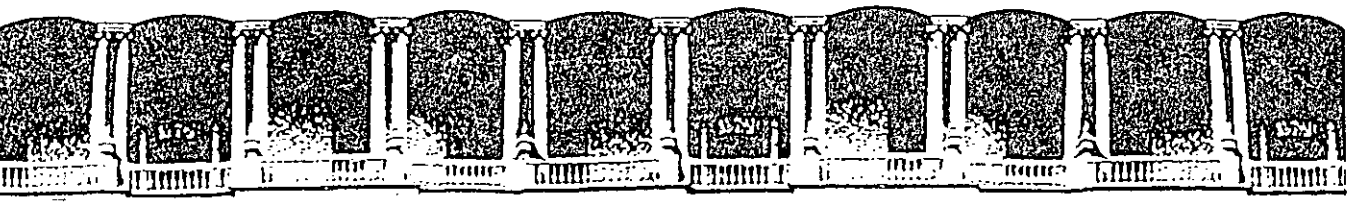


FACULTAD DE INGENIERIA U.N.A.M.
DIVISION DE EDUCACION CONTINUA

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA

Redes en ambiente UNIX

Aarón Arcos Tapia
Norberto Arrieta Márquez
J. Antonio Chávez Flores

Capítulo 1

El desarrollo de la Internet

En los últimos años la capacidad de interconexión de las computadoras ha crecido en forma notable debido a la necesidad de intercambiar información entre diferentes organismos. La tecnología en esta rama ha creado modelos que permiten la interconexión de computadoras en áreas geográficas pequeñas o muy grandes.

Las **redes de área local (Local Area Network, LAN)** proporcionan un medio de comunicación de alta velocidad en áreas geográficas reducidas. Este tipo de redes es muy popular en campus universitarios y empresas con instalaciones cercanas.

Existen también tecnologías bien establecidas para interconectar computadoras separadas geográficamente por miles de kilómetros, aunque claro, con una disminución notable con respecto a la velocidad de transmisión. Este tipo de redes se conocen como **redes de área amplia (Wide Area Network, WAN)**.

1.1 La necesidad de una red universal

Si bien es posible encontrar una tecnología para casi cualquier necesidad, lo ideal sería una que las cubriera todas, y en la cual un gran número de redes se pudieran interconectar para formar una sola entidad. El problema actual es que muchas redes de computadoras forman entidades independientes, que sólo satisfacen las necesidades de información de un grupo limitado. Por otro lado, una red universal tiene múltiples ventajas: las universidades de diferentes partes del mundo podrían intercambiar información con otras universidades o centros de investigación en forma directa e inmediata; la industria se beneficiaría también al tener acceso a las investigaciones más recientes en diversas áreas y en general cualquier otro tipo de organización (gubernamental, militar, servicios, etc.) podría obtener beneficios con este intercambio de información.

Las investigaciones más avanzadas en esta rama han sido realizadas en los E.U. y se han visto cristalizadas con la **Internet**. La Internet es una conexión de redes que abarca alrededor de 40 países y que permite a las computadoras que pertenecen a ella establecer una conexión universal.

Las investigaciones que propiciaron el desarrollo de la Internet se iniciaron en la **Defense Advanced Research Proyect Agency (Agencia de Proyectos Avanzados de Investigación de la Defensa, DARPA)** de los Estados Unidos a inicios de los setentas. La tecnología desarrollada por esta organización incluye un conjunto de estándares que definen los detalles para comunicar computadoras, así como para interconectar redes, los cuales se aplicaron en la construcción de la red militar ARPANET.

1.2 Historia y ambiente de la Internet

La DARPA empezó a trabajar en la Internet a mediados de los setentas utilizando una tecnología de intercambio de paquetes. Muchas de sus ideas tomaron forma en la red ARPANET, que rápidamente alcanzó una gran popularidad atrayendo el interés de varios centros de investigación. Los científicos que se interesaron en ARPANET programaron reuniones informales de investigadores para intercambiar ideas y discutir los resultados de sus experimentos. Para 1979, los esfuerzos de estas reuniones se encaminaban hacia la especificación de un protocolo estándar para la Internet, que posteriormente sería denominado **TCP/IP**.

DARPA formó un comité informal para coordinar y guiar el diseño de este protocolo, así como la arquitectura de la Internet. Este comité se denominó **Internet Control and Configuration Board (Consejo de Configuración y Control de Internet, ICCB)**. Uno de los factores que aceleró la aceptación de TCP/IP fue que en 1983 la **Office of the Secretary of Defense (Oficina de la Secretaría de Defensa, OSD)** de los Estados Unidos determinó que todas las computadoras conectadas a redes de área amplia usaran TCP/IP. Por ese entonces, la **Defense Comunication Agency (Agencia de Comunicación de la Defensa, DCA)** dividió a ARPANET en dos redes: una para investigación a futuro y otra para comunicaciones militares (ARPANET y MILNET).

Para fomentar el desarrollo de la Internet, se financió a dos grandes instituciones para la implementación de los protocolos (Bolt Beranek and Newman, Inc.) y el software de Unix que lo soportara (La Universidad de Berkeley). Con esto DARPA fue capaz de alcanzar alrededor del 90% de los departamentos de ciencias computacionales de las universidades estadounidenses.

Otro hecho que repercutió ampliamente en el desarrollo de la Internet, fue que la **National Science Foundation (Fundación Nacional para la Ciencia, NSF)** decidió participar activamente en la expansión de la Internet con el objeto de abarcar el mayor número posible de científicos.

Actualmente, ARPANET es la base de interconexión de los principales centros de investigación de los E.U., incluyendo universidades, corporaciones comerciales y laboratorios gubernamentales como: NSF, **Departament of Defense (Departamento de la Defensa, DOD)**, **Departament of Energy (Departamento de Energía, DOE)**, **Health and Human Services Agency (Agencia de Servicios Humanos y de Salud, HHS)** **National Aeronautics and Space Administration (Administración Nacional de Aeronáutica y del Espacio, NASA)**, **IBM, HP, DEC** y la **Universidad Nacional Autónoma de México (UNAM)** a partir de 1989.

La Internet ha demostrado la viabilidad de interconectar una amplia variedad de tecnologías de red. En los últimos siete años la Internet ha crecido de manera acelerada, y ahora abarca cientos de redes localizadas a lo largo de E.U., Europa y América Latina. Conecta a cerca de 3,000 redes y 313,000 computadoras que son utilizadas por alrededor de 550,000 personas.

El ritmo de crecimiento de la Internet sobrepasó rápidamente las expectativas iniciales, lo cual introdujo problemas no anticipados en el diseño original y motivó a los investigadores a encontrar nuevas técnicas para administrar recursos muy grandes y distribuidos.

La tecnología de TCP/IP y la red Internet han crecido más allá del proyecto inicial de investigación y se han convertido en un producto que ofrece facilidades de producción para miles de personas que dependen de la red diariamente.

1.3 La familia de protocolos TCP/IP

Hasta ahora sólo se ha descrito a TCP/IP como un protocolo para intercambio de información entre redes. En realidad TCP/IP consta de varios protocolos que tienen aplicaciones específicas. En el desarrollo de este trabajo el término TCP/IP abarcará a todo el conjunto de protocolos.

El propósito fundamental de TCP/IP es permitir el desarrollo de aplicaciones a alto nivel, ocultando las especificaciones inherentes a cualquier sistema de hardware. Esto hace innecesario que los programadores tengan que aprender los detalles de la con-

figuración del hardware y permite desarrollos de software más portables.

Los servicios más populares que ofrece la Internet a nivel de aplicación son:

- **Correo electrónico.-** Este permite al usuario enviar documentos pequeños como memoranda y cartas a diferentes individuos o grupos.

- **Transferencia de archivos.-** Permite el intercambio de archivos de cualquier longitud y tipo entre un par de nodos.

- **Sesión remota.-** Este servicio permite a un usuario conectarse a una máquina remota y establecer una sesión interactiva con la misma.

Por medio de estos servicios, los usuarios de la Internet realizan diariamente actividades tan diversas como intercambio de correo personal y de oficina, distribución de software y de correcciones al mismo, consultas a bases de datos públicas, uso de supercomputadoras remotas, etc.

Todas estas aplicación se basan en dos servicios básicos a nivel de red:

- **Servicio de entrega de paquetes sin conexión.-** Con este servicio, TCP/IP transfiere pequeños mensajes de una máquina a otra, aunque no se garantiza una entrega en orden, ni completa.

- **Servicio de transporte de flujo confiable.-** Este servicio permite establecer una conexión entre aplicaciones en diferentes computadoras, para el intercambio de datos a través de ella, como si fuera un enlace directo y permanente de hardware. Este servicio sí garantiza una entrega completa y en orden.

1.4 Información sobre Internet

La tecnología internet constituye lo que se conoce como un **sistema abierto**. Este hecho ha contribuido de manera notable a aumentar la popularidad de TCP/IP, ya que a diferencia de los sistemas de comunicación comerciales, sus especificaciones son públicas y cualquier persona puede obtenerlas sin costo alguno. De esta forma, es posible construir (e incluso obtener) el software necesario para comunicarse a través de una internet a costos muy

bajos. Algo más que se debe resaltar, es que toda esta tecnología ha sido diseñada para soportar una comunicación entre máquinas de diversas arquitecturas de hardware y para adaptarse a diversos sistemas operativos.

La documentación de protocolos, estándares y políticas no puede ser obtenida de ningún fabricante. La organización que se encarga de manejar los detalles administrativos de la Internet, además de distribuir la documentación sobre el trabajo hecho en ella es el **Network Information Center (Centro de Información de la Red, NIC)**. Las proposiciones para nuevos protocolos y revisiones de los mismos, así como los estándares establecidos actualmente para TCP/IP aparecen en una serie de reportes técnicos denominados **Internet Request for Comments (Petición de Comentario para la Internet, RFC)**. Algunos RFCs fueron también publicados inicialmente en series de reportes denominados **Internet Engineering Notes (Notas de Ingeniería de la Internet, IEN)**. Ambas series están numeradas en forma secuencial en orden cronológico en el que se escribieron. Los RFCs se pueden obtener por medio de correo, correo electrónico o directamente a través de la Internet usando un programa de transmisión de archivos (el apéndice A describe la manera de obtener los RFCs).

Capítulo 2

La arquitectura de la Internet

En este capítulo se presentará el modelo de una internet, la cual es una abstracción que permite reunir un conjunto de redes físicas interconectadas para formar una sola unidad. En esta abstracción, la conexión es independiente del hardware que soporta cada red internamente.

2.1 El concepto de red universal

El principal propósito de una interconexión de redes debe ser el permitir la comunicación entre **nodos** (*hosts*), sin importar la red en que estén localizados, en forma transparente para el usuario.

Un esquema de redes interconectadas, o internet, permite construir un sistema unificado de redes que soporta un **servicio de comunicación universal**. Aunque toda aplicación de red debe utilizar las primitivas de comunicación inherentes a su hardware, debe existir una capa de software intermedia entre dichas primitivas y el programa en sí, que permita ocultar los detalles de bajo nivel haciendo posible que las diferentes redes que componen una internet parezcan ser una sola red.

Una red con estas características proporciona las siguientes ventajas:

- El diseño oculta la arquitectura de la red al usuario, por lo que no se necesita conocer los detalles de hardware.
- La topología de la internet es muy flexible: para conectar una nueva red basta con establecer una conexión con algún punto de la internet; no es necesario conectarla a un punto central, ni hacer conexiones entre ella y el resto de las redes.
- Es posible intercambiar datos entre nodos no conectados

directamente utilizando redes intermedias.

- Todas las máquinas en la internet tienen un identificador universal (nombre o dirección) que las distingue en forma única.

- La interface de los programas de aplicación con el usuario es independiente del hardware de red.

2.2 La arquitectura de la Internet

La interconexión de las redes físicas que forman una internet se realiza mediante computadoras denominadas **gateways**. Estos dispositivos cuentan con conexiones directas a varias redes y se encargan de transferir paquetes de información entre las mismas.

En la figura 2.1 se muestra cómo un gateway se encarga de conectar dos redes. La función del gateway G consiste en capturar los paquetes destinados a nodos que no se encuentren en la misma red en la cual se originaron; una vez capturados, los paquetes serán retransmitidos en la red correcta.

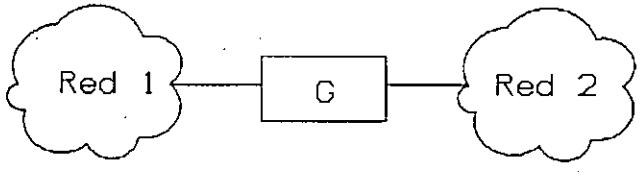


Fig. 2.1. Dos redes conectadas por un gateway

Para poder actuar como conexiones entre redes, los gateways deben también realizar funciones de **ruteo** (elegir el conjunto de redes intermedias para que un paquete llegue a su destino correcto). Para esto, cada gateway debe conocer la topología de la internet más allá de las redes a las que se conecta directamente. En la figura 2.2 se muestra cómo el gateway G1 debe transmitir hacia la red 2 todos los paquetes en la red 1 que estén destinados a las redes 2 y 3. La información de la topología de una internet se mantiene en tablas almacenadas en cada gateway.

El usuario de una internet ve la interconexión de redes como

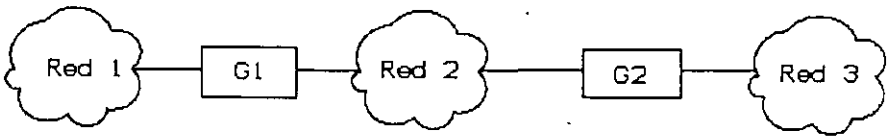


Fig. 2.2. Tres redes conectadas por dos gateways.

si se tratara de una sola red virtual que le permite establecer un canal de comunicación entre máquinas de diferentes redes, incluso si éstas tienen una tecnología de hardware diferente. La figura 2.3 ilustra este concepto.

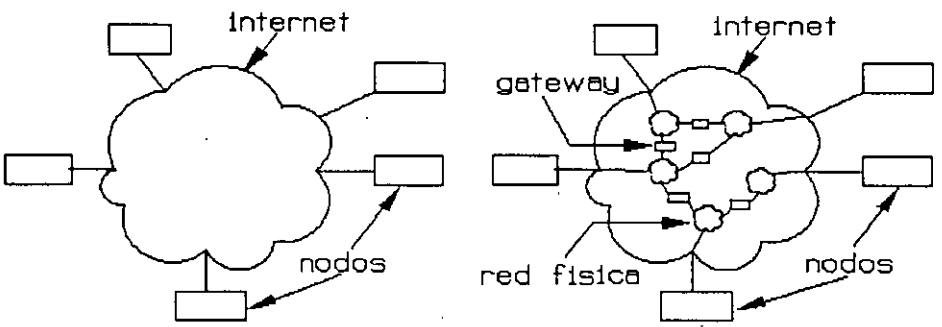


Fig. 2.3. Una internet es una abstracción de la interconexión de un conjunto de redes físicas.

Es importante tener presente que los protocolos TCP/IP tratan a todas las redes de la misma manera, sin importar que se trate de una red de área local, como Ethernet, o de una red de área amplia, como NSFNET.

Capítulo 3

Niveles de Protocolo

Los problemas a los que se debe enfrentar el diseñador de software en un ambiente de redes son muy variados; por ejemplo, el sensado de señales, la detección y corrección de errores de transmisión y la selección de rutas óptimas entre dos nodos, entre otros. Un sólo elemento de software qué se encargara de todas estas tareas sería demasiado complejo, por lo que es necesario buscar otro enfoque.

Para esto, se han desarrollado diversos modelos que se aplican en la construcción de redes. En este capítulo se presentarán dos de esos modelos: el desarrollado por la **International Organization for Standardization** (**Organización Internacional de Estándares, ISO**) y el utilizado por TCP/IP. La característica principal de estos modelos es que separan los problemas en grupos específicos, de tal modo que cada uno se pueda resolver con mayor facilidad.

Antes de explicar estos dos modelos, es necesario saber qué es un protocolo y por qué es tan importante hacer esta separación.

3.1 Principio de división en niveles

Un **protocolo** es un conjunto de reglas que definen las convenciones que deben seguir los elementos que desean establecer cierto tipo de comunicación. Es necesario notar que un protocolo no es un elemento de software, sino una especificación a la que se deben ajustar tales elementos. Existen protocolos de bajo nivel (que se encargan de detalles de hardware como el sensado de señales), de nivel medio (que se encargan de proporcionar servicios de red como el transporte de paquetes) y de alto nivel (que son utilizados directamente por el usuario, por ejemplo el correo electrónico).

Es difícil lograr que un solo protocolo pueda encargarse de aspectos tan diversos como los arriba señalados, así como de fallas de hardware, congestionamiento en la red, pérdida o retardo de paquetes, etc. Para poder manejar tantos problemas, se ha procedido a dividir los protocolos de red en módulos llamados **niveles** (layers), a los cuales se les asigna un problema específico. La figura 3.1 ilustra esta idea.

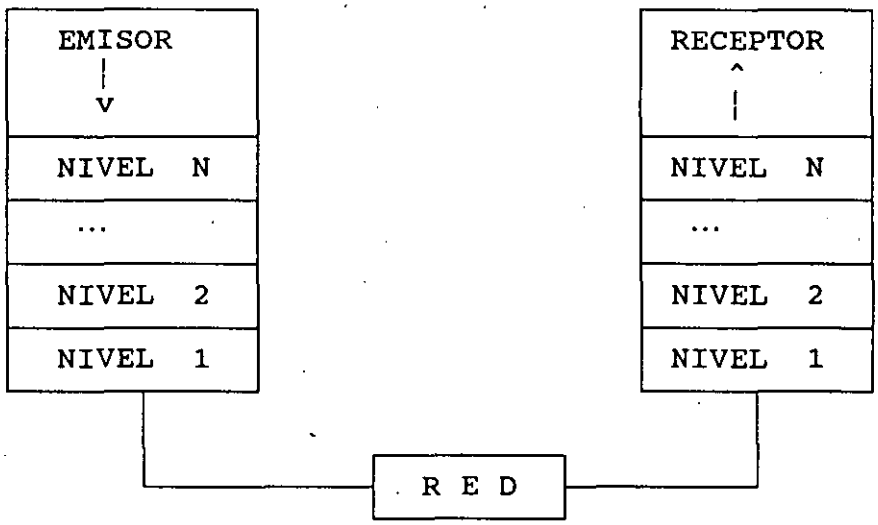


Fig. 3.1. Al transmitir un mensaje, éste debe pasar por cada uno de los niveles.

