que es una dirección *IP* en formato A.B.C.D y es convertida en una dirección *IP* de 32 *bits* por *software*.
- Si se introduce una cadena, se asume que es el nombre simbólico de un *host*, y es convertido en una dirección *IP* de 32 *bits* por *software*.

He aquí una muestra del ejemplo del programa "*INTERNET Time*":

```
armando> inettime                                   ...corre el programa
Nombre del Host name (s = salir, l = local): l      ...fecha en máquina local
El nombre de este host es sysul3
Dirección Internet = 132.248.7.4
La fecha en el puerto de destino es Wed Nov 23 17:03:50 1994
```

```
Nombre del Host name (s = salir, l = local): sysul2                    ...fecha en sysul2
Dirección Internet = 132.248.7.3
La fecha en el puerto de destino es Wed Nov 23 17:04:55 1994

Nombre del Host name (s = salir, l = local): 132.248.7.2    ...prueba sysul1.fisicacu.unam.mx
La fecha en el puerto de destino es Wed Nov 23 16:55:02 1994

Nombre del Host name (s = salir, l = local): s                         ...termina programa
armando> _
```

El Listado del programa *time* se encuentra en el apéndice E.4.

Cuando se crea el *socket* del cliente, el único truco es determinar la dirección *IP* de 32 *bits*. `PromptForINETAddress()` [59] toma el nombre del *host* del usuario, y entonces se invoca `nameToAddr()` [80], para convertirlo en una dirección *IP*. Si el usuario introduce una cadena que inicia con un dígito, `inet_addr()` es llamada para desarrollar la conversión:

Si la cadena no empieza con un dígito, el siguiente paso es ver si es solo una "`l`", lo cual significa el *host* local. El nombre del *host* local se obtiene con la función `gethostname()` [97]:

Una vez que el nombre simbólico del *host* es determinado, el siguiente paso, es observar si se encuentra dado de alta en el archivo de *host* de la red `/etc/hosts`. Esto lo ejecuta la función `gethostbyname()` [104]:

El campo **`h_addr`** contiene el número *IP* asociado al *host* en un subcampo llamado **`s_addr`**. Antes de regresar este número *IP*, el programa despliega la descripción de la dirección *IP* llamando a la función `inet_ntoa()` [109]:

La dirección final de 32 *bits* es regresada en la línea 110. Una vez que la dirección *IP* ha sido determinada (almacenada en **`inetAddress`**), los campos de dirección del *socket* cliente son llenados en las líneas 37..40

```
bzero ((char*)&serverINETAddress,sizeof(serverINETAddress));
serverINETAddress.sin_family = AF_INET;                    ...utiliza INTERNET
serverINETAddress.sin_addr.s_addr = inetAddress;                ...dirección IP
serverINETAddress.sin_port = htons (DAYTIME_PORT);
```

El paso final es crear el *socket* del cliente e intentar la conexión. El código para realizar esto, es el mismo que para los *sockets* `AF_UNIX`:

```
clientFd = socket (AF_INET, SOCK_STREAM, DEFAULT_PROTOCOL);
do                                  ...haz un loop hasta que se realice una conexión con el servidor
{
  result = connect (clientFd,serverSockAddrPtr,serverLen);
  if (result == -1) sleep (1);                  ...espera 1 seg. y vuelve a intentar
}
while (result == -1);
```

6.5.2 Programa de Internet Shell

El siguiente programa es un pequeño *shell* para *UNIX* que funciona como el *shell* original, solo que tiene la posibilidad de crear conexiones por medio de *sockets*. Este programa puede ejecutarse en uno o varios *host*. A continuación se muestra un ejemplo de la creación de un *socket* de servidor, y un *socket* de cliente en el mismo *host*; los datos que se envían es la información obtenida por el comando

rwho, el cual muestra a los usuarios que se encuentran conectados, en ese momento a diferentes máquinas del mismo dominio:

```
armando> ish                                              ...inicia el shell INTERNET.
Internet Shell.
? rwho @>s rwho.sck &                  ...el socket servidor envia datos al puerto "rwho.sck".
? [1778]
? ls                                                      ...muestra lista de archivos.
a.out          como.1          cuales.1        hora.1          ish
ish.c          ish.o           rwho.sck        quien.1
? sort @<c rwho.sck                    ... socket cliente lee datos del puerto "rwho.sck".
armando  sysul3:ttyp2 Dec  3 19:46
bunge    eunice:ttyq2 Dec  3 17:19
gerardo  sysul3:ttyp0 Dec  3 19:33
oliverio sysul1:ttyp0 Dec  3 19:46
pablo    sysul2:ttyp0 Dec  3 18:59
santiago sysul3:ttyp1 Dec  3 19:43
? ^D                                                      ...salir del shell.
Armando> _
```

Para el siguiente ejemplo, se corre el comando `netstat`, y su salida se envía al puerto número 5000; después, desde otra máquina se crea un *socket* que leerá los datos del puerto 5000 en la máquina *sysul2*:

```
sysul2.ifisicacu.unam.mx> ish                             ...ejecuta shell Internet en sysul2.
Internet Shell.
? netstat @>s 5000 &          ...el socket servidor envia datos en background al puerto 5000.
[25621]
? ^ D                                                     ...termina shell en sysul2.
sysul2.ifisicacu.unam.mx> rlogin sysul3                   ...inicia sesión en host sysul3.
passwrd:
armando> ish                                              ...ejecuta shell INTERNET en sysul3.
Internet Shell.
? cat @<c sysul2.5000                                     ...cliente lee datos del puerto 5000 en sysul2.

IP address = 132.248.7.3                                  ...dirección INTERNET de sysul2.
                                                          ...datos leídos del puerto.
Active Internet connections
Proto Recv-Q Send-Q  Local Address          Foreign Address        (state)
tcp        0      0  sysul2.5000           sysul3.1140            ESTABLISHED
tcp        0      0  sysul2.1023           sysul3.login           ESTABLISHED
tcp        0      0  sysul2.3495           condor.dgsca.una.domai TIME_WAIT
tcp        0      0  sysul2.telnet         132.248.7.5.1224       ESTABLISHED
tcp        0      0  sysul2.2455           sirio.cray.unam..telne ESTABLISHED
tcp      920      0  sysul2.1940           sirio.cray.unam..telne CLOSE_WAIT
tcp        0      0  sysul2.704            *.*                    LISTEN
udp        0      0  sysul2.elcsd          *.*
Active UNIX domain sockets
Type    Recv-Q Send-Q    Inode     Conn     Refs  Nextref Addr
dgram        0      0 801edb6c        0        0        0 /dev/elcscntlsckt
dgram        0      0 801ed97c        0        0        0 /dev/snmp
stream       0      0 801edd5c        0        0        0 /dev/printer

? ^D                                                      ...termina shell.
armando> ^D                                               ...salir del host sysul3.
Connection closed.
sysul2.ifisicacu.unam.mx> _                               ...regreso a host sysul2.
```

He aquí una ilustración de la conexión del *socket*

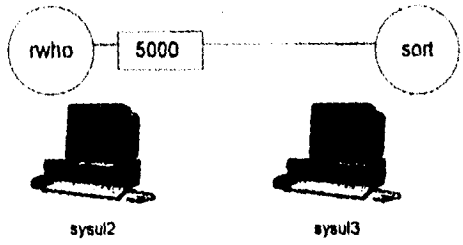


FIGURA 6-5. Conexión por medio del puerto 5000

Del ejemplo anterior, observe la primer línea del resultado del comando `netstat`; esta indica el *socket* que se creó con el número de puerto 5000.

```
Active Internet connections
Proto Recv-Q Send-Q  Local Address          Foreign Address        (state)
tcp        0      0  sysul2.5000           sysul3.1140            ESTABLISHED
tcp        0      0  sysul2.1023           sysul3.login           ESTABLISHED
....
```

El siguiente ejemplo es muy similar al anterior, ya que un *socket* servidor envía la información proporcionada por el comando `ruptime` al puerto 6000; en otra máquina (*sysul3*) se crea un *socket* cliente que lee los datos del puerto 6000 de la máquina *sysul2*, y que a su vez, crea otro *socket* servidor que enviara los datos recibidos al puerto 7000; después, en la primer máquina (*sysul2*), se crea un *socket* cliente que leerá los datos del puerto 7000 de la máquina *sysul3*; y por ultimo los mostrara a la pantalla:

```
sysul2.ifisicacu.unam.mx> ish                          ...inicia shell INTERNET en sysul2.
Internet Shell.
? ruptime @>s 6000 &                    ...servidor envia datos en background al puerto 6000.
[2001]
? ^D                                                   ...termina shell en sysul2.

sysul2.ifisicacu.unam.mx> rlogin sysul3                ...inicia sesión en sysul3.
armando> ish                                           ...inicia shell en sysul3.
Internet Shell.
? sort @<c sysul2.6000 @>s 7000 &       ...cliente lee datos de sysul2 en el puerto 6000.
[ 3756 ]                                ...y envía como servidor al puerto 7000 en sysul3.
IP address = 132.248.7.3                ...dirección INTERNET de sysul2.
? ^D                                                   ...termina shell en sysul3.

armando> ^D                                            ...termina sesión en sysul3.
Connection closed.
sysul2.ifisicacu.unam.mx> ish                          ...inicia otro shell INTERNET en sysul2.
Internet Shell.
? cat @<c sysul3.7000                                  ...leer datos del puerto 7000 en sysul3.
IP address = 132.248.7.4                               ...dirección INTERNET del host sysul3.
```

```
eunice      up 14+22:59,      0 users,  load 1.00, 1.00, 0.90
feynmann    up  1+05:55,      0 users,  load 0.00, 0.00, 0.00
moon      down  1+01:04
nadxeli   down 88+06:52
sysul1      up 13+05:50,      0 users,  load 0.00, 0.00, 0.00
sysul2      up 13+05:57,      1 user,   load 3.08, 3.00, 3.00
sysul3      up  1+01:43,      2 users,  load 0.00, 0.00, 0.00
teorica0  down 155+06:20
teorica1    down ??:??
teorica3  down 180+09:11
teorica5    down ??:??
titan     down 29+08:34
varea       down ??:??

? ^D                                                    ...termina shell.
sysul2.ifisicacu.unam.mx> _
```

He aqui una ilustración de la conexión del *socket*

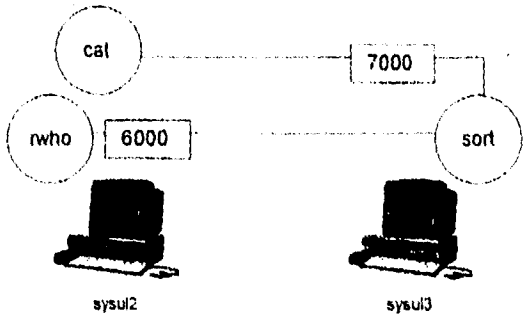


FIGURA 6-6. Conexión por medio del puerto 5000

En este último ejemplo, se crea un *socket* cliente que solicita servicio al puerto perfectamente conocido DAYTIME PORT (13), y por lo tanto mostrara la hora actual de ese *host*.

```
armando> ish                                            ...inicia el shell internet.
Internet Shell.
? cat @<c sysul2.13                                     ...cliente lee datos de la hora (DAYTIME)
                                                        ...en el host sysul2 desde el host sysul3.
IP address = 132.248.7.4                                ...dirección INTERNET del host sysul3.
Sat Dec  3 19:15:13 1994                    ...muestra los datos obtenidos del puerto 13 en sysul2.
? ^D                                                    ...salir del shell.
Armando> _
```

El listado del programa *Ish* , se encuentra en el apéndice E.5.

FALTA PAGINA

No 112 a la .....

CONCLUSIONES

El uso de la topología *Ethernet* permite un crecimiento casi transparente, es decir, no se necesitan muchos cambios para añadir más nodos; además esta red tiene un soporte muy amplio, ya que es la más utilizada en redes *LAN*.

Por otra parte, el sistema operativo *UNIX* es funcional, económico, y se adapta a la red sin muchos cambios; sin embargo como la tecnología de sistemas operativos va cambiando conforme pasa el tiempo, y nuevas tendencias van llegando, el sistema operativo puede ser sustituido por un *Windows NT*, o una nueva versión gráfica de *UNIX*, todo esto sin requerir una reconfiguración de la topología de la red.

UNIX no tiene restricciones para la comunicación con su exterior, ya que se le pueden instrumentar sin mucho esfuerzo varios caminos, como son *UUCP*, *TCP/IP*, comandos remotos, etc.

Además el protocolo *TCP* es tan flexible, que permite a los usuarios avanzados, el desarrollo de herramientas alternativas de comunicación, en otras palabras, la comunicación no se limita al uso de las herramientas preestablecidas.

Por otra parte, las herramientas existentes no requieren de un gran adiestramiento por parte de los usuarios, basta con saber utilizar los comandos básicos en el manejo de una cuenta en un *host*.

La puesta en operación del la red en la DIEEC, y su conexión a red UNAM permite:

- La comunicación entre los diversos *hosts*.
- Por último, y basándose en los servicios automáticos de conversión de direcciones, monitoreo, y mantenimiento de la red, el usuario puede hacer uso de otras herramientas, tales como:
    - *Servidores de correo*. El correo electrónico es uno de los servicios de mayor demanda.
    - *Servicios de Telnet*. Permite aprovechar los recursos remotos que no se poseen en la red local.
    - *Archie*. Permite la búsqueda electrónica de archivos referentes a un tema en los servidores anunciados para tal fin.
    - *Gophers*. Son menús que permiten buscar información en *Internet*.
    - *WWW*. Similar al *Archie* y *Gopher*, pero, que solo funciona bajo ambientes gráficos.

APÉNDICE A

La siguiente tabla muestra algunos de los grupos de noticias más populares en la *netnews*, otros grupos representativos y grupos de distribución amplia, junto con una descripción de sus temas.

Grupo de noticias	Tema
comp.ai	Inteligencia artificial
comp.databases	Cuestiones de bases de datos
comp.graphics	Gráficos por computadora
comp.lang.c	El lenguaje de programación C
comp.misc	Artículos misceláneos sobre computadoras
comp.sources.unix	Código fuente de paquetes de software del sistema UNIX
comp.text	Procesamiento de texto
comp.unix.questions	Cuestiones sobre el sistema UNIX
misc.forsale	Anuncios de objetos en venta
misc.misc	Artículos misceláneos que no encajan en otro grupo
misc.wanted	Peticiones de objetos
news.announce.confere nces	Anuncios de conferencias
news.announce.newus ers	Información para nuevos usuarios
news.list	Estadísticas sobre el uso de *USENET*
rec.arts.movies	Películas y filmaciones
rec.audio	Equipos de alta fidelidad
rec.autos	Todo lo relacionado con automóviles
rec.birds	Observación de pájaros
rec.gardens	Jardinería
rec.humor	Chistes
rec.photo	Fotografía y cámaras
rec.travel	Viajes
sci.crypt	Uso y análisis de sistemas de cifrado
sci.math	Temas matemáticos
sci.math.symbolic	Sistemas de cálculo simbólico
sci.misc	Artículos misceláneos sobre ciencias
sci.physics	Física, incluyendo nuevos descubrimientos
soc.singles	Vida de soltero
soc.women	Temas femeninos

TABLA A-1. Grupos de noticias.

APÉNDICE B

B.1 Formato del encabezado Internet

Un resumen del contenido del encabezado *Internet* se describe a continuación:

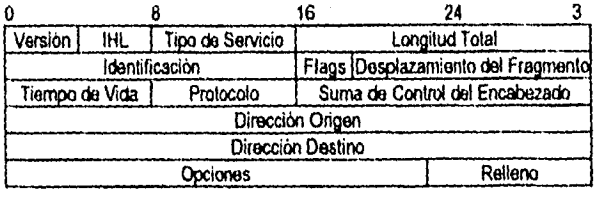


Figura B-1. Encabezado de Datagrama *Internet*

Versión: 4 bits

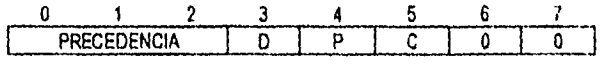
El campo de Versión indica el formato del encabezado Internet. Aquí se describe la versión 4.

Longitud del Encabezado Internet (IHL): 4 bits

La Longitud del Encabezado Internet es la longitud del encabezado Internet en palabras de 32 bits, y así apunta al inicio de la información. El valor mínimo para un encabezado correcto es 5.

Tipo de Servicio: 8 bits

El Tipo de Servicio provee una indicación de los parámetros abstractos de la calidad de servicio deseado. Estos parámetros van a ser utilizados para guiar la selección de los parámetros de servicio actuales cuando se transmite un datagrama a través de una red particular. Varias redes ofrecen servicio de precedencia, que trata de algún modo, el tráfico de alta precedencia como más importante que otro tráfico (generalmente aceptando, solamente tráfico por encima de cierta precedencia al tiempo de lectura alta). La opción mayor es una compensación de tres modos entre baja-demora, alta-productividad, y alta-confiabilidad.