En una red que utilice el esquema de división en niveles, cualquier mensaje debe descender por cada uno de los niveles en el emisor, pasar a la red y ascender los mismos niveles en el receptor. Las funciones de cada nivel deben ser especificadas en uno o varios protocolos bien definidos.

3.2 El modelo ISO/OSI

El modelo **Open System Interconnect (Interconexión de Sistemas Abiertos, OSI)** fue desarrollado por la ISO en 1982, siendo uno de los primeros que utilizaron el principio de la división en niveles. El modelo OSI está dividido en siete niveles organizados conceptualmente de manera jerárquica, de tal forma que los protocolos de más alto nivel utilizan los servicios proporcionados por los niveles inferiores. Los niveles del modelo OSI son:

Físico Es el nivel más bajo y representa la base para los demás. Especifica los detalles de hardware para la conexión física de un nodo a la red, incluyendo características eléctricas de voltaje y corriente, códigos de transmisión y las formas de compartir el medio de comunicación.

Liga de Datos
Se encarga de la transferencia de datos entre los extremos de una conexión física. En este nivel se define el formato de los paquetes de bits (*frames*), incluye también el chequeo de errores y especifica el intercambio de los mensajes necesarios para determinar el éxito o fracaso de una transmisión.

Red Este nivel se encarga de definir la unidad básica de transferencia a través de la red e incluye los conceptos de direccionamiento y ruteo, así como los que involucran problemas de congestión en la red.

Transporte
Este cuarto nivel se encarga de asegurar que los paquetes de datos transmitidos lleguen a su destino sin errores, es decir proporciona un servicio de transmisión confiable.

Sesión El nivel de sesión se encarga del problema de acceso a terminales remotas (conexiones virtuales), así como de problemas de autentificación.

Presentación
El penúltimo nivel provee las conversiones de formato que requiere el nivel de aplicación. En este nivel es muy común utilizar el estándar **ASN.1** que proporciona una especificación formal para la representación de datos que se utilizan en los programas de aplicación.

Aplicación
Este nivel está formado por los programas de aplicación que utilizan a la red. Se incluyen, por ejemplo, el correo electrónico, la transferencia de archivos y las sesiones remotas.

Capítulo 4

EL modelo TCP/IP

El segundo modelo que se analizará fue desarrollado para utilizarse en la Internet TCP/IP. A diferencia del modelo OSI, éste no cuenta con un nivel físico, con el propósito de lograr la independencia de hardware; y está organizado en tres niveles. La figura 4.1 muestra la organización de TCP/IP.

Internet — El nivel internet se encarga de la comunicación de una máquina a otra. Maneja los mensajes de la red, checa su validez y realiza funciones de ruteo. También envía mensajes de control y error cuando se necesitan.

Transporte — Su principal función es establecer un canal de comunicación entre programas de aplicación. Este nivel debe regular el flujo de información, así como asegurar un medio de transporte confiable en el cual los datos lleguen sin errores y en secuencia.

Aplicación — En este nivel, el más alto, el usuario invoca programas de aplicación que accesan los servicios disponibles en una internet TCP/IP. Una aplicación interactúa con los protocolos del nivel de transporte para enviar o recibir datos.

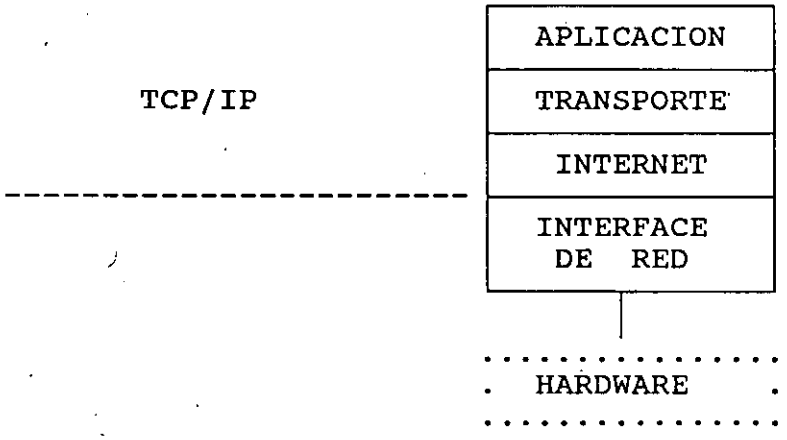


Fig. 4.1. Niveles Internet

En cualquiera de los dos modelos mencionados anteriormente, los protocolos que se encuentran al mismo nivel en diferentes nodos se comunican entre sí e intercambian paquetes del mismo tipo (los paquetes de información reciben nombres diferentes dependiendo del nivel en que se manejen, por ejemplo mensaje, paquete, datagrama, frame). En el caso de la internet esto se ilustra con la figura 4.2.

En la figura 4.2, el gateway G recibe el frame enviado por el nodo A, pero debe retransmitirlo a su destino verdadero, el nodo B. El frame entregado a G es exactamente el mismo que envió el nodo A, pero diferente del que G envía a B (p.e, la fuente y el destino son diferentes). Note que, en niveles altos, los mensajes son idénticos, mientras que en los niveles bajos los mensajes pueden ser diferentes si hay máquinas intermedias.

Un punto importante es que TCP/IP requiere de una participación activa de todos los nodos de la red, debido a que cada uno de ellos debe implementar funciones complejas como ruteo, detección y recuperación de errores, etc.

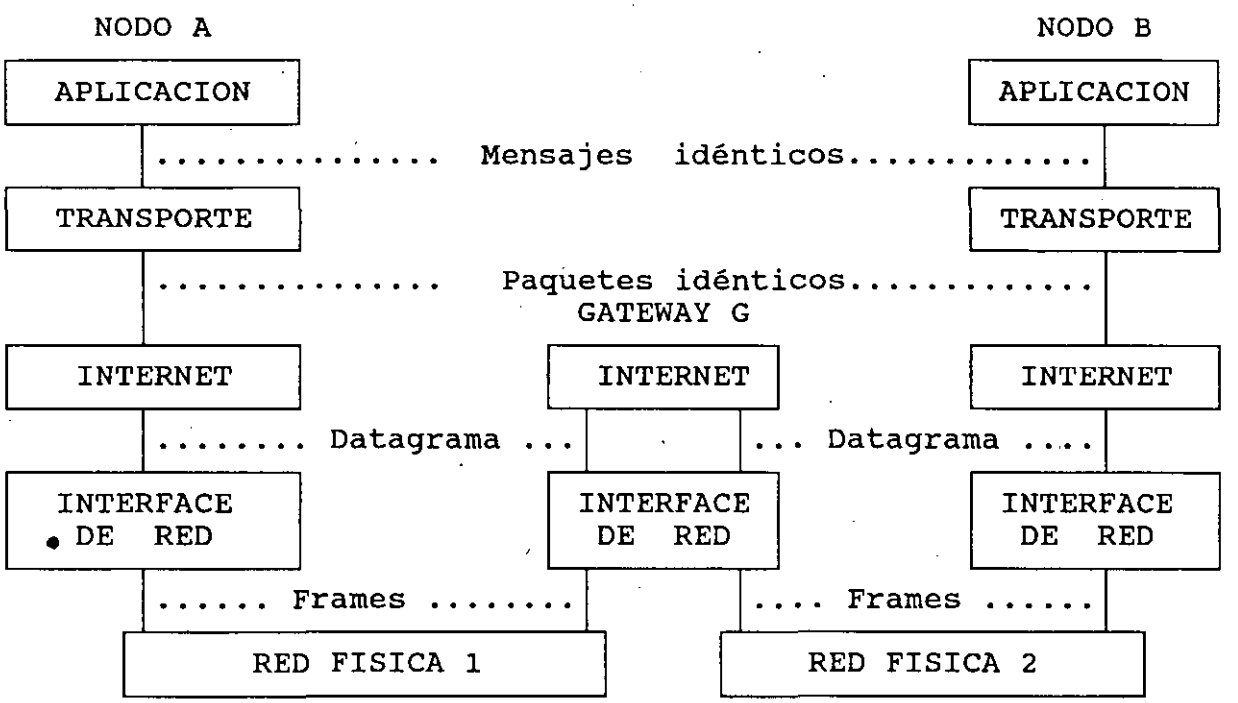


Fig. 4.2. Transferencia de unidades

Aunque las figuras anteriores muestran cada uno de los niveles de protocolo como una entidad individual, en realidad, en cada capa pueden existir varios protocolos. Cuando esto ocurre cada capa debe decidir a cual de los elementos de la capa siguiente debe entregar la información; a este proceso se le conoce como **multiplexaje** y se ilustra en la figura 4.3. En dicha figura, cuando un paquete es recibido por el protocolo P_A, éste debe escoger a cual de los siguientes protocolos P_1, P_2, ..., P_N deberá entregar dicho paquete.

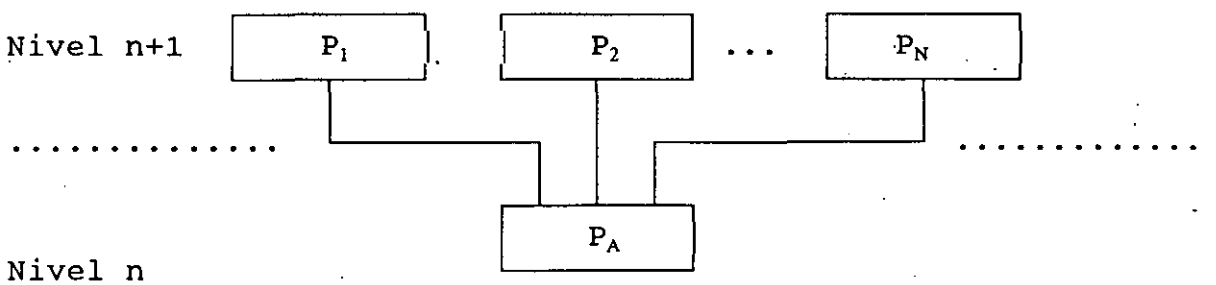


Fig. 4.3. Multiplexaje entre niveles.

En cualquier red se debe establecer un método que permita identifica de manera única a cada una de las computadoras conectadas a ella. Existe varias formas de llevar a cabo esta identificación, por ejemplo: nombres direcciones (números que representen la localización del nodo dentro l red) o rutas (la manera de llegar al nodo). Desde el punto de vista de usuario, es preferible usar un nombre para identificar a una máquina; si embargo, a nivel de software es necesario trabajar con una representació más compacta que, como las direcciones, se pueda manipular eficientemente Este capítulo analiza el direccionamiento empleado en una internet posteriormente se analizará el uso de nombres.

4.1 Esquema de direccionamiento Internet

Los protocolos TCP/IP han estandarizado un direccionamiento binari compacto, en el cual a cada nodo se le asigna una dirección única de 3 bits, llamada **dirección IP.**

Cada dirección IP consta de dos componentes (*idred, idnodo*), dond *idred* identifica una red particular e *idnodo* identifica un nodo dentro d ella. Existen varios tipos de direcciones IP, las cuales se muestran en l figura 4.4. Observe que los bits más significativos se utilizan par distinguir entre las diferentes clases de direcciones.

Las direcciones IP de clase A reservan 7 bits para el identificado de red y 24 para el de nodo; con ello se pueden tener hasta 256 diferente direcciones de red, cada una de ellas con hasta 16 millones de nodos, po lo que se utilizan en redes de gran tamaño. Actualmente, la mayoría de la direcciones de clase A ya han sido ocupadas para redes como la ARPANET, l NSFNET, etc.

Las direcciones IP de clase B utilizan 14 bits para el identificado de red y 16 para el de nodo, resultando útiles en redes medianas (de 25 a 65535 nodos). La RedUNAM utiliza una dirección de este tipo.

Por último, las direcciones IP de clase C destinan únicamente 21 bit para el identificador de red y 8 para el de nodo, por lo que son utilizada en redes pequeñas, que tienen menos de 256 nodos. Las **direcciones multicas** se explicarán posteriormente; las de clase E no se utilizan actualmente.

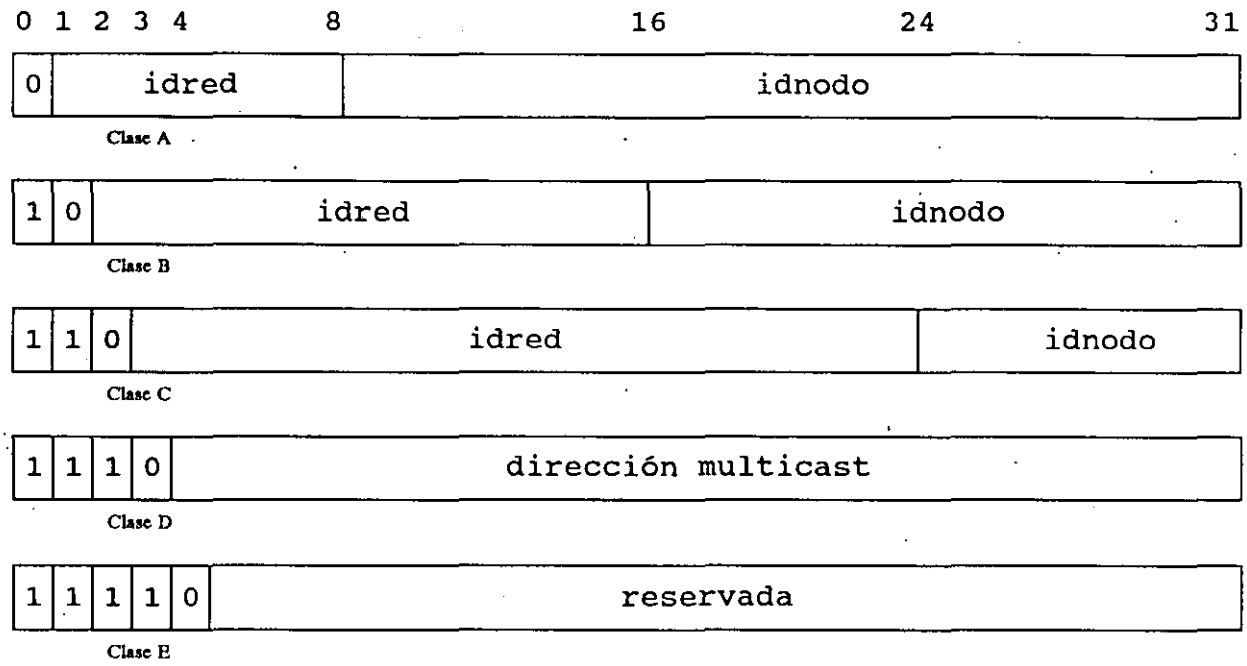


Fig. 4.4. Tipos de direcciones IP

Una dirección IP identifica la conexión física de un nodo a la red, de tal manera que un nodo no puede tener una dirección IP permanente. Esto significa que si un nodo se mueve de una red a otra su dirección IP deberá cambiar. Otra desventaja de este esquema es que cuando una red crece demasiado y necesita una dirección de otra clase (por ejemplo de clase C a clase B) es necesario cambiar las direcciones de todos los nodo de esa red.

Por último se mencionará que pueden existir nodos con más de una dirección IP, es decir con más de una conexión a la red (a dichos nodos se les da el nombre de **nodo multihomed**).

4.2 Representación decimal de direcciones IP

Aunque las direcciones IP son enteros de 32 bits, existe una notación especial que se utiliza para especificar estas direcciones. En esta representación, llamada **notación decimal**, una dirección se denota con cuatro enteros decimales separados por puntos, en donde cada entero representa el valor de un byte de la dirección IP. Por ejemplo, una dirección como la siguiente:

10000100 11110100 000011110 00000010

se representa como:

132.248.54.2

Existen ciertos valores para las direcciones IP que se utilizan con propósitos especiales. Si uno de los identificadores tiene como valor cero, este se refiere al mismo objeto que emite el mensaje (red o nodo). Por ejemplo, para el nodo 132.248.54.1, la dirección 132.248.0.0 se refiere a él mismo; mientras que la dirección 0.0.54.2 se refiere al nodo 54.2 dentro de la misma red.

Existe otra dirección especial, llamada **dirección de broadcast** que se utiliza para hacer referencia a todos los nodos dentro de una red; esta dirección se especifica encendiendo todos los bits del identificador de nodo. Por ejemplo, la dirección 132.248.255.255 se refiere a todos los nodos de la red 132.248.

Las direcciones cuyo primer byte es 127 son conocidas como *dirección de loopback*. Cuando se utiliza esta dirección, el software de protocolo regresa los datos al nodo emisor sin transferirlos a la red, lo cual puede ser utilizado para realizar pruebas o intercomunicar procesos dentro de la misma máquina. Las direcciones de loopback más comunes son 127.0.0.0 y 127.0.0.1.

4.3 Asignación de direcciones IP

Para asegurar que las direcciones de una red sean únicas, éstas son asignadas por una autoridad central llamada **Network Information Center (Centro de Información de la Red, NIC)**. Dicha autoridad se encarga únicamente de asignar las direcciones de red y delega la responsabilidad de asignar las direcciones de los nodos al organismo encargado de la red en cuestión. Así por ejemplo, el NIC asignó a la RedUNAM la dirección 132.248.0.0 y es la Universidad quién se encarga de asignar las direcciones a cada nodo de su red.

Protocolos TCP/IP

Como se mencionó en los capítulos anteriores, una internet es una abstracción de un conjunto de redes físicas, que permite al usuario establecer una conexión entre cualquier par de nodos. En este capítulo analizaremos el protocolo básico que permite construir esa red virtual.

5.1 Protocolo Internet(IP)

Los servicios básicos que ofrece la familia de protocolos TCP/IP pueden agruparse en tres categorías:

- Servicios de aplicación
- Servicio de transporte confiable
- Servicio de entrega de paquetes sin conexión

Es este último (llamado técnicamente **servicio de entrega de paquetes no confiable de mejor esfuerzo sin conexión**) el que proporciona la base sobre la que se sustentan los demás servicios. El protocolo que proporciona este servicio se denomina *Internet Protocol* (**Protocolo Internet, IP**) y se encarga, entre otras cosas, de definir la unidad mínima de transferencia en una internet (**datagrama**), de realizar las funciones de ruteo y de definir las reglas de entrega de paquetes.