Bits 0-2: PRECEDENCIA.
- 111 - Control de Red.
- 110 - Control Internetwork.
- 101 - CRITICO/ECP.
- 100 - Recorre Flash.
- 011 - Flash.
- 010 - Inmediato.
- 001 - Prioridad.
- 000 - Rutina.

Bit 3: 0 = Demora Normal, 1 = Baja Demora.

Bit 4: 0 = Productividad Normal, 1 = Alta Productividad.

Bit 5: 0 = Confiabilidad Normal, 1 = Alta Confiabilidad.

Bits 6-7: Reservado para Uso Futuro.

El uso de las indicaciones de Demora, Productividad, y Confiabilidad pueden aumentar el costo (en algún sentido) del servicio. En muchas redes el mejor desempeño para uno de estos parámetros es acoplado con un peor desempeño en otra. Excepto para casos muy inusuales, a lo más dos de estas tres indicaciones deberán ser establecidas.

El tipo de servicio es utilizado para especificar el tratamiento del datagrama durante su transmisión a través del sistema *Internet*. Se pretende que la designación de la precedencia Control de Red sea

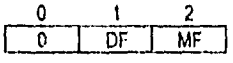
utilizada dentro de una red solamente. El uso y control actual de esa designación está hecha para cada red. La designación del Control Internetwork es para uso de creadores de control de gateway solamente. Si el uso actual de estas designaciones de precedencia conciernen a una red en particular, es responsabilidad de esa red controlar el acceso, y uso de aquellas designaciones de precedencia.

Longitud total: 16 bits
La longitud total es la longitud del datagrama, medido en octetos, incluyendo el encabezado Internet y la información. Este campo permite que la longitud de un datagrama sea de hasta 65,535 octetos. Los datagramas de este tamaño son poco prácticos para muchos hosts y redes. Todos los hosts deben estar preparados para aceptar datagramas de hasta 576 octetos (si arrivan todos o en fragmentos). Se recomienda que los hosts solamente envíen datagramas mayores de 576 octetos, si tienen la seguridad de que el destino esté preparado para aceptar estos datagramas.

El número 576 es seleccionado para permitir un bloque de información de tamaño razonable para ser transmitido, además de la información de encabezado requerida. Por ejemplo, este tamaño permite un bloque de información de 512 octetos más 64 octetos de encabezado para encajar en un datagrama. El encabezado *Internet* máximo tiene 60 octetos, y un encabezado *Internet* típico tiene 20 octetos, permitiendo un márgen para encabezados de protocolos de nivel superior.

Identificación: 16 bits
Un valor de identificación asignado por el emisor para ayudar en el ensamblado de los fragmentos de un datagrama.

Banderas (Flags): 3 bits
Diversas Banderas de Control.



Bit 0: reservado, tiene que ser cero
Bit 1: (DF) 0 = Puede Fragmentar, 1 = No Fragmentar.
Bit 2: (MF) 0 = Ultimo Fragmento, 1 = Más Fragmentos.

Desplazamiento del fragmento: 13 bits
Este campo indica a que parte del datagrama, pertenece este fragmento. El desplazamiento del fragmento es medido en unidades de 8 octetos (64 bits). El primer fragmento tiene un desplazamiento de cero.

Tiempo de Vida: 8 bits
Este campo indica el tiempo máximo que el datagrama está autorizado para permanecer en el sistema *Internet*. Si este campo contiene el valor de cero, entonces el datagrama tiene que ser destruido. Este campo es modificado en procesamiento del encabezado Internet. El tiempo es medido en segundos, pero ya que cada módulo que procesa un datagrama tiene que disminuir el *TdV* al menos en uno, aún si este procesa el datagrama en menos de un segundo, el *TdV* tiene que ser pensado solamente como un límite superior en el tiempo que un datagrama puede existir. La intención es para provocar que datagramas mal enviados sean desechados, y para señalar el máximo tiempo de vida del datagrama.

Protocolo: 8 bits
Este campo indica el protocolo del siguiente nivel utilizado en la porción de información del datagrama *Internet*.

Suma de Control del Encabezado: 16 bits
Solo existe una suma de control en el encabezado. Ya que algunos campos de encabezado cambian (e.g., tiempo de vida), este es recalculado y es verificado en cada punto que el encabezado *Internet* es procesado.

El algoritmo de suma de control es:
El campo de suma de control es el complemento a uno de 16 *bits*, de la suma de todas las palabras de 16 *bits* que se encuentran en el encabezado, complementadas a uno. Para propósitos del cálculo de la suma de control, el valor del campo de la suma de control es cero. Este método es simple para calcular la suma de control y la evidencia experimental indica que es adecuado, pero es provisional y puede ser remplazado por un procedimiento de *CRC*, dependiendo de experiencias adicionales.

Dirección Remitente: 32 bits
La Dirección del *host* o *gateway* que envía el datagrama, de acuerdo con alguno de los tres formatos mencionados anteriormente.

Dirección Destinataria: 32 bits
La Dirección del *host* o *gateway* que recibirá el datagrama y que cumpla con uno de los tres formatos mencionados anteriormente.

B.2 Formatos de mensaje ICMP
Los mensajes de *ICMP*, son enviados usando el encabezado base de *IP*. El primer octeto de la porción de datos del datagrama es un tipo de campo *ICMP*, el valor de este campo determina el formato de los datos restantes. Cualquier campo etiquetado como no utilizado, esta reservado para las posteriores extensiones y debe ser cero cuando es enviado, pero los receptores no usan estos campos (excepto cuando los incluyen en el calculo de la suma de control). A menos que se indique lo contrario en las descripciones de formato individual, los valores de cada uno de los campos del encabezado de *Internet* son los siguientes:

Versión. 4

IHL. La longitud del encabezado *Internet* en palabras de 32 *bits*, y así apunta al inicio de la información.

Tipo de servicio. 0

Longitud Total. La longitud del encabezado *Internet* y de los datos estan dadas en octetos.

Identificación, banderas, desplazamiento de fragmento. Es usada en fragmentación.

Tiempo de vida. El tiempo de vida es en segundos, como este campo es reducido en cada maquina en la que el datagrama es procesado, el valor en este campo deberá ser al menos tan grande como el número de *gateways* que este datagrama cruzará.

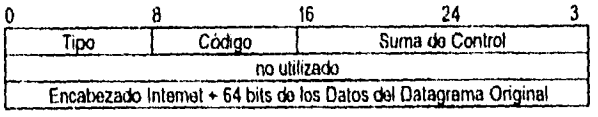
Protocolo. 1=ICMP

Suma de control del encabezado. Es un campo de 16 *bits*, complementando a uno la suma de cada uno de los campos del encabezado complementados a uno. Para calcular la suma de control, el campo de la suma de control deberá ser cero. Esta suma de control podrá ser remplazada en el futuro.

Dirección Fuente. La dirección del *gateway* o la del *host*, que compone o envía el mensaje *ICMP*. A menos que se indique lo contrario, puede ser la dirección de cualquiera de los *gateways*.

Dirección Destino. La dirección del *gateway* o la del *host* a la cual el mensaje deberá ser enviado.

B.2.1 Mensaje de Destino Inaccesible - MDI



Campos IP:
Dirección Destino. La red fuente y la dirección de los datos del datagrama original

Campos ICMP:
Tipo. 3

Código

0 = red inaccesible;
1 = host inaccesible;
2 = protocolo inaccesible;
3 = puerto inaccesible;
4 = se necesita fragmetar y que no este activa la bandera No Fragmentar (Don't Fragment);
5 = falla de ruta origen.

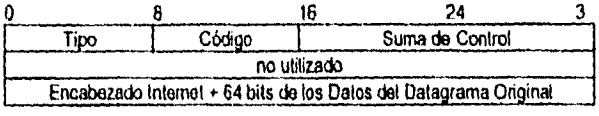
Suma de Control. Es un campo de 16 bits, complementando a uno la suma de cada uno de los campos del mensaje ICMP complementados a uno, empezando con el campo ICMP Tipo. Para calcular la suma de control, el campo de la suma de control debera ser cero. Esta suma de control podra ser remplazada en el futuro.

Encabezado Internet + 64 bits de los Datos del Datagrama Original. El encabezado *Internet* más los primeros 64 *bits* de los datos del datagrama original. Estos datos son utilizados por el *host* para comparar el mensaje con el proceso apropiado. Si un protocolo de alto nivel utiliza números de puerto, estos se encuentran en los primeros 64 *bits* de los datos del datagrama original.

Descripción
Si, de acuerdo a la información en las tablas de ruteo del *gateway*, la red especificada en el campo de destino es inalcanzable, p. e., la distancia hacia la red es infinita, el *gateway* puede enviar un mensaje de destino inaccesible (*MDI*) al *host* remitente del datagrama. Es más, en algunas redes, el *gateway* puede ser capaz de determinar si el *host* destinatario es inaccesible. Los *gateway* en estas redes pueden enviar un *MDI* al *host* remitente cuando el *host* destinatario es inaccesible.

Si, en el *host* destinatario, el módulo *IP* no puede suministrar el datagrama debido a que el módulo del protocolo indicado o el puerto de proceso no esta activo, el *host* destinatario puede enviar un *MDI* al *host* remitente. Otro caso es cuando un datagrama debe ser fragmentado para ser enviado por un *gateway* y la bandera No Fragmentar esta encendida. En este caso, el *gateway* descartará el datagrama y regresará un *MDI*. Los códigos 0, 1, 4, y 5 pueden ser recibidos de un *gateway*. Los códigos 2 y 3 pueden ser recibidos de un *host*.

B.2.2 Mensaje de Tiempo Excedido - MTE



Campos IP:
Dirección Destino

Campos ICMP:
Tipo 11

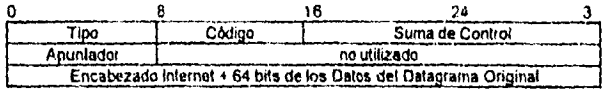
Código
0 = tiempo de vida excedido en el tránsito;
1 = tiempo de reensamblado del fragmento excedido.

Suma de Control. Igual a la suma de control de *MDI*

Encabezado Internet + 64 bits de los Datos del Datagrama Original. Igual al campo *MDI*

Descripción
Si el *gateway* que esta procesando un datagrama encuentra que el campo de tiempo de vida es cero, debe descartar el datagrama. El *gateway* tambien debe notificar al *host* remitente a traves del mensaje de tiempo excedido (*MTE*) Si un *host* que esta reensamblando un datagrama fragmentado, no puede rearmarlo debido a que le hacen falta algunos fragmentos con su limite de tiempo, descartará el datagrama y enviara un *MTE*. El código 0 puede ser recibido de un *gateway*. El código 1 puede ser recibido de un *host*.

B.2.3 Mensaje de Problema en Parametro - MPP



Campos IP:
Dirección Destino

Campos ICMP:
Tipo 12

Código 0 = el apuntador indica el error

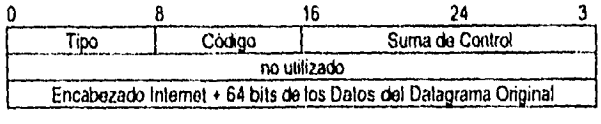
Suma de Control. Igual a la suma de control de *MDI*

Apuntador. Si el código es cero, identifica el octeto donde un error fue detectado
Encabezado Internet + 64 bits de los Datos del Datagrama Original. Igual al campo *MDI*

Descripción
Si el *gateway* o *host* que este procesando un datagrama, encuentra un problema con los parametros del encabezado, de tal forma que no pueda completar el proceso, deberá descartar el datagrama. Una fuente potencial de este tipo de problemas es con argumentos incorrectos en una opción. El *gateway* o *host* deben notificar al *host* remitente a traves del *MPP*. Este mensaje solo es enviado si el error ocasiona que el datagrama sea descartado.

El apuntador identifica el octeto del encabezado original del datagrama en donde se detecto el error (puede estar en la mitad de una opción). Por ejemplo, 1 indica que algo esta mal en el campo Tipo de servicio, y (si hay opciones presentes), 20 indica que algo esta mal con el tipo de código de la primera opción. El código 0 puede ser recibido de un *gateway* o un *host* indistintamente.

B.2.4 Mensaje de Extinguir Fuente - MEF



Campos IP:
Dirección Destino

Campos ICMP:
Tipo 4

Código 0

Suma de Control. Igual a la suma de control de *MDI*

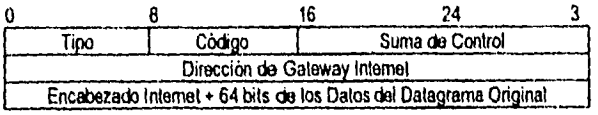
Encabezado Internet + 64 bits de los Datos del Datagrama Original. Igual al campo *MDI*

Descripción
Un *gateway* puede descartar datagramas *Internet*, si no tiene el suficiente espacio de memoria requerido para colocar los datagramas, para su salida hacia la siguiente red en turno, para llegar a su

red de destino. Si un *gateway* descarta un datagrama, deberá enviar un *MEF* al *host Internet* que remite el datagrama. Un *host* destinatario puede tambien enviar un *MEF* si los datagramas arrivan demasiado rápido para ser procesados. El *MEF* es una solicitud para que el *host* reduzca la velocidad a la cual esta enviando el tráfico al destino *Internet*. El *gateway* puede enviar un *MEF* por cada mensaje que descarta. En recepción de un *MEF*, el *host* remitente debera reducir la velocidad de transmisión hasta que no reciba más mensajes del *gateway*. El *host* remitente puede incrementar gradualmente la velocidad de transmisión hasta que reciba de nuevo un *MEF*.

El *gateway* o *host* pueden enviar un *MEF* cuando se acerque al limite de su capacidad en vez de esperar a que su capacidad sea excedida. Esto significa que el datagrama de información que provocó el mensaje puede ser entregado. El código 0 puede ser recibido de un *gateway* o un *host*.

B.2.5 Mensaje de Redirección - MR



Campos IP:
Dirección Destino

Campos ICMP:
Tipo 5

Código

0 = redirige datagramas para la red;
1 = redirige datagramas para el host;
2 = redirige datagramas para el Tipo de Servicio y la Red;
3 = redirige datagramas para el Tipo de Servicio y el Host.

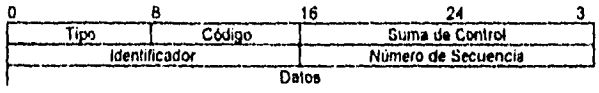
Suma de Control. Igual a la suma de control de *MDI*

Dirección De Gateway Internet. La dirección del *gateway* a la cual el tráfico para la red especificada en el campo de red destino de *Internet* de los datos del datagrama original debe ser enviado.

Encabezado Internet + 64 bits de los Datos del Datagrama Original. Igual al campo *MDI*

Descripción
El *gateway* envia un *MR* a un *host* en el siguiente caso. Un *gateway* G1, recibe un datagrama *Internet* de un *host* en una red a la cual el *gateway* esta asociado. El *gateway* G1, checa su tabla de ruteo y obtiene la dirección del siguiente *gateway*, G2, en la ruta a la red de destino *Internet* X. Si G2 y el *host* identificado por la dirección remitente *Internet* del datagrama están en la misma red, un MR es enviado al *host*. El MR advierte al *host* para que envie el tráfico para la red *X* directamente al *gateway* G2 como si fuera una ruta más corta para el destinatario. El *gateway* envia la información del datagrama original a su destinatario *Internet*. Para datagramas con las opciones de itinerario fuentes *IP* y la dirección en el campo dirección de destino, un MR no es enviado aun si existe una mejor ruta para llegar al destinatario definitivo que la siguiente dirección en el itinerario original. Los códigos 0, 1, 2, y 3 pueden ser recibidos de un *gateway*

B.2.6 Mensaje de Eco (ME) o Respuesta de Eco (MRE)



Campos IP:

Campos de Direcciones. La dirección fuente en un *ME* deberá ser la dirección destino en el *MRE*, las direcciones fuente y destino son simplemente intercambiadas y el campo Tipo es cambiado a 0, y se recalcula la suma de control.

Campos ICMP:

Tipo 8 = mensaje de eco;
0 = mensaje de respuesta de eco.

Código 0

Suma de Control. Es un campo de 16 *bits*, complemento a uno la suma de cada uno de los campos del mensaje *ICMP* complementados a uno, empezando con el campo *ICMP* Tipo. Para calcular la suma de control, el campo de la suma de control debera ser cero. Si la longitud total es impar, los datos recibidos son rellenados con un octeto de ceros para calcular la suma. Esta suma de control podra ser remplazada en el futuro.

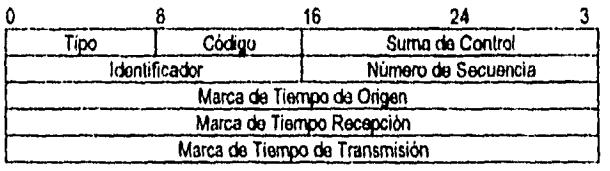
Identificador. Si el Código es cero, un identificador para ayudar a la comparación de ecos y sus respuestas, puede tener un valor de cero.