Transmisión de datagramas

Toda la información que se transmite por una internet viaja en pequeños paquetes denominados datagramas; los mensajes pueden ser divididos en uno o varios datagramas con el propósito de evitar que algún nodo se apropie demasiado tiempo del medio.

Antes de describir los elementos que componen un datagrama, es necesario señalar primero varios aspectos importantes concernientes

al transporte de los datos y su relación con el hardware utilizado.

En la práctica surgen varias limitaciones impuestas por la red física por la que se transmiten los datagramas. Una restricción importante es el tamaño de los paquetes que se pueden enviar a través de la red. El transporte de datos se hace más eficiente cuando cada datagrama puede ser enviado en un paquete físico diferente (**frame**); a este proceso se le denomina **encapsulado.** En el caso ideal, tendríamos que un datagrama se ajusta completamente a un frame físico; sin embargo, el tratar de fijar un tamaño de datagrama que se ajuste al tamaño de frame de las diferentes tecnologías es bastante complejo, ya que el límite en el tamaño máximo de un frame (conocido como **MTU**, Maximum Transfer Unit - **Unidad de Transferencia Máxima**) es muy variado; por ejemplo, Ethernet establece una MTU de 1500 bytes, mientras que Pronet-10 la establece de 2044 bytes.

IP utiliza inicialmente como tamaño de datagrama el MTU de la red a la que se conecta el emisor; si el datagrama debe pasar por una red con una MTU más pequeña, IP lo divide en piezas más pequeñas denominadas **fragmentos** (el proceso de división es llamado **fragmentación**), de tal modo que cada una de ellas se ajuste a un frame. Es importante recalcar que cada fragmento tiene el mismo formato que el datagrama original.

Cuando algún fragmento de un datagrama llega a su nodo destino, el nodo establece un límite de tiempo para que el resto de los fragmentos lleguen a él; si este límite se sobrepasa, se desecha el datagrama completo. Es claro que la fragmentación se realiza en la mayoría de los casos en los gateways. El ensamblado de los fragmentos, no obstante, no se realiza en los gateways intermedios, sino que es una tarea que se lleva a cabo en el destino final del datagrama, de manera que pueda obtener una copia exacta del original.

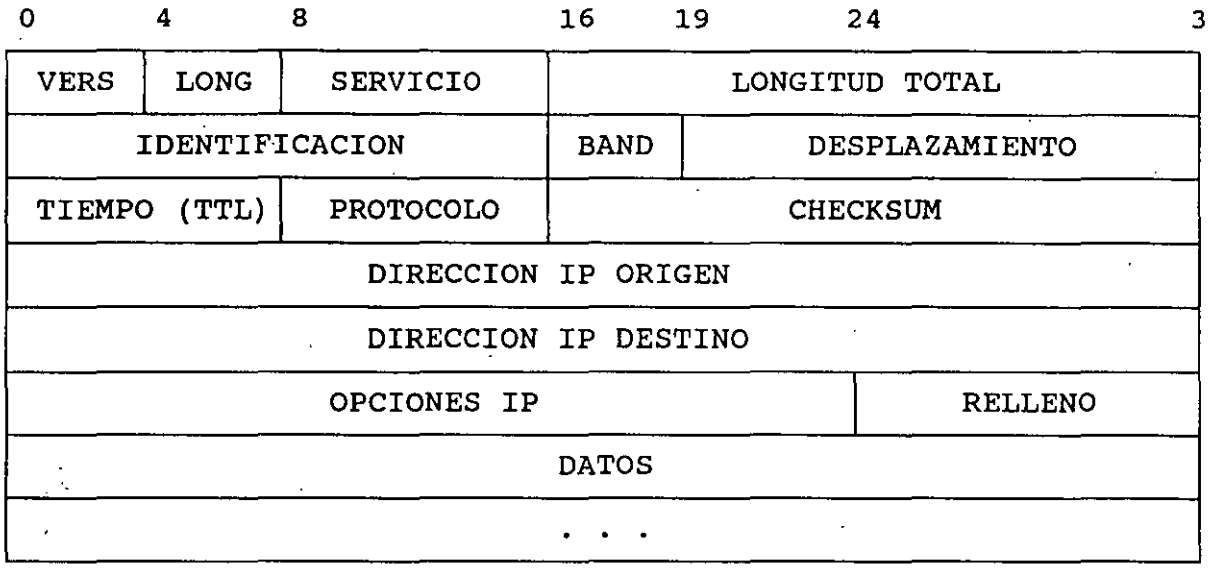


Fig. 5.1. Datagrama internet.

5.2 Protocolo de control de mensajes (ICMP)

Como se mencionó en el capítulo anterior, IP proporciona un servicio de envío de paquetes no confiable en el que pueden existir errores al momento de enviar los datos. Ejemplos típicos de estos errores pueden ser la duplicación de paquetes, la pérdida de los mismos o errores en los datos transmitidos. Aunque este tipo de errores pueden ignorarse, existen otros que son críticos en la operación de la red. La familia de protocolos TCP/IP cuenta con un mecanismo conocido como **Internet Control Message Protocol** (**ICMP**, Protocolo de Mensajes de Control Internet) que permite manejar algunos de estos errores.

ICMP permite a un gateway enviar errores o mensajes de control a otros gateways o nodos con el objeto de controlar el flujo de información en la Internet.

Técnicamente hablando, ICMP es un mecanismo reportador de errores; es decir, ICMP sólo notifica las condiciones de error a la fuente original del mensaje que causó dichas condiciones, pero no las corrige. Cuando un nodo recibe un mensaje ICMP, debe relacionar los errores reportados con alguno de los programas de aplicación que envió un datagrama y tomar las acciones necesarias para corregir el problema.

Aunque la mayoría de las condiciones por las que surge un mensaje ICMP se presentan en los emisores de los datos, en algunas ocasiones los problemas sólo pueden ser resueltos por los gateways intermedios; desafortunadamente ICMP sólo se limita a avisar a la fuente original. La razón de esta restricción es simple: en la mayoría de los casos es imposible conocer la ruta que siguió un paquete (a menos que se especifique la opción **grabar ruta** en el datagrama).

Interacción de ICMP con IP

Parecería que ICMP e IP son protocolos independientes, sin embargo ICMP es una parte fundamental de IP, de hecho el estándar obliga a que cualquier implementación del primero incluya al segundo.

Los mensajes ICMP viajan en datagramas IP, por lo que requiere de dos niveles de encapsulado, como lo muestra la figura 5.2. Cada mensaje de control viaja a través de la Internet en la porción de datos de un datagrama, el cual a su vez viaja en un *frame*.

Los datagramas que transportan mensajes ICMP tienen la misma prioridad y confiabilidad que cualquier otro datagrama; por esta razón los mismos mensajes de error pueden perderse. Sin embargo, nunca se generan paquetes ICMP por errores en mensajes de control. La razón de esta decisión se aprecia claramente si tomamos en cuenta que en una red congestionada cualquier mensaje adicional incrementaría el problema.

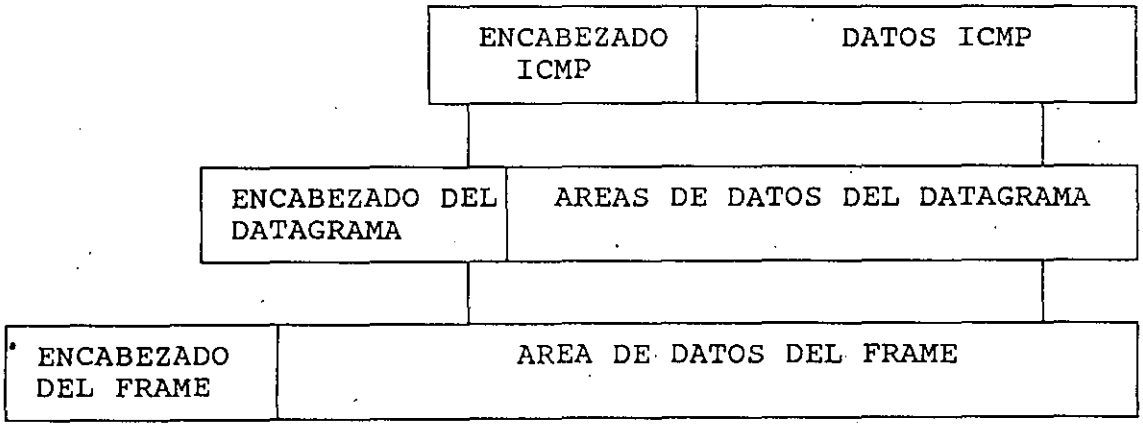


Fig. 5.2 Los dos niveles de encapsulado de ICMP.

Formato de los mensajes ICMP

Aunque existen diferentes tipos de mensajes ICMP, cada uno con su propio formato, los tres primeros campos son iguales en todos ellos: un campo de 8 bits para identificar el TIPO de mensaje, un campo de 8 bits denominado CODIGO, que proporciona mayor información acerca del tipo de mensaje y un campo de 16 bits que es el CHECKSUM. Se debe señalar también que los mensajes ICMP que reportan errores incluyen siempre el encabezado y los primeros 64 bits de datos del datagrama que causó el problema; esto permitirá al receptor determinar qué protocolo y qué aplicación fueron las causantes del problema. Como veremos más adelante, los protocolos de alto nivel de TCP/IP se diseñaron para incluir en los primeros 64 bits la mayor parte de la información de control.

El campo TIPO define el significado del mensaje así como su formato, como se muestra en la tabla 5.1.

Campo TIPO	Tipo de mensaje ICMP
0	Respuesta al Eco
3	Destino no alcanzable
4	Disminuir flujo
5	Redireccionamiento (cambiar una ruta)
8	Petición de Eco
11	Tiempo excedido para el datagrama
12	Problema de parámetro en un datagrama
13	Petición de estampado de tiempo
14	Respuesta de estampado de tiempo
15	Petición de información (obsoleto)
16	Respuesta de información (obsoleto)
17	Petición de máscara de dirección
18	Respuesta de máscara de dirección

Tabla 5.1 Tipos de mensaje ICMP.

5.3 Protocolo de Datagrama de Usuario (UDP)

Debido a que las direcciones IP distinguen a las computadoras conectadas a la red y no a los programas de aplicación que utilizan el protocolo, IP resulta insuficiente para la mayoría de las aplicaciones en las que puede haber destinos múltiples en el mismo nodo. A continuación analizaremos un protocolo que permite resolver este problema.

Aunque a primera vista se podría pensar que el destino final de un mensaje debe ser un proceso particular en una máquina

específica, el hacerlo así implicaría deficiencias muy graves. Por una parte, los procesos se crean y se destruyen en forma dinámica, por lo que el emisor rara vez puede identificar un proceso en otra máquina con el cual pueda comunicarse. Por otro lado, debe ser factible reemplazar los procesos que reciben datagramas sin informar al emisor (por ejemplo, al reinicializar una máquina generalmente las identificaciones de los procesos cambian). Por último, es necesario identificar a los destinos basándose en las funciones que implementan, no en los procesos que implementan esas funciones.

El **User Datagram Protocol (Protocolo de Datagrama de Usuario, UDP)** representa una solución al problema que se plantea. En él, en vez de considerar un proceso como el destino final, se considera que cada máquina contiene un conjunto de puntos abstractos, llamados **puertos**, que se utilizan para el intercambio de información. Cada puerto se identifica con un entero positivo. El sistema operativo de cada nodo debe proporcionar una interface que permita a los procesos accesar un puerto específico (por ejemplo, en Unix BSD esta interface es denominada **socket**). De esta manera, para comunicarse con un puerto externo, el emisor debe conocer tanto la dirección IP de la máquina destino como el número de puerto destino dentro del receptor. El uso de los puertos por las aplicaciones se trata en un capítulo posterior.

Mensajes UDP

UDP utiliza el protocolo IP para transportar un mensaje de una máquina a otra (los mensajes UDP se encapsulan en datagramas IP), por lo que provee el mismo mecanismo no confiable de entrega de datagramas sin conexión que IP. Los programas de aplicación que usan UDP deben aceptar la responsabilidad total de manejar los problemas de confiabilidad tales como pérdida de mensajes, duplicación, retardo y entrega fuera de orden.

Un mensaje de UDP es llamado **datagrama de usuario** y está formado de un encabezado y un área de datos. La figura 5.3 muestra el formato de un mensaje UDP. El encabezado del datagrama se divide en cuatro campos de 16 bits: los puertos origen y destino del mensaje (PUERTO ORIGEN y PUERTO DESTINO), la longitud en octetos del datagrama de usuario (LONGITUD DE DATAGRAMA) y un campo CHECKSUM.

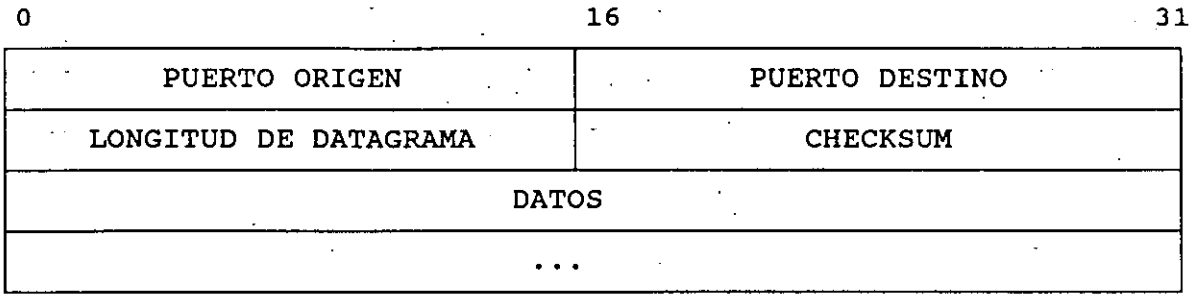


Fig. 5.3. Datagrama UDP.

5.4 Protocolo TCP

El servicio fundamental en una internet es el de entrega de paquetes no confiable. Sin embargo, las aplicaciones que utilizan este servicio para transferir grandes volúmenes de información requieren de un complejo sistema de detección y corrección de errores.

No obstante, es difícil diseñar software que proporcione confiabilidad para cada programa de aplicación. Precisamente, uno de los objetivos de la división de protocolos en niveles es encontrar una solución a este problema asegurando un sistema confiable de flujo de datos. Una alternativa para lograrlo es diseñar un protocolo de propósito general que pueda ser utilizado por cualquier programa de aplicación a través de una interface general para el servicio de flujo de datos.

Servicio de flujo de datos

La interface antes mencionada debe cumplir con las siguientes características:

- **Orientación a flujo de datos.-** Cuando existe una transferencia de datos entre dos procesos, dichos datos constituyen un flujo de octetos en el cual no existe estructura alguna. El sistema de entrega debe llevar a su destino el conjunto de bytes en la misma secuencia que el transmisor los entrega.

- **Conexión de circuito virtual.-** El software de protocolo debe modular la comunicación entre máquinas a través de peticiones de transferencia de datos entre los diferentes procesos involucrados, además de interactuar entre los sistemas

operativos e informar las peticiones de transferencia de datos. Una vez que la transferencia ha sido autorizada y que ambas máquinas están listas, el protocolo debe informar a los programas de aplicación que la transferencia puede comenzar. Durante la transferencia, el software de protocolo debe verificar que los datos sean recibidos correctamente y, si la comunicación falla por alguna razón, debe detectar el error y dar aviso a los programas de aplicación.

- **Buffer de transferencia.**- El software de protocolo puede dividir el flujo de datos en partes, independientemente de las piezas que el programa de aplicación transfiera, con el propósito de hacer la transferencia más eficiente y minimizar el tráfico en la red. Usualmente se colectan bytes de un flujo de datos en un buffer antes de transmitirlos por la red; si el programa de aplicación genera grandes bloques de datos, el protocolo de software puede dividirlos en piezas más pequeñas para su transmisión.

- **Conexión full duplex.**- Cuando se establece una conexión full duplex la comunicación es bidireccional y se puede dar simultáneamente.

Protocolo de acknowledgement positivo

Como ya se ha mencionado, el servicio de flujo de datos garantiza que éstos lleguen a su destino sin error y no duplicados; pero ¿cómo puede el software de protocolo garantizarlo si el sistema de comunicación establece que la entrega de paquetes es no confiable?. Para lograrlo, el protocolo se basa en una derivación de la técnica conocida como **acknowledgement positivo con retransmisión.** Este se puede resumir como sigue: siempre que una máquina recibe datos correctos, debe enviar a la fuente de esos datos un mensaje denominado acknowledgement el cual indica el éxito de la transmisión. Por esta razón, cada vez que el emisor envía datos, debe esperar el mensaje de acknowledgement del emisor; si el mensaje no llega en un determinado tiempo, el paquete se retransmite.

La figura 5.4 muestra como se lleva a cabo una transferencia de datos cuando se utiliza un protocolo de acknowledgement positivo.

Cuando un proceso manda un paquete de datos espera un tiempo antes de hacer una retransmisión, si el tiempo se agota se considera que el paquete fue mal transmitido y se procede a retransmitirlo. Sin embargo, la retransmisión puede causar

problemas debido a que tanto los paquetes como los mensajes de acknowledgement pueden llegar duplicados. Para resolver el problema, los protocolos asignan a cada paquete un número secuencial, de tal forma que el receptor puede asociar cada mensaje de acknowledgement con los paquetes que ha enviado.

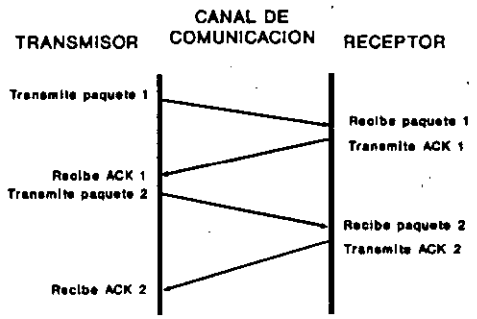


Fig. 5.4 Protocolo de acknowledgement positivo.

Protocolos de ventanas deslizantes

La técnica de acknowledgement positivo tiene una desventaja, y es que durante el tiempo que una máquina espera el mensaje de respuesta debe permanecer ociosa. A continuación se examinará una técnica que nos permite transmitir múltiples paquetes antes de esperar un acknowledgement. Dicho método es conocido como ventanas deslizantes (**sliding windows**).

Para entender el método de ventanas deslizantes pensemos en una secuencia de paquetes que serán transmitidos, como lo muestra la figura 5.5. El protocolo simula una ventana sobre el flujo de datos; inicialmente se envían todos los paquetes que están dentro de la ventana, cuando se recibe el acknowledgement del primer paquete la ventana se desliza hacia la derecha y se manda el paquete siguiente. El proceso se repite cada vez que se recibe el acknowledgement del primer paquete de la ventana. Se observa que los paquetes que se encuentran dentro de la ventana ya se enviaron, pero no se ha recibido su mensaje de acknowledgement correspondiente; mientras que, los que se encuentran a su derecha aún no han sido enviados y los que se encuentran a su izquierda ya han sido enviados y reconocidos. El tamaño de la ventana y la velocidad a la que se transmiten los paquetes determina la eficiencia del protocolo.

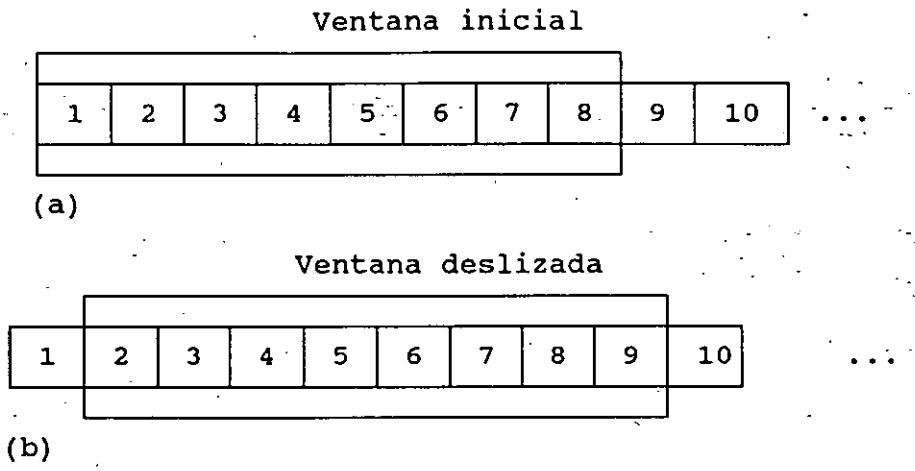


Fig. 5.5. (a) Ventana de tamaño 8 con 10 paquetes por enviar
(b) La ventana se desliza hacia el paquete 9 una
vez que se recibe el acknowledgement de 1.

TCP utiliza un mecanismo especial de ventanas deslizantes, que ayuda a mejorar la eficiencia de la transmisión y a controlar el flujo de información. Este mecanismo opera a nivel de bytes y no a nivel de paquetes. Los bytes del flujo de datos son numerados secuencialmente y se mantienen tres apuntadores sobre la secuencia para definir la ventana deslizante, como lo muestra la figura 5.6.

El apuntador de la derecha separa los bytes que ya han sido enviados de los que aún no lo han sido; los bytes del extremo derecho serán enviados una vez que se mueva la ventana. El apuntador intermedio define la frontera entre los bytes que ya han sido enviados, pero no se ha recibido su mensaje de acknowledgement, y los que se enviarán antes de que se reciba algún acknowledgement. Por último, el apuntador izquierdo separa los bytes que ya han sido enviados, y de los cuales ya se ha recibido el mensaje de acknowledgement, de aquellos que se han enviado sin que aún no se reciba su acknowledgement correspondiente.

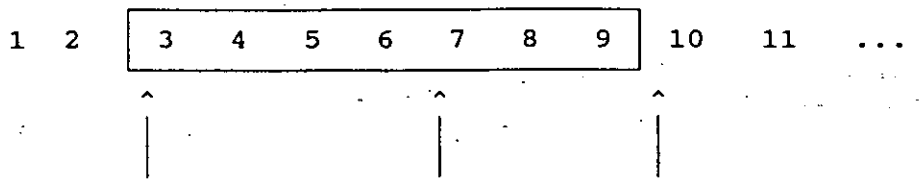


Fig. 5.6. Ejemplo de ventana deslizante en TCP.

TCP permite que el tamaño de la ventana deslizante varíe. Cada acknowledgement especifica cuantos bytes se recibieron, a la vez que contiene un mensaje que indica cuántos bytes se le pueden enviar al receptor (**window advertisement**); de esta forma, el emisor incrementa o decrementa el tamaño de su ventana.

La ventaja de utilizar un tamaño de ventana variable es que se tiene un mecanismo de control de flujo que hace que la transferencia de datos sea más eficiente. Cuando el buffer del receptor se comienza a llenar puede mandar un mensaje de window advertisement para hacer más pequeña la ventana de emisor; incluso puede ser de cero para detener la transmisión de datos. Una vez que el buffer comienza a liberar datos, debe mandar un mensaje para que incremente el tamaño de la ventana del emisor.

Transmisión de datos confiable

Después de conocer el principio del protocolo de ventanas deslizantes, examinaremos el servicio de entrega de flujo de datos que proporciona TCP/IP. La capa que ofrece este servicio se denomina **Transmition Control Protocol** (Protocolo de Control de Transmisión, TCP).

TCP es más complejo que los protocolos que se han visto en los capítulos anteriores y dentro de sus funciones se encuentran especificar el formato de los datos y los mensajes de acknowledgement que se transmiten cuando dos máquinas establecen un canal de comunicación, así como los procedimientos necesarios para que estos lleguen correctamente a su destino. Es muy importante recalcar que TCP, al igual que IP y UDP, no especifica los detalles entre la interface y el programa de aplicación, ni la manera en que los programas de aplicación invocan las operaciones de protocolo. Esto permite implementaciones flexibles que hacen posible que para una sola especificación del protocolo se pueda construir software para una gran variedad de máquinas.

En el esquema de capas de protocolos, TCP se encuentra por encima de IP. La figura 5.7 muestra la organización conceptual de

este esquema.

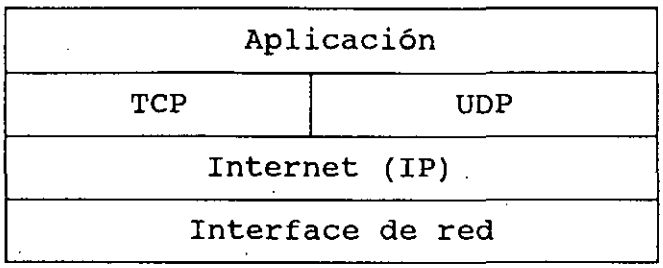


Fig. 5.7 Esquema conceptual de capas.

Dos conceptos importantes necesarios para comprender el funcionamiento de TCP son el de **conexión** y el de **circuito virtual**. Una conexión está definida por dos puntos finales que se utilizan en el intercambio de información; en el caso de TCP, un punto final consiste de un par de números (**N, P**) donde N es la dirección IP de una máquina y P es un puerto TCP en dicho nodo. Por ejemplo, el punto final (132.248.54.2, 161) indica el puerto TCP 161 en la máquina con dirección IP 132.248.54.2. Un puerto TCP en el mismo nodo puede ser compartido por múltiples conexiones, por lo que, se pueden establecer una comunicaciones concurrentes en algún puerto.

Segmentos TCP

En TCP, un flujo de datos es una colección de bytes divididos en **segmentos**. La unidad de transferencia en TCP es el segmento que normalmente forma uno o varios datagramas IP. La figura 5.8 muestra el formato de un segmento TCP.

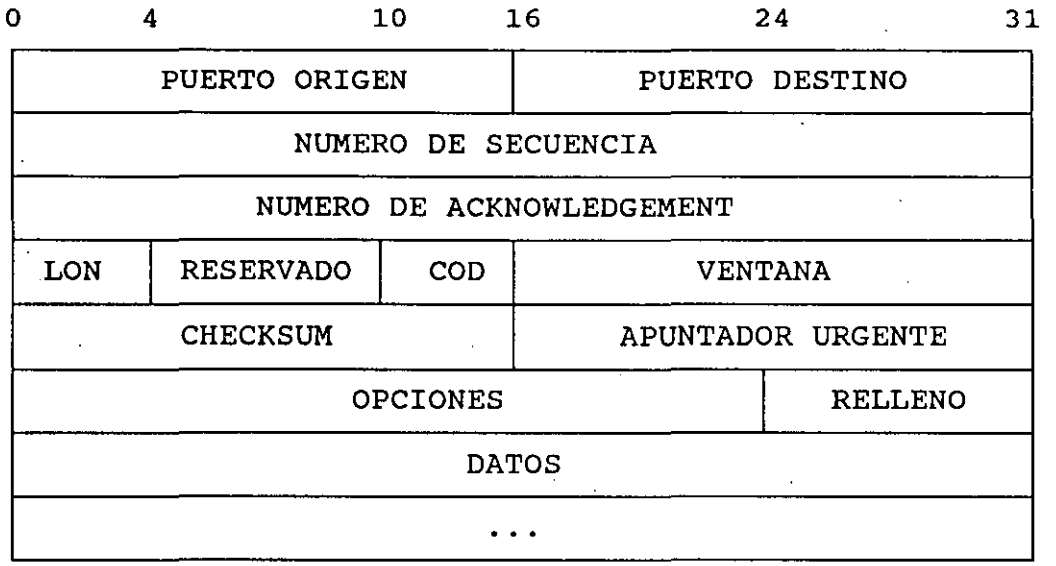


Fig. 5.8 Formato de un segmento TCP.

Capítulo 6

El modelo cliente-servidor

La mayor parte de los servicios que ofrece TCP/IP están basados en un esquema denominado **cliente-servidor**. En este esquema el servicio se implementa con dos procesos denominados precisamente **cliente** y **servidor**. Este capítulo tiene como propósito analizar este modelo.

6.1 Definiciones

Antes de analizar a detalle el modelo cliente-servidor es necesario definir los elementos que lo conforman:

Servidor.- Este término se utiliza para cualquier programa que ofrece un servicio, por ejemplo transferencia de archivos, sesión remota, etc. Un servidor debe encontrarse en un estado pasivo en el que acepta peticiones; al ocurrir esto, ofrece sus servicios y retorna una respuesta. El servidor puede llevar a cabo tareas sencillas o complicadas que pueden ser realizadas por uno o varios procesos. Generalmente los servicios se implementan como programas de aplicación que permiten el acceso a recursos como discos, impresoras, procesadores, bases de datos, ventanas, etc.

Cliente.- Este es un proceso activo que envía una petición al servidor y una vez que éste se activa, espera su respuesta. Generalmente los clientes son las aplicaciones que proporcionan la interfase con el usuario en un ambiente de red (una excepción notable es el protocolo X).

6.2 Ejemplo del modelo cliente-servidor: talk

Estos conceptos se pueden comprender de mejor manera con un ejemplo sencillo, como lo es el servicio de envío de mensajes vía pantalla que ofrece Unix a través del comando *talk*. Este comando permite entablar una conversación entre dos terminales conectadas a la red. A continuación se desglosarán los elementos con los cuales se implementa el servicio *talk*:

El servidor arranca cuando el sistema se inicializa, conectándose al puerto UDP 517, después de lo cual entra en un estado de espera hasta que recibe una petición[1]. Esta petición especifica el usuario con el cual se quiere establecer la conexión; una vez que ésta se establece el servidor únicamente se encarga de copiar los datos que recibe hacia la terminal del usuario y de leer los datos que el usuario proporciona a través del teclado, para enviárselos al cliente.

El cliente arranca cuando algún usuario invoca el comando:

```
% talk usuario_remoto@maquina_remota
```

La primera tarea que realiza es obtener de la línea de comandos la máquina remota con la cual se desea establecer la comunicación. A continuación utiliza un puerto arbitrario para hacer la conexión con el puerto 517 de dicha máquina y enviarle una petición. Una vez que el servidor acepta la petición, se limita a enviar los datos que proporciona el usuario hacia el servidor y de copiar los datos que recibe de éste a la terminal del usuario.

En los siguientes capítulos se presentarán otros ejemplos del modelo cliente-servidor.

6.3 Estructura de un cliente y un servidor

El ejemplo anterior es un caso típico de la interacción cliente-servidor; de hecho la mayoría de los servicios que se implementan utilizando este modelo son muy similares. La figura 6.1 muestra a manera de diagrama de flujo una de las posibles estructuras (la más simple) de estos procesos. En un capítulo

[1]En realidad es el proceso *inetd* quien realiza estas funciones. Es este proceso, quien arranca a *talkd* cuando llega la petición.

posterior se analiza a mayor detalle el procedimiento que aquí se describe:

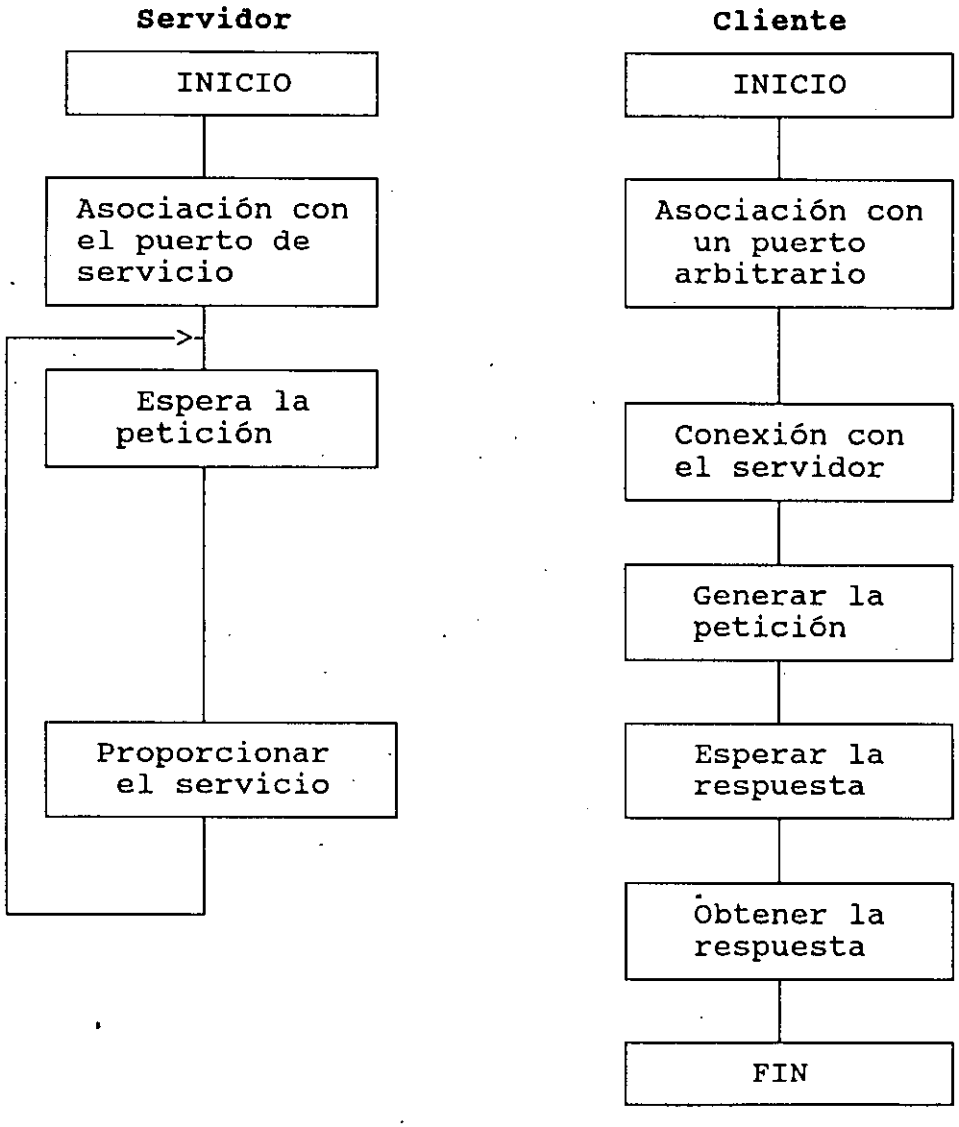


Fig. 6.1. Esquema de un cliente-servidor.

Es importante resaltar varios aspectos respecto al modelo cliente-servidor:

Mientras que un servidor inicia antes de que se realice alguna petición, el cliente lo hace hasta que el usuario lo invoca. Aunque el cliente puede terminar una vez que la petición ha sido satisfecha, el servidor debe permanecer en espera de nuevas peticiones. Note que en general el servidor permanece corriendo mientras la máquina está encendida, a diferencia del cliente que sólo corre mientras se proporciona el servicio.

b) La designación de puertos.- Mientras que un cliente puede utilizar un puerto arbitrario para la comunicación, el servidor debe utilizar un puerto específico reservado para el servicio que proporciona.

6.4 Servidores concurrentes

En el esquema considerado anteriormente se presentó un **servidor secuencial**; éste tiene la desventaja de que sólo puede atender una petición a la vez de manera que si llega un mayor número de peticiones que las que puede almacenar, las desecha. Existe un esquema alternativo en donde cada petición que llega es atendida de manera inmediata; en éste esquema existe un proceso maestro que es responsable de aceptar las peticiones y de crear un conjunto de procesos esclavos que son los que realmente atienden a las peticiones.

El proceso maestro ejecuta cinco acciones:

1. Abre el puerto de servicio.
2. Espera la petición de un cliente.
3. Asigna un nuevo puerto local para atender la petición y conecta este puerto con el cliente.
4. Inicializa un esclavo para que se encargue de esta petición.
5. El maestro regresa al punto 2 y continua aceptando nuevas peticiones mientras el esclavo se encarga de atender la petición en curso.

Además de realizar estas funciones, un servidor debe ser responsable de que las políticas de seguridad y acceso al sistema se cumplan, así como de protegerse contra peticiones mal hechas o peticiones que puedan provocar un mal funcionamiento.

Capítulo 7

Sistema de Dominio de Nombres

Como ya se ha explicado anteriormente, el esquema de direccionamiento IP utiliza enteros de 32 bits para identificar a los diferentes nodos de una internet; sin embargo, para la mayoría de los usuarios de aplicaciones interactivas resulta difícil manejar este tipo de direcciones, por lo que se hace necesario un esquema que permita utilizar identificadores de más alto nivel. Estos identificadores, o **nombres**, son cadenas de caracteres mnemónicas que sirven para referirse de manera única a cada nodo.

7.1 Identificadores

Debe resultar claro que, aunque a nivel aplicación el usuario especifica generalmente una cadena de caracteres para identificar un nodo, en última instancia los protocolos deben utilizar direcciones IP, de manera que debe existir un esquema de mapeo para ligar dicho nombre con la dirección del nodo que se quiere representar.