Número de Secuencia. Si el Código es cero, un número de secuencia para ayudar a la comparación de ecos y sus respuestas, puede tener un valor de cero.

Descripción

Los datos recibidos en el *ME* deben ser regresados en el *MRE*. El identificador y número de secuencia pueden ser utilizados por el emisor del eco para comparar las respuestas con las solicitudes de eco. Por ejemplo, el identificador podria ser usado como un puerto por *TCP* o *UDP* para identificar una sesión, y el número de secuencia podría ser incrementado en cada solicitud de eco enviada. El responsable del eco regresara estos mismos valores en la respuesta del eco. El Código 0 puede ser recibido de un *gateway* o un *host*.

B.2.7 Mensaje de Marca de Tiempo (MMT) o Respuesta de Marca de Tiempo (RMT)



Campos IP:

Campos de Direcciones

La dirección fuente en un *MMT* deberá ser la dirección destino en el *RMT*. Para crear un *RMT* las direcciones fuente y destino son simplemente intercambiadas y el campo Tipo es cambiado a 14, y la suma de control recalculada.

Campos ICMP:

Tipo 13 = mensaje de marca de tiempo;
14 = mensaje de respuesta de marca de tiempo.

Código 0

Suma de Control. Igual a la suma de control de *MDI*

Identificador. Si el Código es cero, un identificador para ayudar a la comparación de marcas de tiempo y sus respuestas, puede tener un valor de cero.

Número de Secuencia. Si el Código es cero, un número de secuencia para ayudar a la comparación de marcas de tiempo y sus respuestas, puede tener un valor de cero.

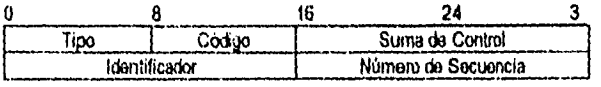
Descripción

Los datos recibidos (una marca de tiempo) en el mensaje es regresado en la respuesta junto con una marca de tiempo adicional. La marca de tiempo es un campo de 32 *bits* en milisegundos a partir de la medianoche *UT*.

El campo Marca de Tiempo de Origen indica el tiempo que el emisor ha contactado por ultima vez el mensaje antes de enviarlo, el campo Marca de Tiempo de Recepción contiene el tiempo que el emisor de ecos ha contactado por primera vez al mensaje en su recepción, y el campo Marca de Tiempo de Transmisión proporciona el tiempo que el emisor de ecos ha manipulado por última vez el mensaje al enviarlo. Si el tiempo no esta en milisegundos o no puede ser provisto con respecto a la medianoche *UT*, entonces cualquier tiempo puede ser insertado en una marca de tiempo, el bit más significativo de la marca de tiempo tambien es puesto para indicar su valor no standard.

El identificador y el número de secuencia puede ser utilizado por el emisor para ayudar a comparar las respuestas con las solicitudes. Por ejemplo, el identificador podría ser usado como un puerto por *TCP* o *UDP* para identificar una sesión, y el número de secuencia podría ser incrementado en cada solicitud enviada. El destinatario regresará estos mismos valores en la respuesta. El Código 0 puede ser recibido de un *gateway* o de un *host*.

B.2.8 Mensaje de Respuesta de Información (MRI)



Campos IP:

Campos de Direcciones

La dirección fuente en un *MSI* deberá ser la dirección destino en el *MRI*. Para crear un *MRI* las direcciones fuente y destino son simplemente intercambiadas y el campo Tipo es cambiado a 16, y la suma de control recalculada.

Campos ICMP:

Tipo 15 = mensaje de solicitud de información;
16 = mensaje de respuesta de información.

Código 0

Suma de Control. Igual a la suma de control de *MDI*

Identificador. Si el Código es cero, un identificador para ayudar a la comparación de solicitud de inf. y sus respuestas, puede tener un valor de cero.

Número de Secuencia. Si el Código es cero, un número de secuencia para ayudar a la comparación de solicitud de inf. y sus respuestas, puede tener un valor de cero.

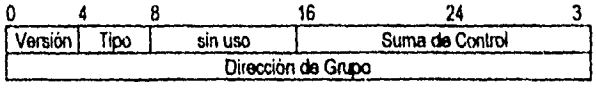
Descripción

Este mensaje puede ser enviado con la red fuente dentro del encabezado *IP*, las direcciónes de origen y destino tendrán un valor de cero (lo cual significa "esta" red). El módulo *IP* que responde deberá enviar la resouesta con la dirección especificada totalmente. Este mensaje es una forma para que un *host* pueda encontrar el número de red en la cual se encuentra.

El identificador y el número de secuencia puede ser utilizado por el emisor para ayudar a comparar las respuestas con las solicitudes. Por ejemplo, el identificador podria ser usado como un puerto por *TCP* o *UDP* para identificar una sesión, y el número de secuencia podria ser incrementado en cada solicitud enviada. El destinatario regresará estos mismos valores en la respuesta. El Código 0 puede ser recibido de un *gateway* o un *host*.

B.3 Encabezado IGMP

IGMP es usado por *hosts IP* para informar los miembros del grupo *host* a cualquier encaminador de múltiple emisión que sea vecino inmediato. El *IGMP*, es un protocolo asimétrico y aquí es especificado desde el punto de vista de un *host*, en vez del de un encaminador de multiple-emisión (*IGMP*, también puede ser usado de forma asimétrica o simétrica, entre los encaminadores de multiple-emisión). Como *ICMP*, el *IGMP* es una parte integral de *IP*. Es requerido para ser instrumentado para todos los *hosts* que conforman el nivel dos de especificaciones de multiple-emisión *IP*. Los mensajes *IGMP* son encapsulados en los datagramas *IP*, con un número de protocolo *IP* igual a dos. Todos los mensajes *IGMP* que conciernen a los *hosts* poseen el siguiente formato:



Versión. Solo se especifica la versión uno, ya que la cero ahora es obsoleta.

Tipo. Existen dos tipos de mensajes *IGMP* que atañen a los *hosts*.

1 = Consulta a los miembros *hosts* (CONSULTA)

2 = Informe de los miembros *hosts* (REPORTES)

Sin uso. Campo no utilizado, al enviarlo, es puesto en cero y es ignorado cuando se recibe.

Suma de control. Es un campo de 16 *bits*; complementando a uno la suma de cada uno de los campos de 8 octetos del mensaje *IGMP* complementados a uno. Para calcular la suma de control, el campo de la suma de control deberá ser cero.

Dirección de grupo. En un mensaje de CONSULTA, el campo de dirección del grupo es puesto en cero, cuando es enviado, e ignorado cuando se recibe. En un mensaje de INFORME, el campo de dirección del grupo mantiene la dirección *IP* del grupo *host*, del grupo que esta siendo reportado.

B.4 Formato del encabezado TCP

Los segmentos *TCP* son enviados como datagramas *Internet*. El encabezado del Protocolo *INTERNET* lleva varios campos de información, incluyendo las direcciones del *host* remitente y destinatario. El encabezado de *TCP* sigue al encabezado *Internet*, suministrando información especifica al protocolo *TCP*. Esta división permite la existencia de protocolos de nivel de *host* diferentes a *TCP*.

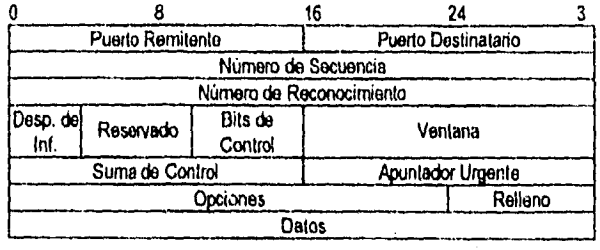


Figura B-2. Encabezado de *TCP*

Puerto Remitente: 16 bits El número del puerto remitente.

Puerto Destinatario: 16 bits El número del puerto destinatario.

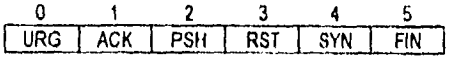
Número de Secuencia: 32 bits. El número de secuencia del primer octeto de información en este segmento (excepto cuando *SYN* esta presente). Si *SYN* esta presente, el número de secuencia es el número de secuencia inicial (*ISN*) y el primer octeto de información es *ISN+1*.

Número de Reconocimiento: 32 bits. Si el *bit* de control *ACK* es establecido, este campo contiene el valor del siguiente número de secuencia que el emisor del segmento está esperando recibir. Una vez que una conexión es establecida, este es enviado siempre.

Desplazamiento de Información: 4 bits. El número de palabras de 32 *bits* en el encabezado de *TCP*. Este indica donde comienza la información. El encabezado de *TCP* (aún aquella que incluye opciones) es un múltiplo de 32 *bits* de longitud.

Reservado: 6 bits. Reservado para futura utilización. Tiene que ser cero.

Bits de Control: 6 bits (de izquierda a derecha)



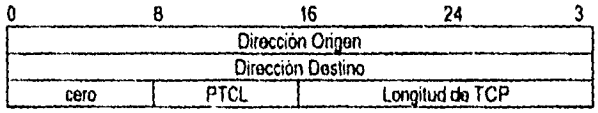
URG	Campo Significativo del Apuntador Urgente,
ACK	Campo Significativo de Reconocimiento;
PSH	Función PUSH;
RST	Restablece la Conexión,
SYN	Números de Secuencia de Sincronía,
FIN	No hay más Información del Emisor.

Ventana: 16 bits

El número de octetos de información, comenzando con el indicado en el campo de reconocimiento que el emisor de este segmento esta dispuesto a aceptar.

Suma de Control: 16 bits
Es un campo de 16 *bits*, complementando a uno la suma de cada uno de los campos de 16 *bits* del encabezado y texto complementados a uno. Si un segmento contiene un número impar de octetos de encabezado y texto, al último octeto se le rellena con ceros para formar una palabra de 16 *bits* para propósitos del calculo de la suma de control. El relleno no es transmitido como parte del segmento. Para calcular la suma de control, el campo de la suma de control deberá ser cero.

La suma de control también cubre unos 96 *bits* de un pseudo encabezado añadido conceptualmente al principio del encabezado de *TCP*. Este pseudo encabezado contiene la Dirección Remitente, la Dirección Destinataria, el Protocolo, y la longitud *TCP*. Este da la protección *TCP* contra segmentos mal enviados. Esta información es llevada en el Protocolo *INTERNET* y es transferida a través de la interfaz *TCP*/Red en los argumentos o resultados de las llamadas hechas por el *TCP* en *IP*.



La longitud de *TCP* es la longitud del encabezado de *TCP* más la longitud de información en octetos (esta no es una cantidad explícitamente transmitida, pero es calculada), y no cuenta los 12 octetos del pseudo encabezado.

Apuntador Urgente: 16 bits. Este campo comunica el valor actual del apuntador urgente como un desplazamiento positivo del número de secuencia en este segmento. El apuntador urgente apunta al número de secuencia del octeto que sigue la información urgente. Este campo solamente es interpretado en segmentos con el *bit* de control *URG* establecido.

Opciones: Variable. Las opciones pueden ocupar espacio al final del encabezado *TCP* y su longitud es un múltiplo de 8 *bits*. Todas las opciones son incluidas en la suma de control. Una opción puede comenzar en cualquier octeto fronterizo. Hay dos casos para el formato de una opción:

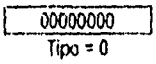
Caso 1: Un octeto único de opción-clase
Caso 2: Un octeto de opción-clase, un octeto de opción-longitud, y los octetos actuales de opción-información.

La opción-longitud cuenta los dos octetos de opción-clase y opción-longitud, así como los octetos de opción-información. Observe que la lista de opciones puede ser más corta que lo que el campo de desplazamiento de información podría implicar. El contenido del encabezado posterior a la opción Fin-de-Opción tiene que ser rellenado (i.e., con ceros). El *TCP* tiene que instrumentar todas las opciones. Las opciones actualmente definidas incluyen (el tipo esta indicado en octal):

Tipo	Longitud	Significado
0	-	Fin de Lista de Opción.
1	-	No-Operación.
2	4	Maximo Tamaño de Segmento.

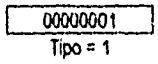
Definiciones de Opción Específicas

Fin de Lista de Opción



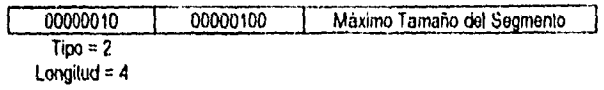
Este código de opción indica el fin de la lista de opciones. Esta podría no coincidir con el fin del encabezado de *TCP* de acuerdo con el campo de Desplazamiento de Información. Este es utilizado al final de todas las opciones, no al final de cada opción, y necesita ser utilizado solo si el fin de las opciones no coinciden de otra forma con el fin del encabezado de *TCP*.

No-Operación



Este código de opción puede ser utilizado entre opciones, por ejemplo, para alinear el comienzo de una opción posterior en una palabra fronteriza. No hay garantía que los remitentes utilizarán esta opción, así que los destinatarios tienen que estar preparados para procesar opciones aún si ellos no empiezan en una palabra fronteriza.

Máximo Tamaño de Segmento



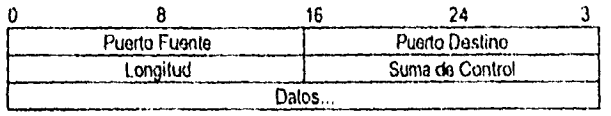
Máximo Tamaño de Segmento: 16 *bits*

Si esta opción esta presente, entonces comunica el máximo tamaño de segmento recibido en el *TCP* que envía este segmento. Este campo tiene que ser enviado solamente en la solicitud de conexión inicial (i.e., en segmentos con el bit de control SYN establecido). Si esta opción no es utilizada, cualquier tamaño de segmento es permitido.

Relleno: Variable

El relleno del encabezado de *TCP* es utilizado para garantizar que los extremos del encabezado de *TCP* y la información comiencen en un múltiplo de 32 *bits*. El relleno está compuesto de ceros.

B.5 Formato del encabezado UDP

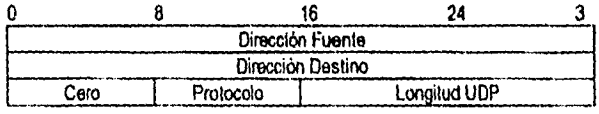


Formato de encauzado de los datagramas de usuario.

Campos. El campo *puerto fuente* es un campo opcional, cuando este presente; indica el puerto del proceso que lo envía, y puede utilizarse para indicar el puerto al cual debe ser enviada una contestación, en ausencia de cualquier otra información; si no esta presente, el valor cero es colocado.

- El *puerto destino* tiene una particular dirección *Internet* destino.
- El campo *longitud* es el tamaño en octetos del datagrama de usuario, incluyendo los encabezados y los datos. (esto significa que el valor mínimo de la longitud es ocho)

La *suma de control* es un campo de 16 *bits*, y es el complemento a uno, *de* la suma de los campos, del pseudo encabezado de información del encabezado *IP*, el encabezado UDP y los datos, rellenando con cero los octetos del final (si es necesario) para hacer un que el tamaño sea un múltiplo de 2 octetos; además, cada campo esta en complemento a uno. Conceptualmente, el pseudo encabezado se antepone al encabezado *UDP*, y contiene la dirección fuente, la dirección destino, el protocolo y la longitud de *UDP*. Esta información proporciona protección contra un mal envío de datagramas; el procedimiento de suma de control es el mismo que el usado en *TCP*.



Si el calculo de la suma de control, es cero, esta es transmitida con unos (el equivalente al complemento a uno aritmético). Un valor de cero transmitido en la suma, significa que el transmisor no ha generado la suma de control (para depuración o para protocolos de nivel más altos, esto no es importante).

Interfaz de usuario
Una interfaz de usuario debe permitir:
- La creación de un nuevo puerto de recepción;
- Operaciones de recepción en los puertos de recepción que regresen los octetos de datos, y una indicación del puerto y la dirección fuente;
- Una operación que permita a un datagrama ser enviado, especificando los datos, puertos fuente y destino, así como las direcciónes para ser enviado.

Interfaz IP
El módulo *UDP* debe ser capaz de determinar la dirección *Internet* fuente y destino, el campo de protocolo del encabezado *Internet*; la interfaz UDP/IP regresaría el datagrama *Internet* completo, incluyendo todo el encabezado *Internet* como respuesta a una operación de recepción. Así, una interfaz semejante podría permitir al *UDP* pasar a *IP* un datagrama *Internet* completo con su encabezado para enviarlo. El *IP* podría verificar ciertos campos para la consistencia y calculo de la suma de control del encabezado *Internet*.

Protocolo de aplicación
El mayor uso que se le da a este protocolo es el Servidor de Nombres *Internet* (*Internet* Name Server, INS) y la Transferencia Trivial de Archivos (*Trivial File Transfer, TFT*).

Número de protocolo
Este protocolo es el número 17 (21 en base ocho cuando se usa en el protocolo *Internet*).

APÉNDICE C

C.1 Las órdenes remotas Berkeley

Las órdenes remotas Berkeley, fueron desarrolladas originalmente como parte del Sistema BSD. Generalmente se les conoce como las órdenes r*, ya que sus nombres comienzan con la letra r, de modo que r* se corresponde con todos sus nombres cuando el * se considera un metacaracter.

Pueden ser utilizadas estas órdenes para llevar a cabo muchas tareas diferentes sobre máquinas remotas conectadas a su máquina a través de una red *TCP/IP*. Las órdenes más utilizadas son *rcp* (copia remota), utilizada para transferir archivos; *rsh* (*shell* remoto), y el *rlogin* (*login* remoto) utilizada para presentarse en una máquina remota. Las órdenes remotas permiten utilizar recursos en otras máquinas. Esto permite tratar a una red de computadoras como si fuera una maquina única.

A nivel anfitrión, cada sistema en la red *TCP/IP* contiene un archivo llamado `/etc/host.equiv`. Este archivo con una lista de las máquinas remotas que pueden abrir sesión remota sin suministrar una contraseña. Un usuario en su directorio raiz en una máquina remota puede tener el archivo `.rhosts`, para permitir o denegar el acceso a la máquima y al usuario que este tratando de obtener acceso. Este archivo define usuarios "equivalentes", a los que se les proporcionan los mismos privilegios de acceso.

C.2 Ejemplos de copiado con rcp

En el capitulo 2 se describió la forma general para copiar archivos con el comando `rcp`, pero además puede cambiar el nombre del archivo al copiarlo especificando un nuevo nombre de archivo. El formato general es el siguiente:

```
$ rcp «máquina»: «nombre-camino» «directorio/archivo»
```

por ejemplo:

```
$ rcp  sysul1:/santiago/gato.txt      /yopo/datos/santiago/datos09.txt
```

copia el archivo `/santiago/gato.txt` de *sysul1* al archivo `/yopo/datos/santiago` en su máquina local.

Incluso se puede utilizar el *rcp* para copiar un archivo de su máquina local hasta una máquina remota. Debe tener permiso de escritura en el directorio de la máquina remota al que desea copiar el archivo. Su formato es el siguiente:

```
$ rcp «archivo» «máquina»: «directorio»
```

Por ejemplo, para copiar el archivo `/armando/cosas` en el anfitrion remoto *sysul1*, llamándole `/rene/datos/listados`, utilice la orden .

```
$ rcp  /armando/cosas     sysul1: /rene/datos
```

Si desea cambiar el nombre del archivo al copiarlo solo agregue el nombre del archivo, por ejemplo,

```
$ rcp  /armando/cosas     sysul1: /rene/datos/cosas2.txt
```

Si se especifica la opción *-p*, *rcp* intentará retener los permisos y tiempos del archivo tras la copia. Si se utiliza la opción *-r* (recursivo), *rcp* copiará el subárbol; en este caso, el destino debe ser el nombre de un directorio. Por ejemplo, para copiar el directorio `/rene/datos` de la máquina remota *sysul1* al directorio `/rene/tesis` en la máquina local escriba:

```
$ rcp  -r sysul1: /rene/datos     /rene/tesis
```

El siguiente ejemplo ilustra como copiar un directorio local en un directorio especificado en la máquina remota:

```
$ rcp  -r /rene/datos   sysul45: /rene/tesis
```

Tenga cuidado al utilizar metacaracteres *shell* con la orden *rcp*. Los metacaracteres *shell* son interpretados en su máquina local ya que en la máquina remota se interpretan si se utilizan caracteres de escape o signos de acotación. Por ejemplo suponga que se desea copiar los archivos `/rene/rfc12`, `/rene/rfc123` y `/rene/rfc249` en la máquina remota *sysul1* y que su directorio actual en la máquina local tiene archivos de nombre `rfoste`, `retexto` y `rtcprog`. Para copiar los archivos `/rene/rfc12`, `/rene/rfc123` y `/rene/rfc249` de *sysul1* a su directorio actual, teclee:

```
$ rcp  sysul1: /rene/rfc\*
```

o utilice:

```
$ rcp  \' sysul1: /rene/* \'
```

C.3 Comandos ftp

A continuación se presenta la tabla C-1, con los comandos usados por el programa *ftp*.

append	dir	mdelete	put	status
ascii	desunir	mdir	pwd	struct
binary	format	mget	get	sunique
bye	put	mkdir	citar	type
case	glob	mls	recv	user
cd	hash	mode	remote help	verbose
cdup	help	mput	rename	vms
close	image	open	reset	?
delete	lcd	prompt	rmdir	
debug	ls	sendport	send	

TABLA C-1. Comandos *ftp*

C.4 Ejemplo de una sesión FTP

A continuación se proporciona un ejemplo de una sesión *FTP*

```
ftp> open nic.wiscnet.net

220 noc FTP server (Ultrix Versión 4.1 Tue May 8 07:03:17
EDT 1990) ready
Connected to nic.wiscnet.net.

Name (nic.wiscnet.net:8098document): anonymous
331 Guest login ok, send ident as password.
Password:(Escriba su dirección Internet, ésta no podrá ser desplegada cuando la
teclee)

230 Guest login ok, access restrictions apply.

ftp> dir
200 PORT command successful.
150 Opening data connection for /bin/ls (134.48.20.4,1096)
(0 bytes)>
total 18
```

```
-r--rw-r--  1     276   89     1312  Apr   18    17:14 ABSTRACTS
-r--rw-r--  1     0     89     810   Apr   18    17:14 INDEX
-r--rw-r--  1     276   89     1044  Apr   18    17:13 README
dr-xr-xr-x  2     0     0      512   Apr   9     17:55 bin
dr-xrwxr-x  4     276   89     512   Feb   19    17:00 desktoptcpip
drwxr-xr-x  2     0     84     512   Apr   9     17:45 etc
drwxr-xr-x  5     0     0      512   Apr   2     14:00 lookup
dr-xrwxr-x  2     276   79     512   Mar   14    10:38 maps
dr-xr-xr-x  2     276   84     512   Feb   21    17:43 notes
dr-xr-xr-x  2     276   84     512   Feb   21    17:44 people
drwxr-xr-x  2     270   84     512   Apr   23    12:15 policies
dr-xr-xr-x  2     276   84     512   Nov   9     1990  pub
dr-xrwxr-x  8     276   79     512   Feb   21    17:44 techinfo
drwxr-xr-x  2     283   84     512   May   16    11:22 techpartners
dr-xrwxr-x  4     276   89     512   Apr   18    16:38 userinfo
drwxr-xr-x  7     0     84     512   Mar   14    16:42 wiscnet
226 Transfer complete.
1011 bytes received in 00:00:02.46 seconds


ftp> cd wiscnet
250 CWD command successful.

ftp> dir
200 PORT command successful.
150 Opening data connection for /bin/ls (134.48.20.4,1097)
(0 bytes).
total 8
-rw-r--r--  1     0     84     767   Mar   4     17:13 ABSTRACTS
-rw-r--r--  1     0     84     425   Mar   4     17:14 INDEX
-rw-r--r--  1     0     84     968   Apr   18    17:29 README
drwxr-xr-x  2     0     84     512   Mar   14    16:47 admin
drwxr-xr-x  2     0     84     512   Apr   18    19:55 maps
drwxrwxr-x  2     0     79     512   Mar   14    16:51 notes
drwxr-xr-x  2     0     84     512   Mar   4     17:44 siteplanning
drwxr-xr-x  2     0     84     512   Feb   22    10:32 userguide
226 Transfer complete.
512 bytes received in 00:00:01.20 seconds

ftp> cd userguide
250 CWD command successful.

ftp> dir
200 PORT command successful.
150 Opening data connection for /bin/ls (134.48.20.4,1100)
(0 bytes).
total 254
-rw-r--r--  1     0     84     2277  Feb   22    10:32 README
-rw-r--r--  1     0     84     1203  Feb   22    10:32 append-a
-rw-r--r--  1     0     84     1389  Feb   22    10:32 append-b
-rw-r--r--  1     0     84     2486  Feb   22    10:32 append-c
-rw-r--r--  1     0     84     7106  Feb   22    10:32 append-d
-rw-r--r--  1     0     84     11418 Feb   22    10:32 append-e
-rw-r--r--  1     0     84     6389  Feb   22    10:32 chapter1
-rw-r--r--  1     0     84     5480  Feb   22    10:32 chapter2
-rw-r--r--  1     0     84     3318  Feb   22    10:32 chapter3
-rw-r--r--  1     0     84     6353  Feb   22    10:32 chapter4
-rw-r--r--  1     0     84     5848  Feb   22    10:32 chapter5
-rw-r--r--  1     0     84     1308  Feb   22    10:32 contents
-rw-r--r--  1     0     84     9162  Feb   22    10:32 discgroups
-rw-r--r--  1     0     84     9760  Feb   22    10:32 email
```

```
-rw-r--r--  1      0      84     9333   Feb    22      10:32 ftp
-rw-r--r--  1      0      84     4362   Feb    22      10:32 libraries
-rw-r--r--  1      0      84     1676   Feb    22      10:32 preface
-rw-r--r--  1      0      84     2059   Feb    22      10:32 supercomputer
-rw-r--r--  1      0      84     3700   Feb    22      10:32 telnet
-rw-r--r--  1      0      84     53975  Feb    22      10:32 userguide
-rw-r--r--  1      0      84     1872   Feb    22      10:32 whatiswiscnet
-rw-r--r--  1      0      84     94643  Mar    18      12:00 wholedoc
-rw-r--r--  1      0      84     1054   Feb    22      10:32 wndocstds
226 Transfer complete.
1485 bytes received in 00:00:03.24 seconds

ftp> get chapter1 chapter1.txt
200 PORT command successful.
150 Opening data connection for ftp (134.48.20.4,1102) (9333 bytes).
226 Transfer complete.
local: chapter1.txt remote: chapter1
9572 bytes received in 00:00:02.05 seconds

ftp> close
200 PORT command successful.

ftp> bye
```

C.5 Opciones y ejemplos del comando netstat

La tabla C-2, muestra una breve descripción de cada una de las opciones del comando *netstat*.

Opciones	Descripción
-A	Despliega la dirección de cualquier protocolo asociado en bloques de control usados para el *debugging*.
-a	Despliega la información por todos los *sockets*. Normalmente *sockets* utilizados por procesos servidores que no son mostrados.
-f	Los límites estadísticos o la dirección de los reportes de bloques de control para estos de la dirección de familia especificada. Las familias de Dirección reconocidas son *inet* , para *AF INET*, y unix, para *AF UNIX*.
-h	Despliega el estado de la tabla de *host IMP*.
-I	Interfaz muestra la información solamente acerca de esta interfaz.
-I interfaces	Despliega el *ADN Ethernet* contadores de Capa de Enlace de Información para una interfaz de *Ethernet*. Despliega el *ADN FDDI* contadores de Capa de Enlace de Información, condición y características de adaptador para una interfaz de *FDDI*.
-i	Presenta el estado de la información para la autoconfiguración de la interfaz. Interfaz estática configurada a un sistema, pero no localizada en tiempo de inicialización; no son mostrados.
-m	Despliega la información para rutinas de administración de memoria La red logra una parte privada de memoria.
-n	Despliega las direcciones de red como números. Normalmente `netsat` interpreta las direcciones e intenta mostrarlos estos simbólicamente.
-r	Despliega las tablas de ruteo. Cuando `-s` también puesto, despliega estadísticas de encauzado en su lugar.
-s	Despliega estadísticas por- protocolo.
-t	Despliega el tiempo que comienza de rutina de perro guardián de interfaz amba (utilizada solamente en conjunto con la opción `-i`).

TABLA C-2. Opciones del comando *netstat*

Ahora se presentan algunos ejemplos de netstat.

```
$ netstat -s
ip:
        53117 total packets received
        0 bad header checksums
        0 with size smaller than minimum
        0 with data size < data length
        0 with header length < data size
        0 with data length < header length
        0 fragments received
        0 fragments dropped (dup or out of space)
        0 fragments dropped after timeout
        0 packets forwarded
        0 packets not forwardable
        0 redirects sent
icmp:
        417 calls to icmp_error
        0 errors not generated 'cuz old message was icmp
        Output histogram:
                echo reply: 101
                destination unreachable: 417
        0 messages with bad code fields
        0 messages < minimum length
        0 bad checksums
        0 messages with bad length
        Input histogram:
                echo reply: 86
                destination unreachable: 61
                echo: 101
                address mask reply: 3
        101 message responses generated
tcp:
        12123 packets sent
                8392 data packets (1004235 bytes)
                72 data packets (4195 bytes) retransmitted
                2470 ack-only packets (2073 delayed)
                5 URG only packets
                63 window probe packets
                921 window update packets
                343 control packets
        16273 packets received
                8455 acks (for 1007007 bytes)
                486 duplicate acks
                0 acks for unsent data
                8659 packets (1094554 bytes) received in-sequence
                62 completely duplicate packets (7666 bytes)
                15 packets with some dup. data (100 bytes duped)
                738 out-of-order packets (350296 bytes)
                0 packets (0 bytes) of data after window
                0 window probes
                134 window update packets
                2 packets received after close
                0 discarded for bad checksums
                0 discarded for bad header offset fields
                0 discarded because packet too short
        128 connection requests
        77 connection accepts
        186 connections established (including accepts)
        253 connections closed (including 3 drops)
        6 embryonic connections dropped
        8389 segments updated rtt (of 8582 attempts)
        106 retransmit timeouts
```

```
        0 connections dropped by rexmit timeout
    12 persist timeouts
    341 keepalive timeouts
        288 keepalive probes sent
        1 connection dropped by keepalive
udp:
    35485 total udp requests
    0 incomplete headers
    0 bad data length fields
    0 bad checksums
    29383 application port unused
    29383 total input dropped
```

$ netstat -A

```
Active Internet connections
PCB      Proto Recv-Q Send-Q  Local Address      Foreign Address    (state)
c0fc0800 tcp        0      0  sysul3.1092        ns.dgsca.una.domai TIME_WAIT
c0fea900 tcp        0      0  sysul3.ftp         ifunam.1314        ESTABLISHED
c0feaa00 tcp        0      0  sysul3.1074        ifunam.telnet      ESTABLISHED
c0feab00 tcp        0      0  sysul3.telnet      132.248.7.5.1032   ESTABLISHED
c0fead00 tcp        0      0  sysul3.telnet      132.248.7.5.1030   ESTABLISHED
c0fea700 tcp        0      0  sysul3.704         *.*                LISTEN
c0fc3400 udp        0      0  sysul3.elcsd       *.*
Active UNIX domain sockets
Address  Type   Recv-Q Send-Q    Inode     Conn     Refs  Nextref Addr
c0fe8e00 stream      0      0 801ef1b4        0        0        0 /usr/pcsa/nbu-
app-io
c0fba200 dgram       0      0 801efd54        0        0        0
/dev/elcscntlsckt
c0fba400 dgram       0      0 801efb64        0        0        0 /dev/snmp
c0fba600 stream      0      0 801f003c        0        0        0 /dev/printer
```

$ netstat -An

```
Active Internet connections
PCB      Proto Recv-Q Send-Q  Local Address      Foreign Address    (state)
c0fea900 tcp        0      0  132.248.7.4.21     132.248.7.1.1314   ESTABLISHED
c0feaa00 tcp        0      0  132.248.7.4.1074   132.248.7.1.23     ESTABLISHED
c0feab00 tcp        0      0  132.248.7.4.23     132.248.7.5.1032   ESTABLISHED
c0fead00 tcp        0      0  132.248.7.4.23     132.248.7.5.1030   ESTABLISHED
c0fea700 tcp        0      0  132.248.7.4.704    *.*                LISTEN
c0fc3400 udp        0      0  132.248.7.4.704    *.*
Active UNIX domain sockets
Address  Type   Recv-Q Send-Q    Inode     Conn     Refs  Nextref Addr
c0fe8e00 stream      0      0 801ef1b4        0        0        0 /usr/pcsa/nbu-
app-io
c0fba200 dgram       0      0 801efd54        0        0        0
/dev/elcscntlsckt
c0fba400 dgram       0      0 801efb64        0        0        0 /dev/snmp
c0fba600 stream      0      0 801f003c        0        0        0 /dev/printer
```