La idea de utilizar nombres para identificar a los nodos de la Internet surgió desde los orígenes de ésta. No obstante, el esquema utilizado ha sufrido varias modificaciones a lo largo del tiempo. Originalmente, los nombres utilizados no seguían formato alguno: el NIC (Internet Network Information Center) era el organismo encargado de asignar nombres adecuados. Con el paso del tiempo esta tarea llego a ser imposible debido al impresionante ritmo de crecimiento de la Internet, por lo que fue necesario implementar otro esquema.

Es claro que se necesita un sistema de mapeo distribuido, puesto que un sistema centralizado tiene varias desventajas: requiere de una máquina con una capacidad de proceso enorme para realizar el mapeo de todas las máquinas de la Internet; en caso de que este sistema falle, el uso de nombres se vuelve imposible y, finalmente, el tráfico hacia el sistema centralizado resultaría

excesivo.

7.2 Sistema de Dominio de Nombres

El sistema actual se basa en un esquema jerárquico de nombres llamado **Domain Name System (Sistema de Dominio de Nombres, DNS)**. Este sistema contempla dos aspectos: un aspecto abstracto que especifica la sintaxis y reglas para la asignación de nombres, y un aspecto concreto que especifica la implementación del mapeo de nombres a direcciones IP.

En este sistema, cada nombre se divide en varios campos que son asignados por organismos diferentes. Una autoridad central (el NIC) determina el contenido del primer campo; los campos restantes son asignados por organismos locales que obtienen autorización del NIC para hacerlo así. Por ejemplo, un nombre puede tener la forma:

nombre_local.dominio_primario

en donde *dominio_primario* es el nombre que determina la autoridad central y *nombre_local* es el nombre controlado por el organismo local. Resta decir que el nombre local puede tener a su vez varios campos determinados por diferentes autoridades. Por ejemplo, en

atl.cecafi.unam.mx

unam.mx se determina centralmente, *cecafi* es asignado por una autoridad a nivel de la Universidad (Dirección General de Servicios de Cómputo Académico) y *atl* es el nombre dado a nivel local (Centro de Cálculo de la Facultad de Ingeniería). Note que en el DNS un nombre está formado por etiquetas separadas por puntos. Todas las máquinas que son administradas por el mismo organismo, y por lo tanto comparten el mismo sufijo en sus nombres, forman grupos llamados **dominios** (en el ejemplo anterior, atl pertenece al dominio cecafi.unam.mx).

Es importante resaltar que, aunque el valor de esas etiquetas puede ser arbitrario, el hecho de utilizar un sistema jerárquico garantiza que cada nombre sea único en todo el dominio de nombres.

Como ya se mencionó, el campo de más alta jerarquía es asignado centralmente; los valores oficiales propuestos por el NIC para este campo, son los que se muestran en la tabla 7.1.

Nombre	Significado
COM	Organizaciones Comerciales
EDU	Instituciones Educativas
GOV	Instituciones Gubernamentales
MIL	Grupos Militares
NET	Centros de soporte de redes
ORG	Organizaciones de soporte de redes
ARPA	Dominio Temporal de Arpanet (obsoleto)
INT	Organizaciones Internacionales
País	Código de país (dos letras, p.e MX para México).

Tabla 7.1 Dominios primarios

Algunos ejemplos de nombres válidos son los siguientes:

dec.com	Digital Equipment Corporation
purdue.edu	Universidad de Purdue
unam.mx	Universidad Nacional Autónoma de México

La figura 7.1 muestra una pequeña parte de la jerarquía de nombres de la Internet.

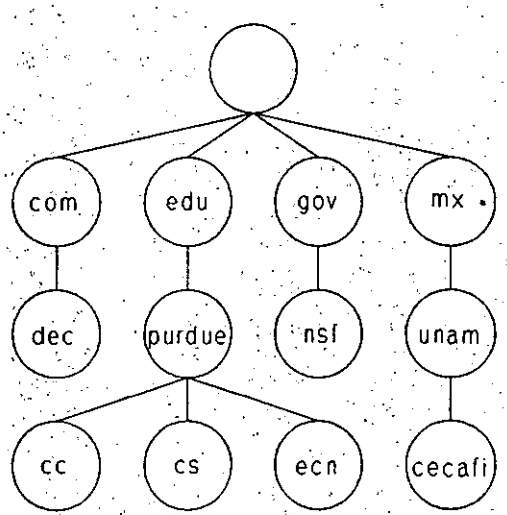


Fig. 7.1. Aspecto parcial del sistema jerárquico de nombres.

7.3 Tipos de nombres

El Sistema de Dominio de Nombres no sólo se puede utilizar para nombrar nodos en la red, sino que es lo suficientemente general para permitir diferentes jerarquías en el mismo sistema. De esta manera, es posible utilizarlo para identificar tanto nombres de nodos como documentos, estándares y otras entidades. Por esta razón se hace necesario especificar un **tipo** para diferenciar entre varios objetos con el mismo nombre. Por ejemplo, cuando una aplicación de correo electrónico necesita mapear un nombre, debe especificar que ese nombre corresponde a un *mail exchanger*.

Es importante notar que la sintaxis de un nombre no es suficiente para determinar el tipo del objeto a que se refiere ese nombre, por ejemplo, se puede tener un nodo llamado

kelem.cecafi.unam.mx

mientras que

comunicaciones.cecafi.unam.mx

se puede referir a un dominio, aunque los dos tengan el mismo número de elementos.

7.4 Mapeo de nombres a direcciones

El esquema de dominio de nombres incluye un sistema distribuido de traslación de nombres a direcciones. Este sistema está formado por un conjunto de elementos cooperativos, llamados **servidores de nombres**, que se encargan de mapear nombres de máquinas a direcciones IP (a este mapeo se le conoce como **resolución de nombres**).

Los servidores de nombres se organizan en forma jerárquica, tal como se muestra en la figura 16.2. El **servidor raíz** contiene información de la raíz del sistema y de los dominios del primer nivel (dec, nfs, purdue, unam); mientras que cada organización en este nivel debe tener un servidor que conozca los nombre de todas las máquinas de esa organización.

Cuando una máquina necesita mapear un nombre, debe contactar al servidor de su dominio; si éste no puede resolver el nombre por alguna razón, puede consultar a un servidor de un nivel más alto, llegando incluso a consultar al servidor raíz.

Existen dos tipos de consultas: **iterativas** y **recursivas.** En la primera, si el servidor no puede responder a la consulta regresa la dirección IP de un servidor que si puede hacerlo. En una consulta recursiva, el servidor se encarga de contactar a otros servidores hasta resolver el nombre.

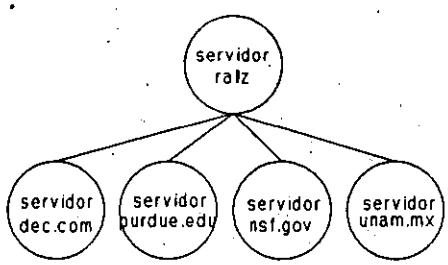


Fig. 16.2. Esquema jerárquico de servidores.

Observe que la resolución de nombres comienza de los niveles inferiores hacia los superiores, de tal forma que la mayoría de las consultas pueden resolverse a nivel local, dado que se refieren a máquinas locales. De esta manera se logra un mapeo eficiente.

Aún con este esquema, el mapeo de direcciones no locales podría tener problemas de eficiencia. Para evitarlo, los servidores utilizan una memoria cache en la que almacenan los resultados de las consultas más frecuentes con el propósito de anticipar las necesidades de los clientes y evitar consultas innecesarias.

Se dice que un servidor es **autoridad** para el dominio al que pertenece. Las respuestas que proporciona un servidor para máquinas que se encuentran en su dominio son llamadas **no autorizadas.**

Cuando se escribe el nombre de alguna máquina sin especificar todo su prefijo, éste se busca en el dominio local. Lo único que se debe hacer es concatenar al nombre una serie de prefijos locales y seguir el procedimiento de consulta normal. No obstante, no es recomendable generar aplicaciones que dependan de este esquema, ya que entonces dichas aplicaciones quedarían restringidas a un sólo dominio. Este mecanismo únicamente se ofrece a los usuarios para hacer más sencilla la interacción con las posibles aplicaciones.

Capítulo 8

Sockets

Aunque a lo largo de este trabajo se han descrito algunos de los detalles de los protocolos TCP/IP, aún no se ha descrito la manera en que un programador puede hacer uso de ellos. La razón de esto es que dichos protocolos no incluyen la descripción de ningún tipo de interface para el programador o el sistema operativo.

Los **sockets** se han convertido en la interface de programación de redes más popular en el ambiente Unix BSD[1]. En este capítulo se describen las **llamadas al sistema** y **funciones de biblioteca** para manejo de sockets de uso común en el ambiente Unix; sin embargo, el lector debe tener presente que la implementación disponible en su sistema puede variar en pequeños detalles, por lo que es recomendable que consulte los manuales de su sistema.

8.1 El concepto de socket

Un socket es una abstracción que representa un punto final de una conexión entre nodos y que se puede utilizar para enviar o recibir información. El diseño de la interface de sockets en Unix fue hecho teniendo en mente la naturaleza evolutiva de la tecnología de redes, por lo que la portabilidad fue uno de los objetivos de diseño principales. Por medio de los sockets es posible accesar el medio físico de una red directamente o bien utilizar alguna familia de protocolos diferente a TCP/IP. Bajo Unix, esta interface debe ser utilizada con el lenguaje de

[1] Aunque en Unix System V también cuenta con una interface de sockets, la interface nativa de este ambiente esta dada por los **streams**.

programación C.

Un programador puede crear sockets, fijarlos a una dirección específica, conectarlos con otros sockets y enviar o recibir información a través de ellos. En este capítulo no se describe la interface completa, aunque las funciones que se presentan son suficientes para escribir programas de mediana complejidad. Las siguientes secciones describen el funcionamiento de las llamadas al sistema más importantes.

8.2 Creación de sockets

La llamada al sistema:

```
int socket(int familia, int servicio, int protocolo);
```

permite crear un socket nuevo. Como ya se mencionó antes, los sockets se pueden usar con diferentes familias de protocolos, por lo que el primer parámetro, *familia*, especifica el conjunto de protocolos que se desea utilizar. Algunos valores posibles para este parámetro son:

AF_INET	Conjunto de protocolos TCP/IP
AF_UNIX	Dominio de Unix (comunicaciones locales)
AF_DECnet	Red DECnet (Digital)
AF_SNA	SNA de IBM

Otros valores se pueden consultar en el archivo de encabezados **sys/socket.h**[2]. En este trabajo únicamente nos interesa el primero de ellos (AF_INET).

El parámetro *servicio* determina el tipo de servicio de transporte que se desea; por supuesto, ese servicio debe ser proporcionado por algún protocolo que se encuentre en la familia especificada (cada familia de protocolos puede brindar servicios diferentes). Los valores posibles para la familia AF_INET son:

SOCK_STREAM	Transporte de un flujo de bytes confiable con conexión (TCP).
SOCK_DGRAM	Servicio de entrega de paquetes no confiable sin conexión (UDP).
SOCK_RAW	Acceso directo a los protocolos de comunicación.

[2]Bajo Unix, los archivos de encabezados para el lenguaje C se encuentran en el directorio /usr/include

Por último, el parámetro *protocolo* determina, en el caso en que varios protocolos dentro de la misma familia presten el servicio especificado, cuál de ellos utilizará el socket. Un valor nulo (0) hará que la función seleccione un valor adecuado.

El número entero que regresa la función es llamado **descriptor de socket** y es a través de él que se manipulará el nuevo socket.

8.3 Direcciones de sockets

Una vez que se ha creado un socket puede ser necesario especificar una dirección para él; por ejemplo, en protocolos como UDP y TCP esta dirección consiste de un par formado por una dirección IP y un número de puerto. Aunque no siempre es necesario especificar una dirección, habrá ocasiones en las cuales un programa debe "oír" en un puerto específico (recuerde el modelo cliente-servidor).

La función **bind** se utiliza para especificar la dirección de un socket:

```
int bind(int sock, struct sockaddr *dir, int longitud);
```

El parámetro *sock* es un descriptor obtenido con la función *socket*. *Dir* es un apuntador hacia una estructura en la que se almacena la dirección del socket; aunque el tipo de este parámetro es *sockaddr*, para la familia AF_INET existe un tipo equivalente llamado *sockaddr_in*. A continuación se muestran las definiciones de ambas estructuras (la definición de *sockaddr_in* se encuentra en el archivo **netinet/in.h**):

```
struct sockaddr {
    unsigned short sa_family;     /* Familia de protocolos */
    char           sa_data[14];   /* Dirección */
};
struct sockaddr_in {
    short          sin_family;   /* Familia de protocolos */
    short          sin_port;     /* Puerto */
    struct in_addr sin_addr;     /* Dirección IP */
    char           sin_zero[8];  /* Relleno de ceros */
};
```

El parámetro *longitud* determina el número de bytes de la dirección del socket. La función regresa un entero positivo o cero si tiene éxito, o -1 si se produce un error.

Si el programador no especifica la dirección de un socket, el sistema operativo le asigna una de manera automática cuando se necesita.

8.4 Conexión de sockets.

Algunos protocolos como TCP requieren que antes de usar un socket éste se conecte con un punto final remoto. Por ejemplo, un cliente del servicio de impresión debe conectarse al puerto 515 del nodo que presta dicho servicio. La función **connect** se utiliza para iniciar dicha conexión.

```
int connect(int sock, struct sockaddr *dir, int longitud);
```

Los parámetros *sock*, *dir* y *longitud* especifican el descriptor del socket local, un apuntador hacia la dirección del socket remoto con el que se hará la conexión y la longitud en bytes de esa dirección, respectivamente.

Para protocolos como UDP que prestan servicios sin conexión, no es necesario llamar a *connect*; sin embargo, si se hace la llamada sólo se almacenará localmente la dirección para usarla en otras funciones. **Connect** regresa un entero positivo si no hay errores y -1 si los hay.

Mientras los procesos clientes tienen que usar **connect** para iniciar una conexión, los procesos servidores deben utilizar la llamada **accept** para reconocer dicha conexión. Esta llamada al sistema tiene la forma:

```
int accept(int sock, struct sockaddr *dir, int *longitud);
```

Esta función permite aceptar las conexiones pedidas al socket cuyo descriptor es *sock*, el cual debe ser un socket amarrado a una dirección especifica mediante la función **bind**. La función crea un nuevo socket conectado con el nodo remoto que inició la conexión (con **connect**) y regresa su descriptor como valor. Al terminar la función, los parámetros *dir* y *longitud* apuntan a la dirección del nodo remoto y a la longitud de dicha dirección.

Un servidor puede recibir varias peticiones de conexión simultáneamente; sin embargo, la función **accept** solo puede procesarlas secuencialmente, por lo que deben ser almacenadas en algún lugar antes de ser atendidas. La función:

```
int listen(int sock, int peticiones);
```

permite especificar el número de *peticiones* que puede almacenar temporalmente el socket *sock*. En la mayoría de las implementaciones dicho parámetro debe ser menor o igual a ocho.

8.5 Envío de datos a través de un socket

Una vez que un proceso ha creado un socket, puede hacer uso de varias llamadas al sistema para enviar datos a través de él. Hay cinco diferentes formas de realizar esta función; aquí sólo se presentan las tres más comunes.

La llamada al sistema:

```
int write(int sock, char *buffer, int n);
```

permite enviar *n* bytes, localizados a partir de la dirección de memoria apuntada por *buffer*, al socket *sock*. **Write** regresa como valor el número de bytes transmitidos (el cual debe ser igual a su tercer parámetro) ó -1 en caso de error.

Por otro lado,

```
int send(int sock, char *buffer, int n, int banderas);
```

realiza la misma función que **write**, pero recibe un parámetro extra, *banderas*, que permite modificar la forma de transmitir los datos. El valor de este último parámetro puede ser MSG_OOB (para enviar datos fuera de banda), MSG_DONTROUTE (para no utilizar las facilidades de ruteo) ó la conjunción de las dos mediante el operador lógico o (|).

Por último,

```
int sendto(int sock, char *buffer, int n, int banderas,
        struct sockaddr *dir, int longitud);
```

permite además especificar la dirección del destino del mensaje. **Sendto** puede ser utilizada en sockets no conectados (que no han usado la función **connect**). El parámetro *dir* especifica la dirección remota, mientras que *longitud* especifica la longitud en bytes de esa dirección.

8.6 Recepción de datos a través de un socket

La interface de sockets provee también de cinco llamadas al

sistema para recibir los datos enviados con las llamadas descritas anteriormente. Al igual que en la sección anterior, sólo se describen las tres llamadas más usadas.

Los datos enviados con **write** pueden ser recibidos con la llamada:

```
int read(int sock, char *buffer, int n);
```

que lee *n* bytes del socket *sock* y lo coloca a partir de la dirección de memoria apuntada por *buffer*. Al igual que **write**, **read** regresa como valor el número de bytes transmitidos (al llegar al fin de la transmisión regresa 0).

La llamada:

```
int recv(int sock, char *buffer, int n, int banderas);
```

recibe los datos transmitidos con **send**. El parámetro *banderas* puede tener los valores MSG_OOB (para recibir datos fuera de banda), MSG_PEEK (para analizar los datos recibidos sin leerlos) ó la combinación de los dos.

Por último,

```
int recvfrom(int sock, char *buffer, int n, int banderas,
        struct sockaddr *dir, int longitud);
```

lee los datos escritos por *sendto*.

8.7 Destrucción de sockets

Una vez que un socket ha dejado de ser útil es posible utilizar la función:

```
int close(int sock);
```

para liberar los recursos utilizados por el socket asociado con el descriptor *sock*. Esta función limpia los buffers internos del sistema operativo enviando la información que pudo haber quedado acumulada en un socket.

8.8 Estructura de clientes y servidores

Dentro de la Internet una buena parte de los servicios de red se encuentran implementados siguiendo el esquema de cliente-

servidor. En esta sección se presentan los esqueletos de un posible cliente y servidor; por supuesto, puede haber variaciones de este esquema.