$ netstat -a

```
Active Internet connections (including servers)
Proto Recv-Q Send-Q  Local Address          Foreign Address        (state)
tcp        0      0  sysul3.1098           ns.dgsca.unam.mx.domai TIME_WAIT
tcp        0      0  sysul3.ftp            ifunam.1314            ESTABLISHED
tcp        0      0  sysul3.1074           ifunam.telnet          ESTABLISHED
tcp        0      0  sysul3.telnet         132.248.7.5.1032       ESTABLISHED
tcp        0      0  sysul3.telnet         132.248.7.5.1030       ESTABLISHED
tcp        0      0  sysul3.704            *.*                    LISTEN
tcp        0      0  *.printer             *.*                    LISTEN
tcp        0      0  *.pcsafstc            *.*                    LISTEN
tcp        0      0  *.shell               *.*                    LISTEN
```

```
tcp        0      0  *.login                *.*                    LISTEN
tcp        0      0  *.exec                 *.*                    LISTEN
tcp        0      0  *.telnet               *.*                    LISTEN
tcp        0      0  *.finger              *.*                    LISTEN
tcp        0      0  *.ftp                 *.*                    LISTEN
tcp        0      0  *.qotd                *.*                    LISTEN
tcp        0      0  *.daytime             *.*                    LISTEN
tcp        0      0  *.systat              *.*                    LISTEN
tcp        0      0  *.smtp                *.*                    LISTEN
tcp        0      0  *.1028                *.*                    LISTEN
tcp        0      0  *.1027                *.*                    LISTEN
tcp        0      0  *.1026                *.*                    LISTEN
tcp        0      0  *.1025                *.*                    LISTEN
tcp        0      0  *.1024                *.*                    LISTEN
tcp        0      0  *.sunrpc              *.*                    LISTEN
udp        0      0  *.1180                *.*
udp        0      0  *.1150                *.*
udp        0      0  *.1140                *.*
udp        0      0  sysul3.elcsd          *.*
udp        0      0  *.*                   *.*
udp        0      0  *.snmp-rt             *.*
udp        0      0  *.snmp                *.*
udp        0      0  *.1060                *.*
udp        0      0  *.ntalk               *.*
udp        0      0  *.talk                *.*
udp        0      0  *.biff                *.*
udp        0      0  *.time                *.*
udp        0      0  *.1058                *.*
udp        0      0  *.netbios-            *.*
udp        0      0  *.1057                *.*
udp        0      0  *.1056                *.*
udp        0      0  *.who                 *.*
udp        0      0  *.1055                *.*
udp        0      0  *.1053                *.*
udp        0      0  *.syslog              *.*
udp        0      0  *.913                 *.*
udp        0      0  *.1047                *.*
udp        0      0  *.1044                *.*
udp        0      0  *.1040                *.*
udp        0      0  *.1035                *.*
udp        0      0  *.1023                *.*
udp        0      0  *.1027                *.*
udp        0      0  *.1024                *.*
udp        0      0  *.sunrpc              *.*
Active UNIX domain sockets
Type    Recv-Q Send-Q    Inode     Conn     Refs  Nextref Addr
stream       0      0 801ef1b4        0        0        0 /usr/pcsa/nbu-app-io
dgram        0      0 801efd54        0        0        0 /dev/elcscntlsckt
dgram        0      0 801efb64        0        0        0 /dev/snmp
stream       0      0 801f003c        0        0        0 /dev/printer
```

$ netstat -an

```
Active Internet connections (including servers)
Proto Recv-Q Send-Q  Local Address          Foreign Address        (state)
tcp        0      0  132.248.7.4.1099       132.248.10.2.53        TIME_WAIT
tcp        0      0  132.248.7.4.21         132.248.7.1.1314       ESTABLISHED
tcp        0      0  132.248.7.4.1074       132.248.7.1.23         ESTABLISHED
tcp        0      0  132.248.7.4.23         132.248.7.5.1032       ESTABLISHED
tcp        0      0  132.248.7.4.23         132.248.7.5.1030       ESTABLISHED
tcp        0      0  132.248.7.4.704        *.*                    LISTEN
tcp        0      0  *.515                  *.*                    LISTEN
tcp        0      0  *.139                  *.*                    LISTEN
```

```
tcp        0      0  *.514                  *.*                    LISTEN
tcp        0      0  *.513                  *.*                    LISTEN
tcp        0      0  *.512                  *.*                    LISTEN
tcp        0      0  *.23                   *.*                    LISTEN
tcp        0      0  *.79                   *.*                    LISTEN
tcp        0      0  *.21                   *.*                    LISTEN
tcp        0      0  *.17                   *.*                    LISTEN
tcp        0      0  *.13                   *.*                    LISTEN
tcp        0      0  *.11                   *.*                    LISTEN
tcp        0      0  *.25                   *.*                    LISTEN
tcp        0      0  *.1028                 *.*                    LISTEN
tcp        0      0  *.1027                 *.*                    LISTEN
tcp        0      0  *.1026                 *.*                    LISTEN
tcp        0      0  *.1025                 *.*                    LISTEN
tcp        0      0  *.1024                 *.*                    LISTEN
tcp        0      0  *.111                  *.*                    LISTEN
udp        0      0  *.1180                 *.*
udp        0      0  *.1150                 *.*
udp        0      0  *.1140                 *.*
udp        0      0  132.248.7.4.704        *.*
udp        0      0  *.*                    *.*
udp        0      0  *.1058                 *.*
udp        0      0  *.137                  *.*
udp        0      0  *.1057                 *.*
udp        0      0  *.1056                 *.*
udp        0      0  *.513                  *.*
udp        0      0  *.1055                 *.*
udp        0      0  *.1053                 *.*
udp        0      0  *.514                  *.*
udp        0      0  *.913                  *.*
udp        0      0  *.1047                 *.*
udp        0      0  *.1044                 *.*
udp        0      0  *.1040                 *.*
udp        0      0  *.1035                 *.*
udp        0      0  *.1023                 *.*
udp        0      0  *.1027                 *.*
udp        0      0  *.1024                 *.*
udp        0      0  *.111                  *.*
Active UNIX domain sockets
Type    Recv-Q Send-Q    Inode      Conn     Refs  Nextref Addr
stream      0      0 801ef1b4        0        0        0 /usr/pcsa/nbu-app-ic
dgram       0      0 801efd54        0        0        0 /dev/elcscntlsckt
dgram       0      0 801efb64        0        0        0 /dev/snmp
stream      0      0 801f003c        0        0        0 /dev/printer
```

C.6 Opciones y ejemplos del comando ruptime

Opción	Descripción
-a	Despliega los usuarios inactivos
-d	Muestra solamente aquellos hosts que son considerados abajo.
-nn	Muestra solamente aquellos hosts con nn o más usuarios.
-r	Despliega los hosts que están activos y funcionando.
-l	Clasifica una lista de estados por promedio de carga*.
-t	Clasifica una lista de estados por uptime*.
-u	Clasifica la lista de estados por número de usuarios*.

TABLA C-3. Opciones más usadas del *ruptime*. *Para estas opciones; si hay más de una opción de clasificación, utilizan la última.

A continuación se presentan ejemplos de *ruptime*.

```
$ ruptime
eunice      up  2+20:52,     0 users,  load 0.00, 0.00, 0.00
feynmann    up  3+23:55,     4 users,  load 0.00, 0.00, 0.00
moon        up     5:23,     0 users,  load 0.34, 0.24, 0.20
nadxeli   down 76+04:13
sysul1      up  1+03:10,     2 users,  load 0.17, 0.04, 0.00
sysul2      up  1+03:18,     1 user,   load 1.00, 1.00, 1.00
sysul3      up  1+03:14,     2 users,  load 0.01, 0.00, 0.00
teorica0  down 143+03:41
teorica1     down ??:??
teorica3  down 168+06:32
teorica5     down ??:??
titan     down 17+05:54
varea        down ??:??

$ ruptime -l
sysul2      up  1+03:24,     1 user,   load 1.00, 1.00, 1.00
moon        up     5:29,     3 users,  load 0.70, 0.53, 0.34
sysul3      up  1+03:20,     3 users,  load 0.08, 0.01, 0.00
eunice      up  2+20:58,     0 users,  load 0.00, 0.00, 0.00
feynmann    up  4+00:01,     3 users,  load 0.00, 0.00, 0.00
sysul1      up  1+03:15,     2 users,  load 0.00, 0.00, 0.00
titan     down 17+06:01
nadxeli   down 76+04:19
teorica0  down 143+03:47
teorica3  down 168+06:38
teorica5     down ??:??
varea        down ??:??
teorica1     down ??:??

$ ruptime -d
nadxeli     down 76+04:19
teorica0    down 143+03:47
teorica1          down ??:??
teorica3    down 168+06:38
teorica5          down ??:??
titan       down 17+06:01
varea             down ??:??
```

APÉNDICE D

D.1 Los archivos más importantes usados por el subsistema UUCP

La siguientes tablas listan algunos de los más importantes archivos usados por *UUCP* en la Versión 2 y la honeydanber, así como una descripción de ellos. Note que el texto en paréntesis indica un prefijo usado para el archivo, así, un archivo de datos inicia con D. , un archivo de comando o de trabajo con W y así sucesivamente.

Archivo	Descripción
Del directorio	/usr/spool/uucp
data (D.)	Cuando una solicitud de transferencia es hecha, el archivo es enviado directamente. Una copia intermediaria es realizada en el directorio de *spool* y es conocida como archivo de datos.
work (W.)	Estos archivos son creados por *uucp* y *uux*, y contiene instrucciones para *uucico*.
Ejecucion (X.)	Contiene instrucciones para el demonio *uuxqt*.
Estado (STST.)	Archivos de estado, son actualizados por *uucico* durante una conexión con una máquina remota.
Lock (LCK.)	Son creados para evitar que un proceso uucico use el mimo puerto de comunicación
Temporal (TM.)	Cuando un archivo de datos es recibido durante una transacción uucp primero son escritos en un archivo temporal. Si la transacción tiene éxito este archivo es removido a su directorio final. Si no hay éxito, este archivo continua en el directorio sin causar daño alguno.
LOGFILE	El sistema uucp registra información acerca de la transferencia y condición del job. Aquí también son registrados los problemas que llegara a tener.
SYSLOG	Contiene las estadísticas de la transferencia.
ERRLOG	Los errores ocurridos en la transacción son escritos en este archivo.
Del directorio	/usr/lib/uucp
L.-devices	Contiene información sobre los puertos de comunicación usados.
L.sys	Contiene la configuración de los sistemas a los que se conecta. Por ejemplo, la velocidad de transmisión, el número telefónico, la línea, etc.
L.-dialcodes	Aquí se describen secuencias mnemotécnicas para abreviar lo números telefónicos.
USERFILE	Comandos permitidos para ejecución remota.
L.cmds	Es un archivo de configuración que a uuxqt le dice que los comandos que pueden ser ejecutados en la máquina local por un usuario remoto.

TABLA 5-3. Los archivos más importantes usados por *uucp* versión 2.

Archivo	Descripción
Configuración de archivos:	
Devices	Archivo de Configuración para dispositivos de comunicación
Dialcodes	Aquí se describen secuencias mnemotécnicas para abreviar lo números telefónicos.
Dialers	Provee un soporte de llamadas telefónicas y *modems*, específicamente describe los comandos usados para el dispositivo de llamada.
Maxuuscheds	Contienen el máximo número de procesos uusched (2 por *default*)
Maxuuxqts	Contiene el máximo número de procesos *uuxqt*
Configuración de archivos:	
Systems	Archivo de configuración para conexión de sistemas remotos.
Uutry	actualizar y guardar su salida a un archivo temporal
Shell de escritura:	
Remote.unknown	Información acerca del area desconocida .
uudemon.admin	Envia el estado de la información del *uucp* al administrador
uudemon.cleanu	Actualiza directorios y archivos de *UUCP*
uudemon.hour	Inicia la actualización cada hora
uudemon. poll	Crea archivos de trabajo temporales

TABLA 5-4. Los archivos más importantes usados por honeydanber.

D.2 Explicación detallada sobre el correo electrónico (FIGURA 5-2).

Paso 1.

El usuario en su sistema local invoca el comando `mail` para entregar un mensaje a un usuario en un sistema remoto. Por ejemplo, el usuario Gerardo en el sistema *eunice* teclea lo siguiente:

```
$ mail sysul3!yopo< HOLA
$
```

El comando *mail* es usado para enviar el archivo HOLA al usuario yopo en la máquina remota *sysul3*. Este comando representa el unico paso que desempeña el usuario Gerardo, después el programa *mail* y el subsistema *UUCP* realiza el resto.

Paso 2.

Cuando el *mail* entiende a que máquina destino tiene que enviar el *mail* (debido a la notación *!*) invoca al *uux*. Por su parte el *uux* solicita la ejecución del *rmail* para entregar el mensaje a *yopo* en *sysul3*.

Paso 3.

El *uux* crea tres archivos en el directorio de *spool* en la máquina local para especificar la solicitud.

(1) Un archivo de trabajo o de comandos con un prefijo `.C` seguido del nombre del nodo remoto (`C.sysul3`), que contiene instrucciones para enviar los otros 2 archivos.

(2) Un archivo de datos con un prefijo `.D` seguido del nombre del nodo remoto (`D.sysul3`) que contiene una copia del mensaje a enviar.

(3) Recibe y ejecuta un archivo con un prefijo `.D` seguido del nombre del nodo local (`D.eunice`) que tiene instrucciones para ejecutar un archivo en el sistema remoto, que inicia con un prefijo `.X` seguido del nombre del nodo fuente (`X.eunice`), este contiene instrucciones para ejecutar el *rmail* para enviar el mensaje.

Paso 4.

Justo antes de terminar *uux*, este invoca al "*daemon*" *uucico* para contactar al sistema remoto. El "*daemon*" es inicializado con el papel de MAESTRO, o sea que puede iniciar la conexión con el sitio remoto.

Paso 5.

El "*daemon*" *uucico* registra en el directorio de *spool* los archivos de trabajo. Cuando encuentra alguno de los archivos el "*daemon*" determina el nombre del nodo remoto, esto lo obtiene del subfijo del nombre del archivo.

Paso 6.

Entonces *uucico* intenta crear un archivo de bloqueo para evitar conversaciones por duplicado a ese sistema remoto. El archivo de bloqueo es llamado `LCK..sysul3`, cuando intenta contactar al sistema *sysul3*.

En la Versión 2, si el "*daemon*" *uucico* no puede crear un bloqueo es porque alguno está presente, o sea que tenga 90 minutos de haber iniciado. En este caso, el "*daemon*" envía un mensaje de "BLOQUEADO" (a *sysul3*) y después termina. En la versión HoneyDanBer los archivos de bloque son más amables. El proceso que crea el archivo de bloqueo, guarda en el archivo el número de identificación del proceso. Así si es necesario modificar el tiempo, el *uucico* abre el archivo de bloqueo y lee el número de identificación del proceso (*PID*). Este mecanismo permite a *uucico*

determinar sin equivocarse si el archivo de bloqueo aun esta en uso. Para eso hace una llamada al sistema (*PID* 0), que determina si un proceso *PID* existe. Si el proceso de bloqueo no existe , entonces uucico pude eliminar el archivo de bloqueo sin crear su propio archivo de bloqueo.

Paso 7.
El "*daemon*" registra a través del archivo de *systems* la correspondiente al sistema destino, *sysul3* en nuestro ejemplo. Si lo encuentra, *uucico* verifica un campo que especifica el tiempo que el sitio pueda estar conectado. Si la fecha y el tiempo actual cae dentro del rango permitido, intentara continuar en el sitio remoto. En caso contrario, el "*daemon*" escribirá un mensaje similar "*WRONG TIME TO CALL*" en el archivo de sesión y terminara.

Paso 8.
Después, *uucico* lee el archivo *devices* para obtener una linea de comunicaciòn con la velocidad correcta (velocidad de transmisión en *baudios*).

Paso 9.
El *uucico* intentará crear un archivo de bloqueo para el puerto de comunicación. Este archivo evitara que otro *uucico* o el programa *cu* utilicen el mismo puerto, en ese momento. El nombre del archivo inicia con un prefijo LCK seguido del nombre base del puerto de comunicación, por ejemplo `LCK.tty3` para el puerto *tty3*. Si el archivo de bloqueo no puede ser creado, en la Versión 2 se escribe un mensaje "*NO (DEVICE)* " <NINGÚN DISPOSITIVO> y para la HoneyDanBer "*CAN'T ACCESS DEVICE*" <INCAPAZ DE HACER EL ACCESO AL DISPOSITIVO> y después termina la sesión.

Paso 10.
Si la conexión es directa, el "*daemon*" intenta abrir el dispositivo de archivos de la línea de comunicación. Si la apertura falla *uucico* en la Versión 2 envia un mensaje "*DIRECT LINE OPEN FAILED* <FALLO EN LA APERTURA DE LA LÍNEA DIRECTA>" y en la HoneyDanBer "*CAN'T ACCESS DEVICE* <ACCESO DENEGADO AL DISPOSITIVO>" y después termina.

Paso 11.
En caso que la conexión pueda ser realizada , el archivo `systems` es consultado como información sobre el sitio. Esta información es enviada y recibida como una serie de cadenas, y como una respuesta del sitio remoto aparece el *prompt* "*login* : ", el sistema local responderá con su nombre de cuenta y su clave de acceso (password). Si la sesión en el sistema destino tiene éxito, "*SUCCEEDED* <ÉXITO> (para *sysul3*) " es escrito al archivo de sesión en el sistema en el sistema *eunice*. De otra forma escribirá en el sistema Versión 2 "*FAILED*" <FALLO> y en el sistema HoneyDanBer "*FAILED (LOGIN FAILED)* <FALLO(FALLO DE SESIÓN)>.

Paso 12.
uucp usa el archivo de contraseñas (*passwords*) en el sistema remoto (*sysul3*), mientras *uucico* en el sistema local (*eunice*) especifica al programa de sesión, que se inicializa con el papel de ESCLAVO.

Paso 13.
A continuación sigue una secuencia de intercambio iniciada por el "*daemon*" remoto ESCLAVO, enviando la cadena "`Shere=sysul3`" que le indica al sistema *eunice* que el sistema al que se contacto se llama sysul3. Si el mensaje no es recibido, *uucico* en la Versión 2, escribe un mensaje

similar "*BAD LOGIN/PASSWORD*, <SESION/ ERROR DE CONTRASEÑA>" y para la HoneyDanBer escribiría algo similar a esto "*WRONG MACHINE NAME*, <ERROR EN EL NOMBRE DE LA MAQUINA>" y después aborta el intento de sesión
El "*daemon*" uucico en el sistema *eunice* responde al mensaje "`Shere=sysul3`" enviando una cadena de la siguiente forma "`Ssysul3-Q`" seguido por un número opcional. Si este número está presente, en ambos se realiza una cuenta de la conversación. Si *eunice* como *sysul3* se ponen de acuerdo para contar su conversación la cuenta es puesta para identificar esta conversación en particular. La cuenta de conversación es una característica de seguridad que ayuda a prevenir que un sistema no sea considerado como otro sistema. Si la cuenta de la conversación se inicia, y el número enviado a *sysul3* de *eunice* no es uno esperado, *sysul3* puede enviar de nuevo una cadena indicando que el número de secuencia es incorrecta. Después de recibir esta respuesta, el "*daemon*" uucico en la Versión 2, escribirá en el archivo de sesión algo muy similar a lo siguiente "*HANDSHAKE FAILED* (*BAD SEQ*), <FALLA DE CONEXIÓN (MALA SECUENCIA)> " para la versión HoneyDanBer escribirá "*HANDSHAKE FAILED* (*BAD SEQUENCE CHECK*), <FALLA DE CONEXIÓN (REVISE LA SECUENCIA)> " y por ultimo abortará la conversación.

De forma alterna, si *sysul3* responde con una cadena que indica que una respuesta de la llamada es necesaria; uucico en la Versión 2 escribirá en el archivo de sesion lo siguiente "*HANDSHAKE FAILED* (*CB*), <FALLA DE CONEXIÓN (*CB*)> " y para la versión HoneyDanBer "*HANDSHAKE FAILED* (*CALLBACK REQUIRED*), <FALLA DE CONEXIÓN (LLAMAR DE NUEVO)> y termina. En ese momento el sistema *sysul3* intentará conectarse a *eunice* para completar la recepción del mensaje *mail*. Volver a llamar es la mejor forma de asegurarse que es el sistema correcto. Por lo tanto significa que a *sysul3* se le cargaría la llamada. *UUCP* no realiza llamada a todos los sistemas. Si *sysul3* responde con un mensaje de enterado, la conversación procede. De esta forma la verificación de secuencia es exitosa y no requiere que se llame de nuevo.

Paso 14.
El sistema remoto envía una "`P`" (para *Protocolo*) seguido por uno o más caracteres, cada uno representa un posible protocolo de linea. Estos protocolos son llamados por uucico para verificar que se encuentren disponibles. Si es encontrado alguno, regresa una U (para uso) seguido de un carácter que representa el protocolo elegido (por lo general *G*, por su inventor Greg Chesson). Si no acepta ningún protocolo el "*daemon*" regresa la cadena "`UN`" (desconocido). Al contrario si la negociación de protocolo fue exitosa, en el archivo de sesión de *eunice* escribe (tanto para la Versión 2 como para HoneyDanBer) "`OK`", en caso contrario escribe "*FAILED* <FALLO>" y aborta la conexión.

Paso 15.
Ahora los "*daemons*" uucico MAESTRO y ESCLAVO, inician con la transmisión real, enviando paquetes de 64 *bytes* , que incluyen tanto mensajes de control como los datos de los archivos encolados. Los paquetes son retransmitidos cuando la suma de control no es igual.

Existen cinco mensajes de control usados durante este proceso : `S`, para enviar un archivo; `R` para recibir el archivo; `C` para copia completa, `X` para ejecución de un comando y `H` para terminar la llamada (colgar). El "*daemon*" MAESTRO puede enviar mensajes `S`, `R` o `X` para indicar el tipo de trabajo, el "*daemon*" ESCLAVO contesta con Y (si) o N (no) por cada mensaje. Estas respuestas dependerán de los permisos para usar los archivos y ejecutar los comandos; son determinados por los archivos de seguridad (*USERFILE* y `L.cmds` en la Versión 2 y Permisions en la HoneyDanBer), así como los de los directorios en el sistema remoto.

En la medida en que cada archivo es recibido, el "*daemon*" ESCLAVO en *sysul3* escribe en un archivo temporal (en el directorio de *spool*) a que sistema pertenecen (este archivo comienza con TM.). Una vez que el archivo ha sido recibido exitosamente, es movido a su destino final (renombrado). En ese momento una copia del mensaje completo es regresado al sistema de envio (*emnice*); "CY" si el mensaje fue exitoso, y "CN" en caso de haber fallado. en este caso el archivo temporal es retenido en el directorio de *spool* - eventualmente será borrado por mantenimiento o manualmente.

Los resultados de las solicitudes y transacciones de trabajo son enviados a los archivos apropiados en el directorio de *spool* en ambos sistemas. Cada archivo transferido, se nombra como "*LOGFILE* y *SYSLOG*", ambos en la Versión 2. Cualquier error relacionado con el sistema es escrito al archivo *ERRLOG*. Para el sistema HoneyDanBer se tienen los archivos `.Admin/xferstats`, y los errores son puestos en `.Admin/errors`, ambos en el subdirectorio `/usr/spool/uucp`

Paso 16.
Después el "*daemon*" MAESTRO en el sistema local (*emnice*), procesa todas las solicitudes de trabajo, los papeles de los "*daemons*" son intercambiados; el ESCLAVO se convierte en MAESTRO y el MAESTRO en ESCLAVO, para continuar el resto de la transacción. Entonces el ESCLAVO remoto *sysul3* registra en el directorio de *spool* cualquier trabajo designado para el sistema que lo llamó (*emnice*).

Paso 17.
El nuevo "*daemon*" MAESTRO procesa todas las solicitudes de trabajo en *sysul3*. Después que las transferencias de *sysul3* a *emnice* han terminado, el MAESTRO envia al ESCLAVO una solicitud de finalización de mensaje.

Paso 18.
La solicitud de finalización de mensaje es realizada, la conexión es terminada y los archivos de bloqueo y temporales que fueron usados son borrados y ambos "*daemons*" terminan su trabajo. Si el enlace de comunicación fue interrumpido el bloqueo y los archivos temporales permanecerán como testamento de la conexión abortada. Por ultimo el "*daemon*" uucico en el sitio remoto (*sysul3*), ejecuta al "*daemon*" *uuxqt* para buscar en el directorio de *spool* trabajo.

Paso 19.
El "*daemon*" *uuxqt* busca en el directorio de `spool` y ejecuta el archivo asociado de datos (el mensaje de *mail* es recibido en *emnice*). Entonces el proceso *uuxqt* comprueba la seguridad del archivo apropiado (`L.cmds` en la Versión 2 y permisions en el sistema HoneyDanBer), para ver si esta autorizado a ejecutar el comando rmail.

Paso 20.
En caso de ser autorizado, el *uuxqt* invoca *rmail* para que envíe un mensaje de *mail* en el archivo de `mailbox` al receptor (`/usr/mail/yopo`). El programa rmail, esta restringido porque solamente puede suministrar correo, no lee correspondencia nueva. Nota: el *rmail* es un enlace entre el programa de correo *UNIX*.

APÉNDICE E

E.1 Asignaciones a números de puertos

Las tablas E-1, E-2, E-3, E-4 y E-5 muestran los números de puerto comunmente utilizados con sus nombres y descripciones.

Decimal	Octal	Descripción
0 - 63	0 - 77	Funciones estandar de Red amplia
64 - 131	100 - 203	Funciones especificas de Hosts
132 - 223	204 - 337	Reservadas para futuro uso
224 - 255	340 - 377	Funciones Experimentales

TABLA E-1. Asignaciones Generales de Números de Puertos

Decimal	Octal	Descripción
1	1	*Telnet* antiguo
3	3	*FTP* antiguo
5	5	Remote Job Entry
7	7	Echo
9	11	Descartado
11	13	Quien esta en el sistema (SYSTAT)
13	15	Fecha y Hora del sistema (DAYTIME)
15	17	Que sistemas estan arriba (NETSTAT)
17	21	Mensaje de texto corto
19	23	Generador de caracteres o TTYTST
20	24	Datos *FTP*
21	25	Control FTP
23	27	*Telnet*
25	31	*SMTP*
27	33	*NSW User System w/COMPASS FE*
29	35	*MSG-3 ICP*
31	37	MSG-3 Autenticidad
33	41	No asignado
35	43	*IO Station Spooler*
37	45	*Time Server*
39	47	No asignado
41	51	*Graphics*
42	52	*Name Server*
43	53	*Whois*
45	55	*Message Processing Module*
47	57	*NI FTP*
49	61	*RAND Network Graphics Conference*
51	63	*Message Generator Control*
53	65	*AUTODIN II FTP*
53	65	*Domain Name Server*
55	67	*ISI Graphics Language*
57	71	*MTP*
59	73	*New MIT Host Status*
61 - 63	75 - 77	No asignados

TABLA E-2. Funciones estandar de red (Asignaciones Especificas)

Decimal	Octal	Descripción
65	101	No asignado
67	103	*Datacomputer at CCA*
69	105	No asignado
69	105	*TFTP*
71	107	NETRJS (EBCDIC) en UCLA-CCN
73	111	NETRJS (ASCII-68) en UCLA-CCN
75	113	NETRJS (ASCII-63) en UCLA-CCN
77	115	Cualquier servidor RJE
79	117	Nombre o FINGER
81	121	No asignado
83	123	MIT ML *Device*
85	125	MIT ML *Device*
87	127	Cualquier enlace de terminal
89	131	SU/MIT *Telnet Gateway*
91	133	MIT *Dover Spooler*
93	135	BBN *RCC Accounting*
95	137	SUPDUP
97	141	*Datacomputer Status*
99	143	CADC - NIFTP via UCL
101	145	NPL - NIFTP via UCL
103	147	BNPL - NIFTP via UCL
105	151	CAMBRIDGE - NIFTP via UCL
107	153	HARWELL - NIFTP via UCL
109	155	SWURCC - NIFTP via UCL
111	157	ESSEX - NIFTP via UCL
113	161	RUTHERFORD - NIFTP via UCL
115 - 129	163 - 201	No asignados
131	203	*Datacomputer*

TABLA E-3. Funciones especificas de host (Asignaciones Especificas)

Decimal	Octal	Descripción
132 - 223	204 - 337	Reservados

TABLA E-4. Reservados para futuro uso (Asignaciones Especificas)

Decimal	Octal	Descripción
224 - 239	240 - 357	No asignado
241	361	*Datacomputer at CCA*
243	363	No asignado
245	365	*TFTP*
247	367	NETRJS (EBCDIC) en UCLA-CCN
249 - 255	371 - 377	NETRJS (ASCII-68) en UCLA CCN

TABLA E-5. Funciones Experimentales (Asignaciones Especificas)

La tabla E-6 muestra los argumentos de las funciones y su descripción.

`getgetpeername (socket,destaddr,addrlen)`	Determina la dirección del extremo al cual el socket esta conectado
`getsockname (socket,localaddr,addrlen)`	Determina la dirección local a la cual el socket esta asociado
`getsockopt(socket,level,optionid, optionval,length)`	Solicita información acerca del socket
`setsockopt(socket,level,optionid, optionval,length)`	Establece las opciones para el socket
`select(ndesc,indesc,outdesc, excdesc,timeout)`	Permite que un solo proceso espere conexiones de varios sockets
`gethostname(name,length)`	Obtiene el nombre de un host
`sethostname(name,length)`	Establece un nombre de host (necesita privilegios)
`getdomainname(name,length)`	Obtiene el nombre del dominio del sistema
`setdomainname(name,length)`	Establece el nombre de un dominio
`gethostbyname(namestr)`	Obtiene información correspondiente a un host indicado por su nombre
`gethostbyaddr(addr,len,type)`	Obtiene información correspondiente a un host indicado por su dirección
`getnetbyname(name)`	Obtiene información correspondiente a una red especificada por un nombre
`getnetbyaddr(netaddr,addrtype)`	Obtiene información correspondiente a una red especificada por una dirección
`getprotobyname(name)`	Obtiene información del protocolo señalado por su nombre
`getprotobynumber(number)`	Obtiene información del protocolo señalado por su número de protocolo
`getservbyname(name,proto)`	Obtiene información de un servicio por su nombre, de acuerdo a su protocolo
`getservbyport(port,proto)`	Obtiene información de un servicio por su número de puerto, de acuerdo a su protocolo
`inet_addr(string)`	Convierten el formato A.B.C.D a una dirección IP de 32 bits
`inet_network(string)`	Convierten el formato A.B.C.D a una dirección IP de 32 bits
`inet_ntoa(internetaddr)`	Convierte una dirección IP de 32 bits al formato A.B.C.D.
`inet_makeaddr(net,local)`	Combina las direcciones de red con direcciones locales y genera una dirección IP
`inet_netof(internetadri)`	Obtiene la dirección de red de una dirección IP
`inet_lnaof(internetaddr)`	Obtiene la dirección local de una dirección IP

TABLA E-6. Funciones de llamadas al Sistema.

E.2 Listado completo del programa CHEF

```
1  #include <stdio.h>
2  #include <signal.h>
3  #include <sys/types.h>
4  #include <sys/socket.h>
5  #include <sys/un.h>                                    /* para sockets AF_UNIX */
6
7  #define DEFAULT_PROTOCOL 0
8
9  /*****************************************************************************/
10
11 main ()
12
13 {
14   int serverFd, clientFd, serverLen, clientLen;
15   struct sockaddr_un serverUNIXAddress;                  /* dirección servidor */
16   struct sockaddr_un clientUNIXAddress;                  /* dirección cliente */
17   struct sockaddr* serverSockAddrPtr;                    /* apuntador a dirección de servidor */
18   struct sockaddr* clientSockAddrPtr;                    /* apuntador a dirección de cliente */
19
```

```
20                                          /* ignora las señales de terminación del proceso hijo */
21    signal (SIGCHLD, SIG_IGN);
22
23    serverSockAddrPtr = (struct sockaddr*) &serverUNIXAddress;
24    serverLen = sizeof (serverUNIXAddress);
25
26    clientSockAddrPtr = (struct sockaddr*) &clientUNIXAddress;
27    clientLen = sizeof (clientUNIXAddress);
28
29                                /* crea un socket UNIX, bidireccional, protocolo más apropiado */
30    serverFd = socket (AF_UNIX, SOCK_STREAM, DEFAULT_PROTOCOL);
31    serverUNIXAddress.sun_family = AF_UNIX;                    /* da el tipo de dominio */
32    strcpy (serverUNIXAddress.sun_path, "receta");      /* indica el nombre ("puerto") */
33    unlink ("receta");                                  /* desliga el archivo si ya existe */
34    bind (serverFd, serverSockAddrPtr, serverLen);                /* crea el archivo */
35    listen (serverFd, 5);                        /* longitud maxima de conexiones pendientes */
36
37    while (1)                                                     /* ciclo infinito */
38    {
39                                                        /* acepta una conexión de cliente */
40      clientFd = accept (serverFd, clientSockAddrPtr, &clientLen);
41
42      if (fork () == 0)                               /* crea proceso hijo para enviar receta */
43      {
44        writeReceta (clientFd);                                   /* envia la receta */
45        close (clientFd);                                        /* cierra el socket */
46        exit (0);                                    /* termina con exito el proceso hijo */
47      }
48      else
49        close (clientFd);                              /* cierra el descriptor del cliente */
50    }
51  }
52
53  /*****************************************************************************/
54
55  writeReceta (fd)
56
57    int fd;
58
59  {
60    static char* line1 = "Nueces, peras, crema, ";
61    static char* line2 = "bate, muele, acitrona y cuece.";
62    write (fd, line1, strlen (line1) + 1);                    /* escribe primer linea */
63    write (fd, line2, strlen (line2) + 1);                   /* escribe segunda linea */
64  }
65
67  /*****************          FIN  DE  LISTADO          ****************/
```