Antes de poder intercambiar información, un cliente debe crear un socket y conectarlo al servidor. Como ejemplo, se podría tener el siguiente fragmento de programa:

```
#include <sys/socket.h>        /* Declaración de sockets */
#include <netinet/in.h>        /* Aspectos específicos a la
                                  Internet */

...

cliente(...) {
     int s;                    /* Descriptor de socket */
     struct sockaddr_in dir;   /* Dirección del servidor */
     /*
          Crea un socket s dentro de la familia de protocolos
          TCP/IP para utilizar un servicio de transporte
          confiable con conexión.
     */
     s = socket(AF_INET, SOCK_STREAM, 0);
     /*
          Obtiene la dirección del servidor.
     */
     dir = ...
     /*
          Inicia la conexión con el servidor
     */
     connect(s, &dir, sizeof dir);
     /*
          Inicia la petición del servicio y obtiene los
          resultados del servidor.
     */
     ...
     write(s, ...);
     read(s, ...);
     ...
     /*
          Limpia el socket
     */
     close (s);
}
```

La estructura de un servidor es un poco más compleja. Un servidor, además de crear un socket, debe fijarlo al puerto asociado con el servicio que presta, especificar cuántas peticiones de conexión simultáneas puede almacenar, esperar conexiones y prestar el servicio:

```
#include <sys/socket.h>
#include <netinet/in.h>

...

servidor(...) {
     int u, t;                    /* Descriptores de socket */
     struct sockaddr_in dirl,     /* Dirección local */
                        dirr;     /* Dirección remota */

     u = socket(AF_INET, SOCK_STREAM, 0);
     /*
          Obtiene la dirección del puerto local en donde debe
          esperar conexiones.
     */
     dirl = ...
     /*
          Fija el socket a la dirección local.
     */
     bind(u, &dirl, sizeof dirl);
     /*
          Fija el número máximo de peticiones simultáneas que
          se pueden almacenar a 5.
     */
     listen(u, 5);
     /*
          Entra en un ciclo en el que espera conexiones y las
          atiende.
     */
     while(1) {
          /*
               Espera una petición de conexión en u y crea un
               nuevo socket t cuando ésta ocurre.
          */
          int l = sizeof dirr;
          t = accept(u, &dirr, &l);
          /*
               Presta el servicio utilizando el nuevo socket.
          */
          servicio (t);
```

```
            /*
                 Destruye el socket por el cual prestó el
                 servicio.
            */
            close(t);
        }
    }
```

El servidor anterior únicamente puede atender una petición de servicio a la vez. Sin embargo, es posible prestar servicio a varios clientes en forma concurrente creando procesos asíncronos que atiendan cada una de las peticiones (en Unix, esto se puede hacer con la llamada al sistema *fork*):

```
        while(1) {
            int l = sizeof dirr;
            t = accept(u, &dirr, &l);
            /*
            Crea un nuevo proceso que prestará el servicio.
            */
            if (fork() == 0) {
                /*
                     Cierra el socket por el cual se aceptan
                     las conexiones.
                */
                close(u);
                /*
                     Presta el servicio utilizando el nuevo
                     socket.
                */
                servicio (t);
                /*
                     Destruye el socket por el cual prestó el
                     servicio.
                */
                close(t);
                /*
                     Termina el proceso hijo.
                */
                exit(0);
            }
            /*
                 Cierra el socket por el cual el nuevo proceso
                 prestará el servicio.
            */
            close (t);
        }
```

La figura 8.1 muestra el diagrama de flujo del modelo descrito anteriormente (con un servidor secuencial).

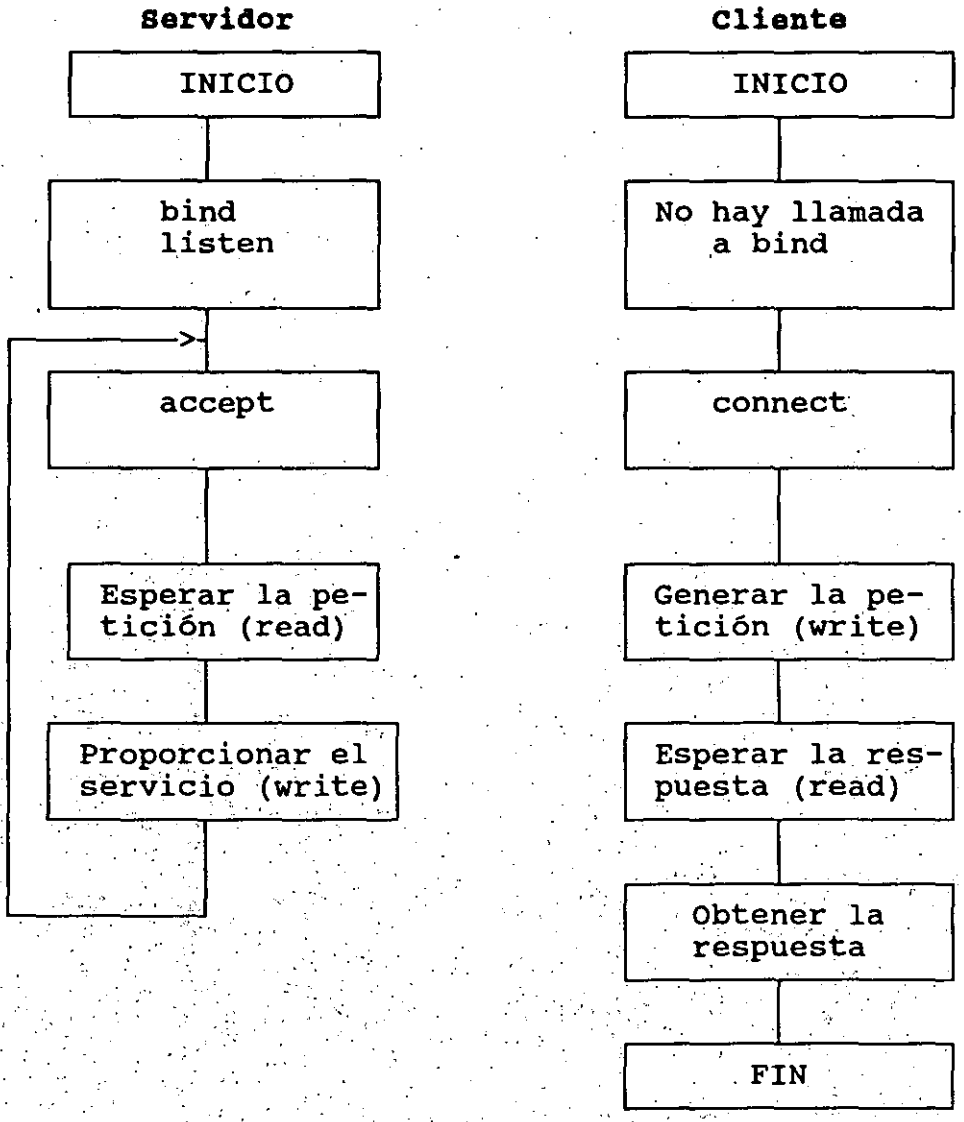


Fig. 8.1. Esquema del modelo cliente-servidor

8.8 Otras llamadas al sistema relacionadas con sockets

Además de las llamadas al sistema ya descritas, la interface de Unix proporciona otras cuantas llamadas para manipular sockets.

Entre ellas se encuentran las siguientes:

socketpair	Crea un par de sockets locales conectados (se utiliza en la implementación de *pipes*).
writev, sendmsg	Envío de datos.
readv, recvmsg	Recepción de datos.
getpeername	Determina la dirección a la que se encuentra conectado un socket.
getsockname	Determina la dirección local de un socket.
getsockopt	Obtiene información interna de un socket.
setsockopt	Modifica la "configuración" de un socket.
shutdown	Vacía los buffer del socket.

8.9 Biblioteca de funciones de red

Todas las funciones descritas anteriormente representan llamadas al sistema; no obstante, Unix ofrece además un conjunto de funciones de biblioteca que proporcionan diferentes servicios. La diferencia entre una llamada al sistema y una función de biblioteca es que la primera representa una interface directa con el sistema operativo, mientras que la segunda debe valerse de las llamadas al sistema para poder realizar su tarea.

Las bibliotecas de red de Unix incluyen funciones de conversión de direcciones, conversión de formatos de red, acceso al servidor de nombres y a información sobre los nodos, redes, servicios y protocolos. Nuevamente, aquí sólo se describen las más comunes.

8.10 Información acerca de nodos

La biblioteca de funciones de red proporciona un conjunto de procedimientos que permiten obtener información relacionada con algún nodo de la red. Dichas funciones regresan la información requerida en estructuras del siguiente tipo:

```
struct hostent {          /* Definida en netdb.h */
     char *h_name;        /* Nombre oficial del nodo */
     char **h_aliases;    /* Lista de sinónimos */
     int  h_addrtype;     /* Tipo de dirección */
     int  h_length;       /* Longitud de la dirección */
     char **h_addr_list;  /* Lista de direcciones para el
                             servidor de nombres */
     #define h_addr h_addr_list[0] /* direccion IP */
};
```

La función:

```
struct hostent *gethostbyname(char *nombre);
```

obtiene el registro asociado con el nodo cuyo nombre se especifica en el parámetro *nombre*, o regresa NULL si el nodo no existe. Similarmente

```
struct hostent *gethostbyaddr(char *dir, int l, int t);
```

obtiene el registro para el nodo con dirección *dir*. Los parámetros *l* y *t* especifican la longitud y tipo de esa dirección.

Este conjunto de funciones obtienen la información mencionada del servidor de nombres si es posible, o del archivo /etc/hosts dependiendo de la configuración del sistema.

8.11 Información de protocolos

Las funciones para obtener información relativa a los protocolos con que cuenta un nodo son muy parecidas a las descritas en la sección anterior. En este caso, dicha información se mantiene en el archivo /etc/protocols, cuyos registros pueden ser obtenidos usando la estructura:

```
struct protoent {         /* Definida en netdb.h */
     char *p_name;        /* Nombre oficial del protocolo */
     char **p_aliases;    /* Lista de sinónimos */
     long p_proto;        /* Número de protocolo */
};
```

Cada protocolo puede ser obtenido a partir de su nombre con la función:

```
struct protoent *getprotobyname(char *nombre);
```

o partir de su número oficial con la función:

```
struct protoent *getprotobynumber(int numero);
```

8.12 Información de servicios.

La información de los servicios puede se obtenida a partir del archivo /etc/services utilizando la estructura:

```
struct servent {          /* Definida en netdb.h */
     char *s_name;        /* Nombre oficial del servicio */
     char  s_aliases;     /* Lista de sinónimos */
     long  s_port;        /* Puerto en que reside el serv. */
     char *s_proto;       /* Protocolo a usar */
};
```

La función:

```
struct servent getservbyname(char *nombre, char *protocolo);
```

permite obtener la información del servicio llamado *nombre*; si el parámetro *protocolo* no es nulo, el servicio requerido deberá ser prestado por dicho protocolo.

8.13 Conversión de formatos de direcciones

En muchas ocasiones las aplicaciones de red deben manipular direcciones tanto en formato binario como en formato decimal; la biblioteca de Unix proporciona funciones que permiten realizar conversiones entre dichos formatos. Las funciones **inet_addr** e **inet_network** convierten una dirección en formato decimal a una dirección IP de 32 bits. La primera, obtiene una dirección IP completa, mientras que la segunda llena con ceros la parte de la dirección correspondiente al nodo:

```
long inet_addr(char *dir);
long inet_network(char *dir);
```

La función **inet_toa** realiza la función inversa, al convertir una dirección binaria a formato decimal:

```
char *inet_toa(long dir);
```

Otro tipo de conversiones incluyen a las funciones:

```
long inet_makeaddr(long red, long host);
```

que combina las direcciones IP de un *nodo* y la *red* en que se encuentra,

```
long inet_netof(long dir);
```

que extraer la parte de red de la dirección IP *dir*, y

```
long inet_lnaof(long dir);
```

que obtiene la dirección del nodo local especificada por *dir*.

8.14 Otras funciones de biblioteca.

Otras de las funciones útiles que provee la biblioteca de Unix son las siguientes:

a) Información de redes:

```
struct netent *getnetbyname(char *nombre);
struct netent *getnetbyaddr(long dir);
```

b) Acceso secuencial a las bases de datos de nodos, protocolos, servicios y redes:

```
int sethostent(int m);
struct hostent *gethostent();
int endhostent();

int setprotoent(int m);
struct protoent *getprotoent();
int endprotoent();

int setservent(int m);
struct servent *getservent();
int endservent();

int setnetent(int m);
struct netent *getnetent();
int endnetent();
```

c) Acceso al Sistema de Dominio de Nombres:

```
int res_init();
int res_mkquery(int op, char *nombre, int clase,
                int tipo, char *datos, int longdatos,
                int newrr, char *buffer, int longbuffer);
int res_send(char *mensaje, int longmensaje,
             char *respuesta, int *longrespuesta);
```

d) Conversión de orden de bytes:

```
short ntoh(short corto_de_red);
long ntohl(long largo_de_red);
short htons(short corto_local);
long htonl(long largo_local);
```

8.15 Un programa de ejemplo.

En esta última sección se presenta un ejemplo de servicio de red que sigue el esquema cliente-servidor. Dicho ejemplo consta de dos programas basados en las funciones descritas en este capítulo. El ejemplo implementa una versión sencilla de la utilería **write** de Unix, extendida para poder establecer "conversaciones telefónicas" dentro de una red.

```
/*
 *-------------------------------------------------------------
 *
 * Programa:    write
 *
 * Propósito:   Servicio de mensajes hacia maquinas remotas.
 *              Requiere que la máquina destino corra el programa
 *              'writeserver'.
 *
 * Uso:         write nodo username
 *
 * Autor:       Norberto Arrieta, UNAM
 *
 * Fecha:       Abril, 1991
 *
 *-------------------------------------------------------------
*/

#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
```

```
#include <netdb.h>
#include <errno.h>
#include <pwd.h>

#define NOPERMISO       1
#define NOSESION        2
#define OK              3

#define MAXLONG 16

main(int argc, char *argv[]) {
 int s;                     /* Socket */
 int l;                     /* Longitud de datos recibidos */
 struct sockaddr_in dir;    /* Dirección IP de la máquina remota */
 struct hostent *nodo;      /* Información del nodo remoto */
 struct servent *servicio;  /* Información del servicio */
 char buf[BUFSIZ + 1];      /* Buffer */
 char nombre[MAXLONG + 1]; /* Nombre del nodo local */
 char error;                /* Código de error */

        if (argc!=3) {
                fprintf(stderr,"Uso: %s nodo username\n", argv[0]);

                exit(1);
        }

        /* Obtiene la dirección del nodo */
        if ((nodo = gethostbyname(argv[1])) == NULL) {
                fprintf(stderr,"%s: No se tiene acceso a %s.\n",
                        argv[0], argv[1]);
                exit(1);
        }
        /* Obtiene el puerto del servicio */
        if((servicio = getservbyname("write","tcp")) == NULL){
                fprintf(stderr,"%s No hay servicio  de \"write\""
                        "en este nodo\n",argv[0]);
                exit(1);
        }
        /* Llena la Dirección IP del nodo remoto */
        dir.sin_family = nodo->h_addrtype;
        dir.sin_port = servicio->s_port;
        bcopy((char *)nodo->h_addr, (char *)&dir.sin_addr,
              nodo->h_length);
         /* Crea un nuevo socket y lo conecta con el nodo
            remoto */
        if ((s= socket(nodo->h_addrtype, SOCK_STREAM, 0)) < 0) {
                perror("socket");
```

```
                exit(1);
        }
        if (connect(s, &dir, sizeof dir) < 0) {
                perror("connect");
                exit(1);
        }
        /* Envia el nombre del usuario remoto y del usuario y
           máquina locales */
        gethostname(nombre, MAXLONG);
        sprintf(buf, "%s:%s@%s", argv[2],
                getpwuid( getuid())->pw_name, nombre);
        write(s, buf, strlen(buf));
        /* Checa la respuesta del nodo remoto */
        read(s, &error, sizeof error);


        if (error == NOSESION)
                fprintf(stderr,"%s: %s no se encuentra en sesión"
                        " en %s.\n", argv[0], argv[2], argv[1]);
        else if (error == NOPERMISO)
                fprintf(stderr,"%s: %s no desea recibir"
                        " mensajes.\n", argv[0], argv[2]);
        else if (error == OK)
                while ((l = read(0, buf, BUFSIZ)) > 0)
                        write(s, buf, l);
        close(s);
        exit(0);
}
```

```
/*
 *-------------------------------------------------------------
 *
 * Programa:    writeserver
 *
 * Propósito:   Servidor para mensajes hacia maquinas remotas
 *              (servicio 'write'). Utiliza el puerto 2000 para
 *              recibir peticiones (el servicio WRITE se dio de
 *              alta en /etc/services).
 *
 * Uso:         writeserver &
 *
 * Autor:       Norberto Arrieta Márquez
 *
 * Fecha:       Abril, 1991
 *
 *-------------------------------------------------------------
 */

#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <sys/file.h>
#include <limits.h>
#include <utmp.h>
#include <signal.h>

#define BACKLOG 5 /* número de peticiones que se pueden encolar */
#define MAXLONG 32/* longitud max. del nombre del nodo */

#define NOPERMISO       1
#define NOSESION        2
#define OK              3

struct utmp * getsessionbyuser(char *login);
int writeserver(int sock);

char *nombreprog;

main(int argc, char **argv) {
  int s, t;                       /* descriptores de socket */
  struct sockaddr_in sdir, tdir; /* direcciones IP */
  struct hostent *nodo;          /* información del nodo local */
  struct servent *servicio;       /* información del servicio */
  char nombrenodo[MAXLONG+1];     /* nombre del nodo local */
```

```
int i;                                  /* auxiliar */

        nombreprog = argv[0];
        /* Obtiene la información (número de puerto) del servicio
           WRITE */
        if((servicio = getservbyname("writenam","tcp")) == NULL){
                fprintf(stderr, "%s: Este nodo no cuenta con"
                        " servicio write.\n",  nombreprog);
                exit(1);
        }
        /* Obtiene la información (dirección IP) del nodo local */
        gethostname(nombrenodo, MAXLONG);
        if ((nodo = gethostbyname(nombrenodo)) == NULL) {
                fprintf(stderr, "%s: No se puede obtener la"
                        " información del nodo local.\n",
                        nombreprog);
                exit(1);
        }
        /* Llena la estructura de dirección IP (familia, puerto,
           dirección) */
        sdir.sin_family = nodo->h_addrtype;
        sdir.sin_port = servicio->s_port;
        bcopy((char *) nodo->h_addr, (char *) &sdir.sin_addr,
              nodo->h_length);
        /* Crea un socket (s) para escuchar las peticiones de
           conexión. */
        if ((s = socket(nodo->h_addrtype, SOCK_STREAM, 0)) < 0) {
                perror("socket");
                exit(1);
        }

        /* Fija el socket al puerto de servicio para poder oir las
           peticiones */
        if (bind(s, &sdir, sizeof sdir) < 0) {
                perror("bind");
                exit(1);
        }
        /* Fija el número máximo de conexiones en espera */
        listen(s, BACKLOG);
        /* Entra en un ciclo infinito esperando nuevas
           conexiones */
        while (1) {
                i = sizeof tdir;
                /* Espera una nueva conexión */
```

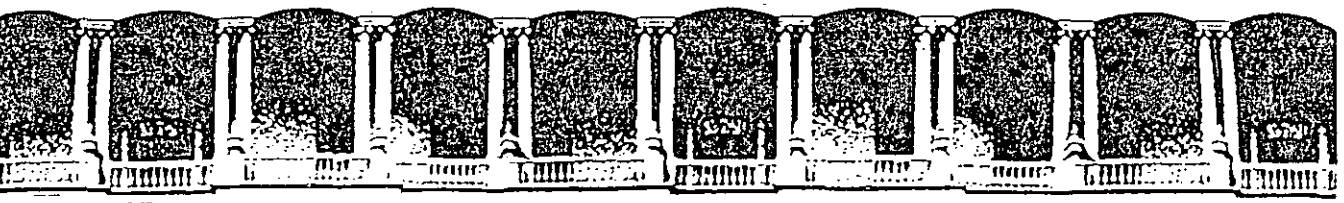
```
                if ((t = accept(s, &tdir, &i)) < 0) {
                        perror("accept");
                        exit(1);
                }
                /* Crea un nuevo proceso que atiende la
                   conexión */
                if (fork() == 0) { /* proceso hijo */
                        close(s);
                        writeserver(t);
                        close(t);
                        break;
                }
                close(t);
        }
}

int writeserver(int sock) {
     char destino[BUFSIZ + 1];  /* A quien se dirige el mensaje */
     char origen[BUFSIZ + 1];   /* El que manda el mensaje */
     char nombreterm[14];       /* Nombre de la terminal */
     int term;                  /* Descriptor de la terminal */
     int estado;                /* Código de error del mensaje */
     struct utmp *sesión;       /* Info de la sesión de destino */
     char buf[BUFSIZ + 1];      /* Buffer recepción/transmisión */
     int l;                     /* Longitud de datos recibidos */

        /* Lee el nombre del usuario a quien va dirigido el mensaje
           y el nombre y nodo del usuario que inició la
           conexión */
        l = read(sock, buf, BUFSIZ);
        buf[l] = '\0';
        for (l = 0; buf[l] != ':'; l++)
                destino[l] = buf[l];
        destino[l] = '\0';
        strcpy(origen, buf + l + 1);
        /* Obtiene la terminal del usuario a quien se dirige el
           mensaje */
        sesión = getsessionbyuser(destino);
        if (sesión == NULL) { /* El usuario no está en sesión */
                estado = NOSESION;
                write(sock, &estado, sizeof estado);
                return 0;
        }
        /* Abre la terminal */
        sprintf(nombreterm, "/dev/%s", sesión->ut_line);
```

```
        if ((term = open(nombreterm, O_WRONLY, 0)) == -1) {
                /* No se tiene permiso de escritura a la
                   terminal */
                estado = NOPERMISO;
                write(sock, &estado, sizeof estado);
                return 0;
        }
        estado = OK;
        write(sock, &estado, sizeof estado);
        /* Avisa al usuario que va a recibir un mensaje */
        sprintf(buf, "\07\07\t*** %s: llamada de %s ***\n\n",
                nombreprog, origen);
        write(term, buf, strlen(buf));
        /* Despliega el mensaje */
        while ((l = read(sock, buf, BUFSIZ)) > 0)
                write(term, buf, l);
        /* Despliega un aviso de fin de mensaje */
        sprintf(buf, "\07[EOF]\n");
        write(term, buf, strlen(buf));
}

struct utmp * getsessionbyuser(char *login) {
        static struct utmp r;
        int futmp;
        if ((futmp = open("/etc/utmp", O_RDONLY, 0)) == -1) {
                fprintf(stderr, "%s: error al tratar de abrir"
                        " /etc/utmp.\n", nombreprog);
                exit(1);
        }
        while (read(futmp, &r, sizeof r) > 0)
                if (!strncmp(r.ut_name, login, 8)) {
                        close (futmp);
                        return &r;
                }
        return NULL;
}
```



FACULTAD DE INGENIERIA U.N.A.M.
DIVISION DE EDUCACION CONTINUA

Universidad Nacional Autónoma de México

Facultad de Ingeniería

Secretaría General

Centro de Cálculo

Introducción a la RedCECAFI

Norberto Arrieta Márquez
Marzo, 1992

RedCECAFI

A inicios de 1990, el Centro de Cálculo de la Facultad de Ingeniería instaló una pequeña red local para unir a sus sistemas de cómputo principales. Hasta hace poco tiempo el uso de dicha red se había limitado prácticamente a la transferencia de archivos entre los sistemas internos del Centro de Cálculo. Sin embargo, durante 1991 se registraron tres sucesos que han incrementado notablemente las aplicaciones de esta red:

* La integración del Centro de Cálculo a la RedUNAM
* La conexión de computadoras personales a la red de CECAFI
* La adquisición de estaciones de trabajo y su conexión a dicha red

Este documento tiene como intención presentar una introducción al uso de las facilidades ofrecidas por la red del Centro de Cálculo, la RedCECAFI. Se asume que el lector tiene conocimientos básicos de Unix y VMS.

Si desea obtener información adicional sobre alguno de los temas aquí tratados, o hacer cualquier sugerencia sobre este folleto, puede dirigirse al Departamento de Soporte Técnico de éste Centro al teléfono 550-5734, ext.5734, o mediante correo electrónico a info@kelem.cecafi.unam.mx.

I. Configuración de la RedCECAFI

Este primer capítulo tiene como propósito describir los elementos de hardware que componen a la red del Centro de Cálculo. En él se describen la topología de esta red y la configuración de los sistemas que la componen.

I.1 Topología de bus

De manera breve, podemos definir la **topología** de una red de computadoras como la forma de las conexiones entre sus elementos. Durante el desarrollo de la tecnología de redes han surgido diversas topologías, entre las que destacan las conocidas como bus, anillo, estrella y punto a punto.

En una topología de bus, todas las computadoras se conectan a un cable central que se utiliza como medio de transporte durante la transferencia de datos. Tal como se muestra en la figura 1.1, los nodos de un bus comparten el mismo medio y se encuentran al mismo nivel; es decir, no existe un nodo central del cual dependan el resto de los nodos.

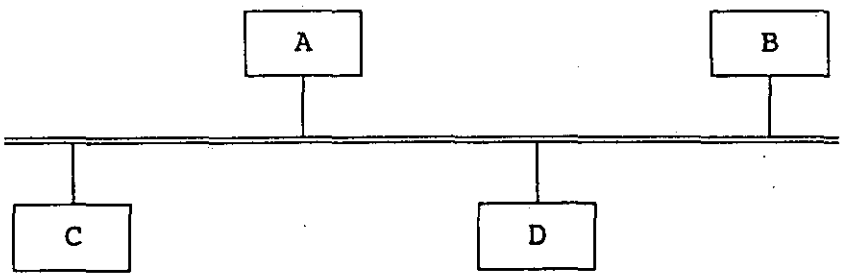


Figura 1.1 Topología de bus

I.2 Topología de la RedCECAFI

La RedCECAFI se basa en la tecnología **Ethernet**, que es una de las más populares en la actualidad. La topología de Ethernet es una derivación del bus mencionado más arriba (**bus lineal**), llamada **bus ramificado**. En ésta puede haber varios buses conectados mediante dispositivos llamados **repetidores**, que se encargan de duplicar la señal en cada uno de ellos, tal como lo ilustra la figura 1.2.

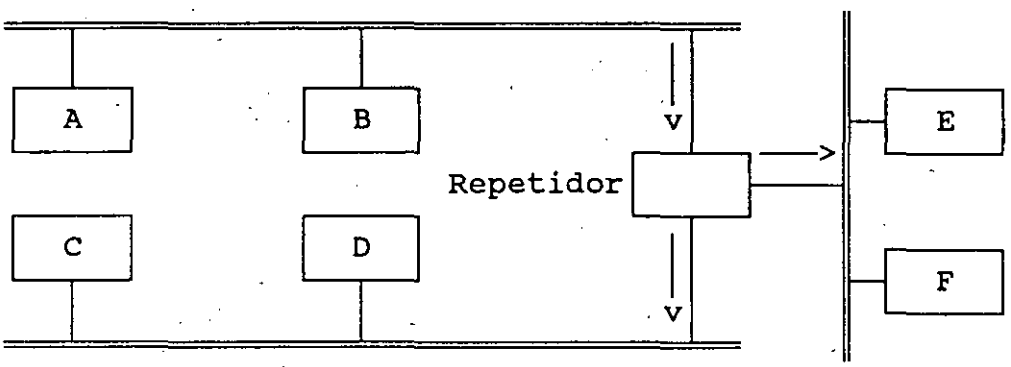


Figura 1.2. Topología de bus ramificado

La topología exacta de la RedCECAFI se muestra en la figura 1.3. En ella se observan cinco buses unidos por un repetidor multipuerto; en los cinco buses se distribuye el siguiente equipo: dos microcomputadoras VAX, tres estaciones de trabajo, catorce computadoras personales y cuatro servidores de terminales (con una capacidad de 16 terminales cada uno).

Este equipo se utiliza con diversos fines y cuenta con puntos de acceso en varios lugares de la Facultad: el Centro de Cálculo en sus dos edificios; la División de Ciencias Básicas; la División de Ingeniería Mecánica e Industrial; la División de Ingeniería Eléctrica, Electrónica y en Computación; la División de Ingeniería Civil; el Departamento de Fluidos y Térmica; la Unidad de Apoyo Editorial; la Secretaría de Servicios Escolares y la Oficina de Servicios Escolares.

La red del Centro de Cálculo se encuentra conectada a la RedUNAM mediante un repetidor de fibra óptica. La RedUNAM tiene sus nodos principales en la Dirección General de Servicios de Cómputo Académico, el Instituto de Matemáticas Aplicadas y Sistemas y el Instituto de Astronomía, aunque incluye a muchos otros institutos y facultades. Además de proporcionar la capacidad de comunicación entre cualquiera de sus nodos, la RedUNAM presta los servicios de

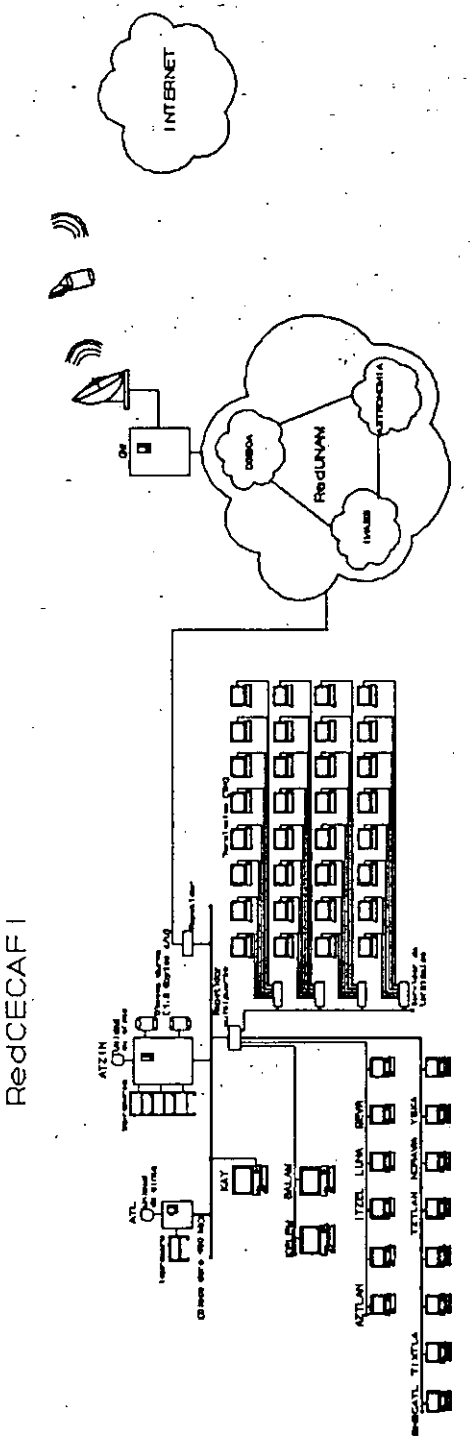


Figura 1.3 RedCECAFI

acceso a las facilidades de supercómputo y de conexión con la Internet, que se describen más adelante.

II.3 Configuración de los nodos

A continuación se hace una pequeña descripción de los nodos que componen a la RedCECAFI y su configuración de hardware.

El nodo principal de la red, Atzin, es una minicomputadora VAX 6000-210 construida por Digital Equipment Corporation. El hardware de esta computadora incluye

memoria principal de 32 Mbytes,
dos discos duros de 1.2 Gbytes cada uno,
una unidad de cartucho TK70,
una unidad de cinta TU81,
un graficador LG02 de 600 líneas por minuto,
una impresora LP37 de 1200 líneas por minuto y
una impresora láser LN03.

Otro sistema conectado a la red es Atl, una minicomputadora MicroVAX 3400, construida también por DEC. Esta computadora tiene las siguientes características:

memoria principal de 12 Mbytes,
un disco duro de 400 Mbytes,
una unidad de cartucho TK70,
una unidad de cinta TS05 y
una impresora LPA0 de 600 líneas por minuto.

La RedCECAFI ofrece también el acceso a varias estaciones de trabajo, computadoras con hardware orientado al despliegue de gráficos. Los nodos Kay, Balam y Kelem son máquinas de este tipo; sus componentes son:

* Kay (Control Data)
  Memoria principal de 16 Mbytes,
  disco duro de 400 Mbytes,
  una unidad de cartucho QIC-120/150 y
  video con resolución de 1280 x 1024

* BALAM (Hewlett-Packard serie 400)
  Memoria principal de 12 Mbytes,
  un disco duro de 210 Mbytes y
  video con resolución de 1280x1024

* KELEM (Digital Equipment Corporation, modelo DECsystem 3100)
  Memoria principal de 12 Mbytes,
  un disco duro de 1 Gbyte,
  una unidad de cartucho TK50 y
  video con resolución de 1024 x 864 pixeles.

El resto de los nodos de la red son computadoras personales con diferentes configuraciones, que incluyen:

- Computadoras 286, con 1 Mbyte de RAM, 40 Mbytes en disco duro y monitor VGA
- Computadoras 286, con 6 Mbytes de RAM, 80 Mbytes en disco duro y monitor super VGA
- Computadoras 386, con 8 Mbytes de RAM, 80 Mbytes en disco duro y monitor super VGA

Además, la red cuenta con cuatro servidores de terminales que permiten la conexión con los nodos Atl, Atzin y Kelem desde terminales de tipo carácter.

Es importante notar que, a pesar de que cada máquina tiene sus propios recursos, el software de la red permite usar los recursos de hardware y software de un nodo desde cualquier otro. Por mencionar un ejemplo, una computadora personal puede usar como almacenamiento secundario el disco de Atzin o utilizar la impresora láser de esa máquina.

I.4 Nombres de los nodos

Cada nodo de una red posee un nombre que permite a sus usuarios diferenciarlo de los demás nodos. Las máquinas de la RedCECAFI tienen como nombres palabras de origen nahuatl y maya. La tabla 1.1 describe brevemente el significado de cada uno de ellos.

Nombre	Significado
Atl	Agua, en nahuatl.
Atzin	Uno de los fundadores de Tenochtitlán.
Aztlán	Lugar de Garzas, en nahuatl. Lugar mítico en el que los aztecas iniciaron su recorrido en busca de un lugar en donde asentarse.
Balam	Tigre o protector sobrenatural, en maya.
Ehécatl	Viento, en nahuatl.
Kay	Música, en maya.
Kelem	Fuerte, en maya.

Tabla 1.1 Nombres de los nodos de RedCECAFI

II. Servicios de la RedCECAFI

Los nodos de la RedCECAFI cuentan con un conjunto significativo de utilerías que pueden ser accesadas por cualquiera de sus usuarios. En este capítulo se hace una breve descripción de la mayor parte de esas utilerías; en capítulos posteriores se describirán con mayor detalle las que se refieren a comunicaciones. Cabe mencionar que el Centro de Cálculo cuenta con documentación de cada una de las utilerías que aquí se mencionan y que dicha documentación se encuentra a disposición de sus usuarios.

II.1 Ambiente VMS

El sistema operativo VMS (Virtual Memory System) fue desarrollado por Digital Equipment Corporation para su serie de computadoras VAX. VMS ha llegado a ser un sistema operativo muy común en diversos ambientes, tales como universidades, empresas gubernamentales y muchos otros. El Centro de Cálculo cuenta con sistemas VMS desde 1982, año en el que se adquirió una computadora VAX 11-780; actualmente cuenta con una computadora VAX 6000-210 (Atzin) con la versión 5.3 de VMS.

Atzin se utiliza para diversos fines, entre los que destacan el desarrollo de aplicaciones académico-administrativas para la Facultad de Ingeniería (sistema de inscripciones, generación de horarios, asistencia de profesores, etc,) y el apoyo a actividades académicas de los profesores y alumnos de la misma.

Las utilerías de software que se encuentran instaladas en esta máquina actualmente, son las siguientes:

* Compiladores
  - ANSI C
  - Basic
  - Cobol
  - Fortran
  - Pascal
* Manejadores de bases de datos
  - Adabas
  - RDB
  - Oracle
* Procesadores de texto
  - Runoff
* Productos de comunicación
  - Decnet
  - Pathworks
  - LAT
* Otros
  - FMS (Sistema Administrador de Formas)
  - CPQ (Interface amigable para ejecutar otras aplicaciones)

II.2 Ambiente UNIX

Este sistema operativo fue desarrollado por la AT&T a inicios de los setentas; Unix es el sistema operativo más empleado actualmente y se está convirtiendo rápidamente en un estándar mundial. Existe una gran variedad de versiones de Unix: Ultrix, Xenix, Aix, Dynix, SCO Unix, etc.