E.3 Listado completo del programa EMPLEADO

```
1   #include <stdio.h>
2   #include <signal.h>
3   #include <sys/types.h>
4   #include <sys/socket.h>
5   #include <sys/un.h>                                    /* para sockets AF_UNIX */
6
7   #define DEFAULT_PROTOCOL 0
8
9   /*****************************************************************************/
10
```

```
11 main ()
12
13 {
14   int clientFd, serverLen, result;
15   struct sockaddr_un serverUNIXAddress;
16   struct sockaddr* serverSockAddrPtr;
17
18   serverSockAddrPtr = (struct sockaddr*) &serverUNIXAddress;
19   serverLen = sizeof (serverUNIXAddress);
20
21                                /* crea un socket UNIX, bidireccional, protocolo más apropiado */
22   clientFd = socket (AF_UNIX, SOCK_STREAM, DEFAULT_PROTOCOL);
23   serverUNIXAddress.sun_family = AF_UNIX;                    /* dominio del servidor */
24   strcpy (serverUNIXAddress.sun_path, "receta");             /* nombre del servidor */
25
26   do                                    /* repite hasta que una conexión con el servidor se realice */
27   {
28     result = connect (clientFd, serverSockAddrPtr, serverLen);
29     if (result == -1) sleep (1);              /* si no se conectó, espera y prueba de nuevo */
30   }
31   while (result == -1);
32
33   readReceta (clientFd);                                          /* lee la receta */
34   close (clientFd);                                              /* cierra el socket */
35   exit (0);                                                        /* termino */
36 }
37
38 /*************************************************************************/
39
40 readReceta (fd)
41
42 int fd;
43
44 {
45   char str[200];
46
47   while (readLine (fd, str))                /* lee las líneas hasta que no haya más entradas */
48     printf ("%s\n", str);                        /* muestra la línea recibida del socket */
49 }
50
51 /*************************************************************************/
52
53 readLine (fd, str)
54
55   int fd;
56   char* str;
57
58                                    /* lee una línea que termina con el caracter nulo (NULL) */
59
60 {
61   int n;
62
63   do                          /* lee caracteres hasta que encuentres un nulo, o no haya más entradas */
64   {
65     n = read (fd, str, 1);                                  /* lee un solo caracter del socket */
66   }
67   while (n > 0 && *str++ != NULL);
68   return (n > 0);                                  /* regresa falso si ya no hay más entradas */
69 }
70
```

```
71 /*****************          FIN  DE  LISTADO          *****************/
```

E.4 Listado del Programa Internet Time

```
1   #include <stdio.h>
2   #include <signal.h>
3   #include <ctype.h>
4   #include <sys/types.h>
5   #include <sys/socket.h>
6   #include <netinet/in.h>                                  /* para sockets AF_INET */
7   #include <arpa/inet.h>
8   #include <netdb.h>
9
10  #define DAYTIME_PORT 13                                  /* puerto estandar de servicio */
11  #define DEFAULT_PROTOCOL 0
12
13  unsigned long promptForINETAddress ();
14  unsigned long nameToAddr ();
15
16 /*****************************************************************************/
17
18  main ()
19
20  {
21    int clientFd;                              /* descriptor de archivo del socket del cliente */
22    int serverLen;                          /* longitud de la estructura de la dirección del servidor */
23    int result;                                /* almacena el resultado de la conexión */
24    struct sockaddr_in serverINETAddress;                  /* dirección del servidor */
25    struct sockaddr* serverSockAddrPtr;                    /* apuntador a la dirección */
26    unsigned long inetAddress;                             /* dirección IP de 32 bits */
27
28                                                 /* da valores a las dos variables del servidor */
29    serverSockAddrPtr = (struct sockaddr*) &serverINETAddress;
30    serverLen = sizeof (serverINETAddress);                /* longitud de la dirección */
31
32    while (1)                                  /* realiza ciclos hasta que el usuario indique "q" */
33    {
34      inetAddress = promptForINETAddress ();               /* obten dirección IP de 32 bits */
35      if (inetAddress == 0) break;                                  /* termina ciclos */
36                                                 /* inicializa la estructura de la dirección */
37      bzero ((char*) &serverINETAddress,sizeof(serverINETAddress));
38      serverINETAddress.sin_family = AF_INET;                       /* utiliza Internet */
39      serverINETAddress.sin_addr.s_addr = inetAddress;               /* dirección IP */
40      serverINETAddress.sin_port = htons (DAYTIME_PORT);
41                                                                    /* create el socket del cliente */
42      clientFd = socket (AF_INET, SOCK_STREAM, DEFAULT_PROTOCOL);
43      do                                 /* repite hasta que una conexión se realice con el servidor */
44      {
45        result = connect (clientFd, serverSockAddrPtr, serverLen);
46        if (result == -1) sleep (1);              /* si no hubo conexión, espera un momento */
47      }
48      while (result == -1);
49
50      readTime (clientFd);                                  /* lee la fecha y hora del servidor */
51      close (clientFd);                                             /* cierra el socket */
52    }
54    exit (0);                                                 /* termina exitosamente */
55  }
56
```

```
57  /*************************************************************************/
58
59  unsigned long promptForINETAddress ()
60
61  {
62    char hostName [100];                    /* nombre del host servidor, numerico o simbolico */
63    unsigned long inetAddress;                                  /* dirección IP de 32 bits */
64
65                                  /* repite hasta que indiquen salir, o un nombre legal sea escrito */
66                                /* si es salir, regresa 0 (cero), sino, regresa la dirección IP del host */
67    do
68    {
69      printf ("Nombre del Host (s = salir, l = local): ");
70      scanf ("%s", hostName);                                   /* obten nombre del teclado */
71      if (strcmp (hostName, "s") == 0) return (0);                          /* salir */
72      inetAddress = nameToAddr (hostName);              /* convierte a una dirección IP */
73      if (inetAddress == 0) printf ("Nombre de host no encontrado\n");
74    }
75    while (inetAddress == 0);
76  }
77
78  /*************************************************************************/
79
80  unsigned long nameToAddr (name)
81
82    char* name;
83
84  {
85    char hostName [100];
86    struct hostent* hostStruct;
87    struct in_addr* hostNode;
88
89                                          /* convierte nombre en una dirección IP de 32 bits */
90
91                        /* Si el nombre empieza con un digito, se asume que es un valor numerico valido */
92                           /* de la dirección Internet en el formato A.B.C.D y conviertelo directamente */
93    if (isdigit (name[0])) return (inet_addr (name));
94
95    if (strcmp (name, "l") == 0)                /* toma el nombre del host de la base de datos */
96    {
97      gethostname (hostName, 100);
98      printf ("El nombre de este host es %s\n", hostName);
99    }
100   else                                      /* asume que es un nombre simbolico de host valido */
101     strcpy (hostName, name);
102
103                                       /* obten la información de la dirección de la base de datos */
104   hostStruct = gethostbyname (hostName);
105   if (hostStruct == NULL) return (0);                                  /* no se encontro */
106                                          /* extrae la dirección IP de la estructura hostent */
107   hostNode = (struct in_addr*) hostStruct->h_addr;
108                                            /* muestra la dirección en formato A.B.C.D */
109   printf ("Dirección Internet = %s\n", inet_ntoa (*hostNode));
110   return (hostNode->s_addr);                                  /* Regresa la dirección IP */
111 }
112
113 /*************************************************************************/
114
115 readTime (fd)
116
```

```
117   int fd;
118
119 {
120   char str [200];                                                    /* buffer */
121
122   printf ("La fecha en el puerto de destino es ");
123   while (readLine (fd, str))          /* lee las lineas hasta que no haya más lineas a leer */
124     printf ("%s\n", str);                    /* muestra la linea enviada por el servidor */
125 }
126
127 /*************************************************************************/
128
129 readLine (fd, str)
130
131   int fd;
132   char* str;
133
134                                  /* lee una simple linea terminada con el caracter nulo (NULL) */
135
136 {
137   int n;
138
139   do                              /* lee caracteres hasta recibir un nulo o no haya más datos a leer */
140   {
141     n = read (fd, str, 1);                              /* lee un solo caracter del socket */
142   }
143   while (n > 0 && *str++ != '\n');
144   return (n > 0);                                    /* regresa falso si ya no hay más datos */
145 }
146
147 /*****************          FIN  DE  LISTADO          ****************/
```

E.5 Listado completo del programa `Ish`

```
  1 #include <stdio.h>
  2 #include <stdlib.h>
  3 #include <string.h>
  4 #include <signal.h>
  5 #include <ctype.h>
  6 #include <sys/types.h>
  7 #include <sys/file.h>
  8 #include <sys/ioctl.h>
  9 #include <sys/socket.h>
 10 #include <sys/un.h>
 11 #include <netinet/in.h>
 12 #include <arpa/inet.h>
 13 #include <netdb.h>
 14
 15
 16                                                                  /* constantes */
 17 #define MAX_STRING_LENGTH       200
 18 #define MAX_TOKENS              100
 19 #define MAX_TOKEN_LENGTH        30
 20 #define MAX_SIMPLE              5
 21 #define MAX_PIPES               5
 22 #define NOT_FOUND               -1
 23 #define REGULAR                 -1
 24 #define DEFAULT_PERMISSION      0660
 25 #define DEFAULT_PROTOCOL        0
 26 #define DEFAULT_QUEUE_LENGTH    5
 27 #define SOCKET_SLEEP            1
```

```
28
29
30                                                          /* Identificadores */
31 enum { FALSE, TRUE };
32 enum metacharacterEnum
33 {
34   SEMICOLON, BACKGROUND, END_OF_LINE, REDIRECT_OUTPUT,
35   REDIRECT_INPUT, APPEND_OUTPUT, PIPE,
36   REDIRECT_OUTPUT_SERVER, REDIRECT_OUTPUT_CLIENT,
37   REDIRECT_INPUT_SERVER, REDIRECT_INPUT_CLIENT
38 };
39 enum builtInEnum { ECHO_BUILTIN, SETENV, GETENV, CD };
40 enum descriptorEnum { STDIN, STDOUT, STDERR };
41 enum pipeEnum { READ, WRITE };
42 enum IOEnum
43 {
44   NO_REDIRECT, FILE_REDIRECT,
45   SERVER_REDIRECT, CLIENT_REDIRECT
46 };
47 enum socketEnum { CLIENT, SERVER };
48 enum { TWO_WAY_SOCKET, INPUT_SOCKET, OUTPUT_SOCKET };
49
50
51
52 struct simple
53 {
54   char* token [MAX_TOKENS];
55   int count;
56   int outputRedirect;
57   int inputRedirect;
58   int append;
59   char *outputFile;
60   char *inputFile;
61   char *outputSocket;
62   char *inputSocket;
63 };
64
65
66
67 struct pipeline
68 {
69   struct simple simple [MAX_SIMPLE];
70   int count;
71 };
72
73
74
75 struct sequence
76 {
77   struct pipeline pipeline [MAX_PIPES];
78   int count;
79   int background;
80 };
81
82
83
84 struct sequence parseSequence ();
85 struct pipeline parsePipeline ();
86 struct simple parseSimple ();
87 char *nextToken ();
88 char *peekToken ();
89 char *lastToken ();
```

```
90 char* getToken ();
91
92
93                                                              /* Globales */
94 char* metacharacters [] = { ";", "&", "\n", ">", "<", ">>",
95                              "|", "@>s", "@>c", "@<s", "@<c", "" };
96 char* builtIns [] = { "echo", "setenv", "getenv", "cd", "" };
97 char line [MAX_STRING_LENGTH];
98 char tokens [MAX_TOKENS][MAX_TOKEN_LENGTH];
99 int tokenCount;
100 int tIndex;
101 int errorFlag;
102
103
104
105 void (*originalQuitHandler) ();
106 int quitHandler ();
107
108
109
110 char **environ;
111
112 /*************************************************************************/
113
114 main (argc, argv)
115
116   int argc;
117   char* argv [];
118
119 {
120   initialize ();
121   commandLoop ();
122   return ( 0);
123 }
124
125 /*************************************************************************/
126
127 initialize ()
128
129 {
130   printf ("Internet Shell.\n");
131
132   originalQuitHandler = signal (SIGINT, quitHandler);
133 }
134
135 /*************************************************************************/
136
137 quitHandler ()
138
139 {
140
141   printf ("\n");
142   displayPrompt ();
143 }
144
145 /*************************************************************************/
146
147 error (str)
148
149   char* str;
150
151 {
```

```
152
153   fprintf (stderr, "%s", str);
154   errorFlag = TRUE;
155 }
156
157 /*************************************************************************/
158
159 displayPrompt ()
160
161 {
162   printf ("? ");
163 }
164
165 /*************************************************************************/
166
167 commandLoop ()
168
169 {
170   struct sequence sequence;
171
172
173   while (TRUE)
174   {
175     displayPrompt ();
176     if (gets (line) == NULL) break;
177     tokenize ();
178     errorFlag = FALSE;
179
180     if (tokenCount > 1)
181     {
182       sequence = parseSequence ();
183
184
185       if (!errorFlag) executeSequence (&sequence);
186     }
187   }
188 }
189
190 /*************************************************************************/
191
192
193
194 struct sequence parseSequence ()
195
196 {
197   struct sequence q;
198
199
200   q.count = 0;
201   q.background = FALSE;
202
203   while (TRUE)
204   {
205     q.pipeline[q.count++] = parsePipeline ();
206     if (peekCode () != SEMICOLON) break;
207     nextToken ();
208   }
209
210   if (peekCode () == BACKGROUND)
211   {
212     q.background = TRUE;
213     nextToken ();
```

```
214   }
215
216   getToken (END_OF_LINE);
217   return (q);
218 }
219
220 /*************************************************************************/
221
222 struct pipeline parsePipeline ()
223
224 {
225   struct pipeline p;
226
227
228   p.count = 0;
229
230   while (TRUE)
231   {
232     p.simple[p.count++] = parseSimple ();
233     if (peekCode () != PIPE) break;
234     nextToken ();
235   }
236
237   return (p);
238 }
239
240 /*************************************************************************/
241
242 struct simple parseSimple ()
243
244 {
245   struct simple s;
246   int code;
247   int done;
248
249
250   s.count = 0;
251   s.outputFile = s.inputFile = NULL;
252   s.inputSocket = s.outputSocket = NULL;
253   s.outputRedirect = s.inputRedirect = NO_REDIRECT;
254   s.append = FALSE;
255
256   while (peekCode () == REGULAR)
257     s.token[s.count++] = nextToken ();
258
259   s.token[s.count] = NULL;
260   done = FALSE;
261
262
263   do
264   {
265     code = peekCode ();
266
267     switch (code)
268     {
269       case REDIRECT_INPUT:                                    /* < */
270         nextToken ();
271         s.inputFile = getToken (REGULAR);
272         s.inputRedirect = FILE_REDIRECT;
273         break;
274
275       case REDIRECT_OUTPUT:                                   /* > */
```

```
276       case APPEND_OUTPUT:                                          /* >> */
277         nextToken ();
278         s.outputFile = getToken (REGULAR);
279         s.outputRedirect = FILE_REDIRECT;
280         s.append = (code == APPEND_OUTPUT);
281         break;
282
283       case REDIRECT_OUTPUT_SERVER:                                 /* @>s */
284         nextToken ();
285         s.outputSocket = getToken (REGULAR);
286         s.outputRedirect = SERVER_REDIRECT;
287         break;
288
289       case REDIRECT_OUTPUT_CLIENT:                                 /* @>c */
290         nextToken ();
291         s.outputSocket = getToken (REGULAR);
292         s.outputRedirect = CLIENT_REDIRECT;
293         break;
294
295       case REDIRECT_INPUT_SERVER:                                  /* @<s */
296         nextToken ();
297         s.inputSocket = getToken (REGULAR);
298         s.inputRedirect = SERVER_REDIRECT;
299         break;
300
301       case REDIRECT_INPUT_CLIENT:                                  /* @<c */
302         nextToken ();
303         s.inputSocket = getToken (REGULAR);
304         s.inputRedirect = CLIENT_REDIRECT;
305         break;
306
307       default:
308         done = TRUE;
309         break;
310     }
311   }
312   while (!done);
313
314   return (s);
315 }
316
317
318
319 /*************************************************************************/
320
321 tokenize ()
322
323 {
324   char* ptr = line;
325   char token [MAX_TOKEN_LENGTH];
326   char* tptr;
327
328   tIndex = 0;
329
330
331   while (TRUE)
332   {
333     tptr = token;
334     while (*ptr == ' ') ++ptr;
335     if (*ptr == NULL) break;
336
337     do
```

```
338     {
339       *tptr++ = *ptr++;
340     }
341     while (*ptr != ' ' && *ptr != NULL);
342
343     *tptr = NULL;
344     strcpy (tokens[tIndex++], token);
345   }
346
347
348   strcpy (tokens[tIndex++], "\n");
349   tokenCount = tIndex;
350   tIndex = 0;
351 }
352
353 /*************************************************************************/
354
355 char* nextToken ()
356
357 {
358   return (tokens[tIndex++]);
359 }
360
361 /*************************************************************************/
362
363 char *lastToken ()
364
365 {
366   return (tokens[tIndex - 1]);
367 }
368
369 /*************************************************************************/
370
371 peekCode ()
372
373 {
374
375   return (tokenCode (peekToken ()));
376 }
377
378 /*************************************************************************/
379
380 char* peekToken ()
381
382 {
383
384   return (tokens[tIndex]);
385 }
386
387 /*************************************************************************/
388
389 char *getToken (code)
390
391   int code;
392
393 {
394   char str [MAX_STRING_LENGTH];
395
396
397
398   if (peekCode () != code)
399   {
```

```
400     sprintf (str, "Expected %s\n", metacharacters[code]);
401     error (str);
402     return (NULL);
403   }
404   else
405     return (nextToken ());
406 }
407
408 /*************************************************************************/
409
410 tokenCode (token)
411
412   char* token;
413
414 {
415
416   return (findString (metacharacters, token));
417 }
418
419 /*************************************************************************/
420
421 findString (strs, str)
422
423   char* strs [];
424   char* str;
425
426 {
427   int i = 0;
428
429
430
431   while (strcmp (strs[i], "") != 0)
432     if (strcmp (strs[i], str) == 0)
433       return (i);
434     else
435       ++i;
436
437   return (NOT_FOUND);
438 }
439
440
441
442 /*************************************************************************/
443
444 executeSequence (p)
445
446   struct sequence* p;
447
448 {
449   int i, result;
450
451
452   if (p->background)
453   {
454     if (fork () == 0)
455     {
456       printf ("[%d]\n", getpid ());
457
458       signal (SIGQUIT, originalQuitHandler);
459       setpgrp (0, getpid ());
460       for (i = 0; i < p->count; i++)
461         executePipeline (&p->pipeline[i]);
```

```
462       exit ( 0);
463     }
464   }
465   else
466     for (i = 0; i < p->count; i++)
467       executePipeline (&p->pipeline[i]);
468 }
469
470 /*****************************************************************************/
471
472 executePipeline (p)
473
474   struct pipeline *p;
475
476 {
477   int pid, processGroup, result;
478
479
480   if (p->count == 1 && builtIn (p->simple[0].token[0]))
481     executeSimple (&p->simple[0]);
482   else
483   {
484     if ((pid = fork ()) == 0)
485     {
486
487       if (p->count == 1)
488         executeSimple (&p->simple[0]);
489       else
490         executePipes (p);
491       exit (  0);
492     }
493     else
494     {
495
496       waitForPID (pid);
497     }
498   }
499 }
500
501 /*****************************************************************************/
502
503 waitForPID (pid)
504
505   int pid;
506
507 {
508   int status;
509
510
511   while (wait (&status) != pid);
512 }
513
514 /*****************************************************************************/
515
516 executePipes (p)
517
518   struct pipeline *p;
519
520 {
521   int pipes, status, i;
522   int pipefd [MAX_PIPES][2];
523
```

```
524
525   pipes = p->count - 1;
526   for (i = 0; i < pipes; i++)
527     pipe (pipefd[i]);
528   for (i = 0; i < p->count; i++)
529   {
530     if (fork () != 0) continue;
531
532
533     if (i != 0) dup2 (pipefd[i-1][READ], STDIN);
534
535     if (i != p->count - 1) dup2 (pipefd[i][WRITE], STDOUT);
536
537     closeAllPipes (pipefd, pipes);
538
539     executeSimple (&p->simple[i]);
540     exit (0);
541   }
542
543
544   closeAllPipes (pipefd, pipes);
545   for (i = 0; i < p->count; i++)
546     wait (&status);
547 }
548
549 /*************************************************************************/
550
551 closeAllPipes (pipefd, pipes)
552
553   int pipefd [][2];
554   int pipes;
555
556 {
557   int i;
558
559
560   for (i = 0; i < pipes; i++)
561   {
562     close (pipefd[i][READ]);
563     close (pipefd[i][WRITE]);
564   }
565 }
566
567 /*************************************************************************/
568
569 executeSimple (p)
570
571   struct simple* p;
572
573 {
574   int copyStdin, copyStdout;
575
576
577   if (builtIn (p->token[0]))
578   {
579
580
581     copyStdin = dup (STDIN);
582     copyStdout = dup (STDOUT);
583     if (redirect (p)) executeBuiltIn (p);
584
585     dup2 (copyStdin, STDIN);
```

```
586     dup2 (copyStdout, STDOUT);
587     close (copyStdin);
588     close (copyStdout);
589   }
590   else if (redirect (p))
591     executePrimitive (p);
592 }
593
594 /*************************************************************************/
595
596 executePrimitive (p)
597
598   struct simple* p;
599
600 {
601
602   if (execvp (p->token[0], p->token) == -1)
603   {
604     perror ("ish");
605     exit ( 1);
606   }
607 }
608
609 /*************************************************************************/
610
611
612
613 builtInCode (token)
614
615   char* token;
616
617 {
618
619   return (findString (builtIns, token));
620 }
621
622 /*************************************************************************/
623
624 builtIn (token)
625
626   char* token;
627
628 {
629
630   return (builtInCode (token) != NOT_FOUND);
631 }
632
633 /*************************************************************************/
634
635 executeBuiltIn (p)
636
637   struct simple* p;
638
639 {
640
641   switch (builtInCode (p->token[0]))
642   {
643     case CD:
644       executeCd (p);
645       break;
646
647     case ECHO_BUILTIN:
```

```
648       executeEcho (p);
649       break;
650
651     case GETENV:
652       executeGetenv (p);
653       break;
654
655     case SETENV:
656       executeSetenv (p);
657       break;
658   }
659 }
660
661 /*****************************************************************************/
662
663 executeEcho (p)
664
665   struct simple* p;
666
667 {
668   int i;
669
670
671   for (i = 1; i < p->count; i++)
672     printf ("%s ", p->token[i]);
673
674   printf ("\n");
675 }
676
677 /*****************************************************************************/
678
679 executeGetenv (p)
680
681   struct simple* p;
682
683 {
684   char* value;
685
686
687   if (p->count != 2)
688   {
689     error ("Usage: getenv variable\n");
690     return;
691   }
692
693   value = getenv (p->token[1]);
694
695   if (value == NULL)
696     printf ("Environment variable is not currently set\n");
697   else
698     printf ("%s\n", value);
699 }
700
701 /*****************************************************************************/
702
703 executeSetenv (p)
704
705   struct simple* p;
706
707 {
708
709   if (p->count != 3)
```

```
710     error ("Usage: setenv variable value\n");
711   else
712     setenv (p->token[1], p->token[2]);
713 }
714
715 /*************************************************************************/
716
717 setenv (envName, newValue)
718
719   char* envName;
720   char* newValue;
721
722 {
723   int i = 0;
724   char newStr [MAX_STRING_LENGTH];
725   int len;
726
727
728   sprintf (newStr, "%s=%s", envName, newValue);
729   len = strlen (envName) + 1;
730
731   while (environ[i] != NULL)
732   {
733     if (strncmp (environ[i], newStr, len) == 0) break;
734     ++i ;
735   }
736
737   if (environ[i] == NULL) environ[i+1] = NULL;
738
739   environ[i] = (char*) malloc (strlen (newStr) + 1);
740   strcpy (environ[i], newStr);
741 }
742
743 /*************************************************************************/
744
745 executeCd (p)
746
747   struct simple* p;
748
749 {
750
751   if (p->count != 2)
752     error ("Usage: cd path\n");
753   else if (chdir (p->token[1]) == -1)
754     perror ("ish");
755 }
756
757 /*************************************************************************/
758
759
760
761 redirect (p)
762
763   struct simple *p;
764
765 {
766   int mask;
767
768
769   switch (p->inputRedirect)
770   {
771     case FILE_REDIRECT:
```

```
772       if (!dupFd (p->inputFile, O_RDONLY, STDIN)) return(FALSE);
773       break;
774
775     case SERVER_REDIRECT:
776       if (!server (p->inputSocket, INPUT_SOCKET)) return(FALSE);
777       break;
778
779     case CLIENT_REDIRECT:
780       if (!client (p->inputSocket, INPUT_SOCKET)) return(FALSE);
781       break;
782   }
783
784
785   switch (p->outputRedirect)
786   {
787     case FILE_REDIRECT:
788       mask = O_CREAT | O_WRONLY | (p->append?O_APPEND:O_TRUNC);
789       if (!dupFd (p->outputFile, mask, STDOUT)) return (FALSE);
790       break;
791
792     case SERVER_REDIRECT:
793       if (!server(p->outputSocket, OUTPUT_SOCKET)) return(FALSE);
794       break;
795
796     case CLIENT_REDIRECT:
797       if (!client(p->outputSocket, OUTPUT_SOCKET)) return(FALSE);
798       break;
799   }
800
801   return (TRUE);
802 }
803
804 /*************************************************************************/
805
806 dupFd (name, mask, stdFd)
807
808   char* name;
809   int mask, stdFd;
810
811 {
812   int fd;
813
814
815   fd = open (name, mask, DEFAULT_PERMISSION);
816
817   if (fd == -1)
818   {
819     error ("Cannot redirect\n");
820     return (FALSE);
821   }
822
823   dup2 (fd, stdFd);
824   close (fd);
825   return (TRUE);
826 }
827
828 /*************************************************************************/
829
830
831
832 internetAddress (name)
833
```

```
834   char* name;
835
836 {
837
838   return (strpbrk (name, "0123456789") != NULL);
839 }
840
841 /*************************************************************************/
842
843 socketRedirect (type)
844
845   int type;
846
847 {
848   return (type == SERVER_REDIRECT || type == CLIENT_REDIRECT);
849 }
850
851 /*************************************************************************/
852
853 getHostAndPort (str, name, port)
854
855   char *str, *name;
856   int* port;
857
858 {
859   char *tok1, *tok2;
860
861
862
863   tok1 = strtok (str, ".");
864   tok2 = strtok (NULL, ".");
865   if (tok2 == NULL)
866   {
867     strcpy (name, "");
868     sscanf (tok1, "%d", port);
869   }
870   else
871   {
872     strcpy (name, tok1);
873     sscanf (tok2, "%d", port);
874   }
875 }
876
877 /*************************************************************************/
878
879 client (name, type)
880
881   char* name;
882   int type;
883
884 {
885   int clientFd, result, internet, domain, serverLen, port;
886   char hostName [100];
887   struct sockaddr_un serverUNIXAddress;
888   struct sockaddr_in serverINETAddress;
889   struct sockaddr* serverSockAddrPtr;
890   struct hostent* hostStruct;
891   struct in_addr* hostNode;
892
893
894   internet = internetAddress (name);
895   domain = internet ? AF_INET : AF_UNIX;
```

```
896
897   clientFd = socket (domain, SOCK_STREAM, DEFAULT_PROTOCOL);
898
899   if (clientFd == -1)
900   {
901     perror ("ish");
902     return (FALSE);
903   }
904
905   if (internet)
906   {
907     getHostAndPort (name, hostName, &port);
908     if (hostName[0] == NULL) gethostname (hostName, 100);
909     serverINETAddress.sin_family = AF_INET;
910     hostStruct = gethostbyname (hostName);
911
912     if (hostStruct == NULL)
913     {
914       perror ("ish");
915       return (FALSE);
916     }
917
918     hostNode = (struct in_addr*) hostStruct->h_addr;
919     printf ("IP address = %s\n", inet_ntoa (*hostNode));
920     serverINETAddress.sin_addr = *hostNode;
921     serverINETAddress.sin_port = htons (port);
922     serverSockAddrPtr = (struct sockaddr*) &serverINETAddress;
923     serverLen = sizeof (serverINETAddress);
924   }
925   else
926   {
927     serverUNIXAddress.sun_family = AF_UNIX;
928     strcpy (serverUNIXAddress.sun_path, name);
929     serverSockAddrPtr = (struct sockaddr*) &serverUNIXAddress;
930     serverLen = sizeof (serverUNIXAddress);
931   }
932
933   do
934   {
935     result = connect (clientFd, serverSockAddrPtr, serverLen);
936     if(result == -1) sleep(SOCKET_SLEEP);
937   }
938   while (result == -1);
939
940
941   if (type == OUTPUT_SOCKET) dup2 (clientFd, STDOUT);
942   if (type == INPUT_SOCKET) dup2 (clientFd, STDIN);
943   close (clientFd);
944
945   return (TRUE);
946 }
947
948 /*****************************************************************************/
949
950 server (name, type)
951
952   char* name;
953   int type;
954
955 {
956   int serverFd, clientFd, serverLen, clientLen;
957   int domain, internet, port;
```

```
958   struct sockaddr_un serverUNIXAddress;
959   struct sockaddr_un clientUNIXAddress;
960   struct sockaddr_in serverINETAddress;
961   struct sockaddr_in clientINETAddress;
962   struct sockaddr* serverSockAddrPtr;
963   struct sockaddr* clientSockAddrPtr;
964
965
966   internet = internetAddress (name);
967   domain = internet ? AF_INET : AF_UNIX;
968
969   serverFd = socket (domain, SOCK_STREAM, DEFAULT_PROTOCOL);
970
971   if (serverFd == -1)
972   {
973     perror ("ish");
974     return (FALSE);
975   }
976
977   if (internet)
978   {
979     sscanf (name, "%d", &port);
980
981     serverLen = sizeof (serverINETAddress);
982     bzero ((char*) &serverINETAddress, serverLen);
983     serverINETAddress.sin_family = AF_INET;
984     serverINETAddress.sin_addr.s_addr = htonl (INADDR_ANY);
985     serverINETAddress.sin_port = htons (port);
986     serverSockAddrPtr = (struct sockaddr*) &serverINETAddress;
987   }
988   else
989   {
990     serverUNIXAddress.sun_family = AF_UNIX;
991     strcpy (serverUNIXAddress.sun_path,name);
992     serverSockAddrPtr = (struct sockaddr*) &serverUNIXAddress;
993     serverLen = sizeof (serverUNIXAddress);
994     unlink (name);
995   }
996
997
998   if (bind (serverFd, serverSockAddrPtr, serverLen) == -1)
999   {
1000    perror ("ish");
1001    return (FALSE);
1002  }
1003
1004
1005  if (listen (serverFd, DEFAULT_QUEUE_LENGTH) == -1)
1006  {
1007    perror ("ish");
1008    return (FALSE);
1009  }
1010
1011  if (internet)
1012  {
1013    clientLen = sizeof (clientINETAddress);
1014    clientSockAddrPtr = (struct sockaddr*) &clientINETAddress;
1015  }
1016  else
1017  {
1018    clientLen = sizeof (clientUNIXAddress);
1019    clientSockAddrPtr = (struct sockaddr*) &clientUNIXAddress;
```

```
1020   }
1021
1022
1023   clientFd = accept (serverFd, clientSockAddrPtr, &clientLen);
1024
1025   close (serverFd);
1026
1027   if (clientFd == -1)
1028   {
1029     perror ("ish");
1030     return (FALSE);
1031   }
1032
1033
1034   if (type == OUTPUT_SOCKET) dup2 (clientFd, STDOUT);
1035   if (type == INPUT_SOCKET) dup2 (clientFd, STDIN);
1036   close (clientFd);
1037
1038   return (TRUE);
1039 }
1040
1041 /*****************            FIN  DE  LISTADO            ****************/
```

Bibliografía

UNIX Version 4, Manual de Referencia Englewood Cliffs, *New Jersey 07632 Prentice Hall p. 361- 388*

UNIX Sistema V Version 4 Englewood Cliffs, *New Jersey 07632 Prentice Hall p. 493- 525, 532, 533, 535-542*

UNIX Administration Guide For System V Rebeca Thomas, PhD., Rick Farrow Englewood Cliffs, *New Jersey 07632. Prentice Hall*

UNIX for programmers and users. Graham Glass. *Prentice Hall*

El Entorno De Programación UNIX. Brian W. Kernighan, Rob Pike Englewood Cliffs, *new Jersey 07632 Prentice Hall*

UNIX Comunications . Bart Anderson, Bryan Costales, Harry Henderson Indianapolis, IN: Harward W. SAMS & COMPANY, Second Edition , 1991

TCP/IP and Related Protocols. Uyles Black. Mc Graw-Hill, Inc.

TCP/IP Network Administration. Craig Hunt. O'Reilly & Associates

Version 2.05 PC/TCP Command Reference. FTP Softwarwe Inc; Racal-Datacom InterLan Series. Wakefield. MA; *Release of October, 1990*

The Whole Internet Users's Guide & Catalog. De Krol. *O'Reilly &Associates, Inc.*

Using C on the UNIX System: A guide to system programming. David A. Curry *O'Reilly; Sebastopal, 1989*

Zen and the art of the Internet: A begunners guide. Brendan P. Kehoe *PTR Prentice Hall, 1993*

Open Computing Guide to Mosaic. Levi Reiss, Joseph Radin. *McGraw Hill, 1994*