Las primeras máquinas con Unix con las que contó el Centro de Cálculo fueron dos microcomputadoras Altos; actualmente cuenta con una MicroVAX 3400 (Atl) y una DECstation 3100 (Kelem) con Ultrix (Unix de DEC), una HP serie 400 (Balam) con Domain (Unix de HP), una Control Data (Kay) con Unix BSD y dos PC386 (Aztlán y Ehécatl) con SCO Unix.

El principal uso que se les da a estos nodos es académico, para atención a alumnos y profesores de la Facultad y para investigaciones internas del Centro de Cálculo. Una característica importante de estas máquinas es que el software con el que cuentan permite el acceso a la Internet, la red más importante a nivel mundial en la actualidad.

Todos esos nodos cuentan con el siguiente software:

* Compiladores
  - ANSI C
* Procesadores de texto
  - Nroff
* Productos de comunicación
  - TCP/IP
* Herramientas de desarrollo de sistemas
  - Make
  - SCCS (Sistema de Control de Código Fuente)
  - Yacc
  - Lex

Adicionalmente, cada nodo cuenta con

* ATL
  + Compiladores (y/o intérpretes)
    - APL
    - C K&R
    - Fortran
    - Franz-Lisp
    - Modula
    - Pascal
  + Manejadores de bases de datos
    - Ingres
    - Ultrix/SQL
  + Productos de comunicación
    - Decnet
    - LAT
  + Juegos
    - Ahorcado
    - Ajedrez
    - Guerra de las Estrellas
    - Gusanos
    - etc.

* KELEM
  + Compiladores
    - Fortran
  + Procesadores de texto
    - Troff
  + Productos de comunicación
    - Decnet
    - LAT

+ Ambientes gráficos
  - DECwindows
  - Motif
  - X-windows
+ Herramientas de desarrollo de sistemas
  - Motif
  - X-windows
+ Juegos
  - Tetris
  - Ajedrez
  - Guerra de las Estrellas
  - Quince
  - etc.

* KAY
  + Compiladores
    - Fortran 77
  + Ambientes gráficos
    - Motif
    - X-windows
    - RISC-Windows
  + Herramientas de desarrollo de sistemas
    - Motif
    - X-windows

* BALAM
  + Ambientes gráficos
    - Motif
    - X-windows
  + Herramientas de desarrollo de sistemas
    - Motif
    - X-windows

* AZTLAN, EHECATL
  + Procesadores de texto
    - Word Perfect
    - Word

  + Ambientes gráficos
    - Open Desktop
    - Motif
    - X-Windows
  + Bases de datos
    - Informix
    - Ingres

II.3 Ambiente MS-DOS

El Centro de Cálculo cuenta también con más de 80 computadoras personales con MS-DOS. CECAFI cuenta con un conjunto de utilerías para estas computadoras que incluye a los paquetes más usados, como procesadores de texto (WORDPERFECT), bases de datos (ORACLE), paquetes de dibujo (AUTOCAD) y muchos otros.

Los usuarios de estas máquinas cuentan además con una impresora láser y un graficador a colores, además de la documentación de los paquetes de software que se usan en ellas.

III. Protocolos

Un **protocolo de comunicación** es un conjunto de reglas que especifican de manera precisa la manera en que dos computadoras pueden intercambiar información. En el mercado actual existen diversos protocolos; entre los más importantes se encuentran TCP/IP, DECNET, X.25, LAT y los protocolos OSI. Es importante recordar que dos computadoras sólo pueden comunicarse si cuentan con el mismo protocolo o con una utilería de conversión de protocolos.

Las máquinas de la RedCECAFI cuentan con tres de los protocolos mencionados anteriormente: TCP/IP, DECNET y LAT. Las siguientes secciones se encargan de describir brevemente estos protocolos; los capítulos subsecuentes se ocupan de las utilerías desarrolladas para esos protocolos de manera más profunda.

III.1 TCP/IP

Los protocolos TCP/IP[1] fueron desarrollados durante la década de los setentas bajo auspicios del Departamento de Defensa de los Estados Unidos, con el propósito de establecer conexiones entre sus principales centros de investigación y las universidades y corporaciones comerciales y gubernamentales de mayor importancia en los Estados Unidos. La conexión de computadoras que resultó de ese proyecto es actualmente la red de mayor tamaño en todo el mundo: la Internet.

La Internet cuenta con alrededor de 313,000 nodos distribuidos en 40 países, entre los que se encuentran Estados Unidos, Inglaterra, Francia, Alemania, Holanda, México y varios países de Latinoamérica. Recientemente, los administradores de la Internet

[1] Comúnmente se hace referencia a la familia de protocolos TCP/IP como si fuera un sólo protocolo.

han reportado un promedio de 500,000 usuarios diarios en esta red. La Internet proporciona además conexiones con las redes de mayor importancia a nivel mundial, tales como BITNET, ACSNET, UUCPNET, EASYNET, etc.

La RedCECAFI se encuentra conectada a la Internet a través de la RedUNAM, por lo que sus usuarios tienen acceso a prácticamente cualquier lugar del mundo (recordemos que los nodos con TCP/IP en CECAFI son Atl, Kelem, Kay, Balam, Ehécatl y Aztlán).

TCP/IP (Transport Control Protocol/Internet Protocol) ofrece tres servicios básicos:

* Correo electrónico

  Que permite intercambiar mensajes pequeños entre dos usuarios cualquiera de la red. A través de este servicio, la Internet implementa un conjunto de foros de discusión sobre diversos temas, en los que es posible intercambiar información sobre un tema específico con todos los usuarios de la red.

* Transferencia de archivos

  Este servicio consiste en obtener copias de archivos localizados en máquinas remotas. La Internet permite el acceso a bases de datos de las que se puede obtener software de dominio público e información sobre diversos temas.

* Sesión remota

  Permite entablar una sesión en cualquier máquina conectada a la red. La RedUNAM ofrece el acceso al servicio de supercómputo, que permite la ejecución de aplicaciones en la supercomputadora CRAY-YMP.

III.2 DECNET

DECNET es el conjunto de hardware y software de comunicaciones desarrollado por Digital Equipment Corporation. DECNET está entre los protocolos más utilizados actualmente; su importancia radica en el hecho de que su diseño (a partir de su versión 5) se ajusta a los estándares de la OSI y, aunque en la actualidad son pocos los fabricantes que también lo hacen, se espera que a finales de los noventas el modelo OSI sea un estándar universal.

Dentro de la RedUNAM son pocos los nodos que cuentan con DECNET, por lo que su uso es limitado. El Centro de Cálculo cuenta con las versiones de DECNET para VMS, Ultrix y MS-DOS, a través de las cuales ofrece los siguientes servicios:

- Correo electrónico
- Transferencia de archivos
- Sesiones remotas
- Servicios de disco para computadoras personales
- Servicios de impresión para computadoras personales

III.3 LAT

El protocolo LAT (Local Area Transport) permite la comunicación de sistemas de computadoras con dispositivos de entrada/salida o de comunicaciones, tales como terminales, impresoras y modems. A través de LAT es posible, por ejemplo, que varias computadoras compartan las mismas terminales o impresoras.

Las terminales que se utilizan en el Centro de Cálculo se encuentran conectadas a servidores LAT, por lo que pueden ser usadas para establecer sesiones con los nodos Atzin, Atl y Kelem, que cuentan también con ese protocolo. Los servidores de terminales permiten además tener varias sesiones simultáneas en diferentes máquinas.

IV. LAT: Servidores de terminales

El protocolo LAT (Local Area Transport) fue diseñado para funcionar sobre una red Ethernet, con el propósito de permitir la comunicación entre sistemas de computadoras y otros dispositivos como terminales, impresoras y modems.

Este capítulo describe el subconjunto de comandos LAT que permite establecer y manipular sesiones en un sistema de computadoras.

IV.1 Inicio de una sesión LAT

Como ya se mencionó anteriormente, las terminales con las que cuenta el Centro de Cálculo no se conectan directamente a alguna computadora, sino que se encuentran conectadas a servidores LAT. Estos servidores poseen un modo de comandos con el cual es posible establecer conexiones con los nodos de la red que también cuentan con LAT; sin embargo, para hacer uso de esos comandos es necesario primero establecer una sesión en alguno de los servidores.

Al encender una terminal, el usuario debe presionar varias veces la tecla *RETURN* hasta obtener una respuesta del servidor. En la RedCECAFI, esa respuesta es una serie de letreros informativos seguidos del prompt "*Enter username*>", que solicita el nombre del usuario. Dicho nombre puede ser cualquier cadena y únicamente se utiliza con propósitos de identificación; una vez que se haya proporcionado el nombre, el usuario se encontrará en el **modo local**, y podrá hacer uso de cualquiera de los comandos que se describen en las siguientes secciones.

Enseguida se muestra un ejemplo del procedimiento descrito en el párrafo anterior[1]:

```
<RETURN>
<RETURN>
DECserver 300 Terminal Server V1.0 (BL7) - LAT V5.1
CECAFI UNAM VAX/VMS 5-4.2, Ultrix 4.0

Please enter HELP if you need assistance
Enter username> nam                          Nombre del usuario

Local>                                       Modo local
```

IV.2 Servicios LAT

Un **servicio** LAT es un elemento de hardware o software (o una combinación de ambos) que puede ser utilizado desde un nodo LAT. El Centro de Cálculo ofrece tres servicios: el acceso a los sistemas Atl, Kelem y Atzin. El usuario puede consultar los servicios disponibles en cualquier momento con el comando

```
Local> show services
Service name    Status          Identification
ATL             Available       Ultrix 4.0 (VAX)
ATZIN           Available       Welcome to VAX/VMS V5.4-2
KELEM           Available       Ultrix 4.2 (RISC)
Local>
```

En este caso, el comando muestra los servicios disponibles en CECAFI.

IV.3 Conexión con un servicio

Una vez que se sabe el nombre del servicio que se desea obtener, el comando *connect* permite establecer una sesión[2]:

```
Local> connect kelem                  Conexión con el servicio Kelem
Local -010- Session 1 to KELEM established
```

[1] Las negritas indican los comandos que debe proporcionar el usuario.

[2] La cadena 'Local' indica el prompt de LAT, miestras que 'kelem>' indica el del sistema operativo. En adelante, cuando sea necesario distinguir el nodo desde el que se ejecuta un comando, se usará como prompt el nombre del nodo seguido de '>'; si no es así, se usará '%' en sistemas Unix y '$' en sistemas VMS.

```
ULTRIX V4.2 (Rev. 96) (kelem.cecafi.unam.mx)        Mensajes de Kelem

login: nam                                          Inicio de la sesión en Kelem
Password:
Last login: Thu Jan 9 13:30
ULTRIX V4.2 (Rev. 96) System #2: Mon Dec 9 18:54:44 CST 1991
...

kelem> cd tesis                                     Se usan algunos comandos durante la sesión
...
kelem> logout                                       Fin de la sesión
Local -011- Session 1 disconnected from KELEM

Local>                                              De nuevo en modo local
```

Si por alguna razón el servicio no está disponible (falla del sistema operativo, el nombre del servicio es incorrecto, etc.), LAT dará aviso de ello y seguirá intentando la conexión solicitada. El usuario puede interrumpir estos intentos con la tecla "*BREAK*".

Break sirve también para regresar al modo local cuando hay una conexión con un servicio. La sesión se reanuda con el comando *resume*:

```
Local> connect kelem                                Conexión con el servicio Kelem
Local -010- Session 1 to KELEM established

ULTRIX V4.2 (Rev. 96) (kelem.cecafi.unam.mx)        Mensajes de Kelem
...
kelem> <BREAK>                                      Regresar al modo local
Local> resume                                       Reanudar la sesión con Kelem
Local -012- KELEM session 1 resumed
kelem> logout                                       Fin de la sesión en Kelem
Local -011- Session 1 disconnected from KELEM
```

IV.4 Fin de una sesión LAT

El usuario de LAT puede terminar su conexión con el servidor utilizando el siguiente comando:

```
Local> logout
Local -020- Logged out port 8 server SERV02
```

que finaliza las conexiones con todos los servicios.

IV.5. Conexiones múltiples a servicios

LAT permite mantener varias sesiones simultáneas con diferentes servicios. Para esto, basta con usar repetidamente los comandos break y connect:

```
Local> connect kelem                          Conexión con el servicio Kelem
Local -010- Session 1 to KELEM established

ULTRIX V4.2 (Rev. 96) (kelem.cecafi.unam.mx)  Mensajes de Kelem
...
kelem> <BREAK>                                Regresar al modo local
Local> connect atl                            Conexión con el servicio Atl
Local -101- 1 other session(s) active
Local -010- Session 2 to ATL established

ULTRIX V4.2 (Rev. 96) (atl.cecafi.unam.mx)    Mensajes de Atl
...
atl> <BREAK>                                  Regresar al modo local
Local> connect atzin                          Conexión con el servicio Atzin
Local -101- 2 other session(s) active
Local -010- Session 3 to ATZIN established

          Centro de Calculo de la Facultad de Ingenieria
                      Sistema VAX/VMS 6210    Mensajes de Atzin
...
$ <BREAK>
Local>
```

El comando *show sessions* muestra las conexiones establecidas hasta el momento:

```
Local> show sessions
Port 8:          NAM        Local Mode      Current Session: 3
- Session 1: Connected    Interactive     KELEM
- Session 2: Connected    Interactive     ATL
- Session 3: Connected    Interactive     ATZIN
Local>
```

Para restablecer alguna de las sesiones, se utiliza el comando resume especificando el número de la sesión:

```
Local> resume 1                               Reanudar la sesión 1
Local -012- KELEM session 1 resumed
kelem>                                        De nuevo en Kelem
```

Por supuesto, para navegar por las diferentes conexiones se utilizan repetidamente los comandos break y resume:

```
kelem> <BREAK>                                    Modo local
Local> resume 2                                   Sesión en Atl
Local -012- ATL session 2 resumed

atl> <BREAK>
Local> resume 3                                   Sesión en Atzin
Local -012- ATZIN session 3 resumed

$
```

Por último, el comando *disconnect* permite finalizar una o todas las sesiones:

```
Local> disconnect session 1                       Terminar la sesión con Kelem
Local -011- Session 1 disconnected from ATL

Local> disconnect all                             Terminar todas las sesiones
Local -014- All sessions disconnected
```

IV.6 Información adicional

El comando *help* permite desplegar las funciones y forma de uso de cualquier comando LAT. El lector interesado puede consultar la información sobre los siguientes comandos, que resultan útiles durante una sesión LAT:

```
show users      - Muestra los usuarios del servidor
broadcast       - Envía un mensaje hacia cierto puerto
show port       - Despliega las características de un puerto
lock            - Asigna una contraseña a la sesión LAT
```

V. TCP/IP: Sesiones remotas

La mayor parte de los usuarios de una red aprovechan los servicios que ésta ofrece a través de utilerías que les ocultan los detalles del funcionamiento de los protocolos de red. Existe un amplio conjunto de utilerías que se han construido sobre los protocolos TCP/IP; en este capítulo y los siguientes se presentan únicamente las que prestan los servicios básicos de estos protocolos.

Es conveniente hacer notar que existen implementaciones de TCP/IP para diversos ambientes (VMS, Unix, MS-DOS, etc); aquí solo se discuten las utilerías desarrolladas sobre Unix, por lo que posiblemente los comandos difieran en pequeños detalles con otras implementaciones.

V.1 Nombres y direcciones de nodos

En las siguientes secciones se hará referencia continuamente a "nombres" o "direcciones" de máquinas cuando se describan las utilerías de red; por ello, es conveniente aclarar en qué consisten esos nombres y direcciones antes de iniciar la descripción de dichas utilerías.

Para poder utilizar los servicios que ofrece un nodo particular de la red, es necesario contar con un mecanismo que permita diferenciarlo de los demás. Una manera de lograrlo es asignado identificadores únicos a cada uno de los nodos; las utilerías desarrolladas sobre TCP/IP utilizan dos tipos de identificadores: direcciones y nombres.

Las direcciones usadas por los protocolos TCP/IP, llamadas **direcciones IP**, son números enteros de 32 bits. Por ejemplo, la dirección de Atl es

$$10000100111110000011011000000010_2$$

Sin embargo, el uso de esta notación es demasiado complejo, por lo que es más común dividir la dirección en cuatro bytes y escribir sus equivalentes en decimal, separados por puntos (**notación decimal punteada**). De esta manera, la dirección anterior se expresa como

132.248.54.2

$$10000100\ 11111000\ 00110110\ 00000010_2$$
$$\backslash\ /\ \backslash\ /\ \backslash\ /\ \backslash\ /$$
$$132 \quad 248 \quad 54 \quad 2$$

Las direcciones de las máquinas de la RedCECAFI que cuentan con TCP/IP son las siguientes:

Máquina	Dirección
Atl	132.248.54.2
Aztlán	132.248.54.3
Ehécatl	132.248.54.4
Kay	132.248.54.5
Balam	132.248.54.6
Kelem	132.248.54.7

La asignación de direcciones IP se hace de manera jerárquica, así que todas las máquinas de CECAFI tienen direcciones que inician con 132.248.54, mientras que todas las direcciones de las máquinas de la UNAM inician con 132.248.

Aunque este tipo de direcciones pueden aprenderse con facilidad, la mayoría de las personas prefieren usar nombres para identificar a los nodos de una red. La asignación de nombres es también jerárquica. Cada nombre consiste de un **identificador de nodo** y un **dominio**; el dominio del Centro de Cálculo tiene el nombre

cecafi.unam.mx

Cuando se usa el nombre de una máquina dentro de su dominio, basta con especificar el identificador de nodo; sin embargo, cuando se hace desde otro dominio es necesario especificar el nombre completo. Por ejemplo, el nombre de Atl es

atl.cecafi.unam.mx

si se utiliza fuera de CECAFI. Observe que los nombres de máquinas

deben escribirse en minúsculas. Los sistemas Unix mantienen una tabla con los nombres y direcciones de los nodos con lo que comúnmente se comunican en el archivo */etc/hosts;* Atl cuenta con una tabla de aproximadamente 9,000 nodos; el lector interesado puede consultar esa tabla con el comando

atl> **more /etc/hosts**

En adelante, cuando se haga referencia a un *nodo* en la descripción de algún comando, la referencia podrá ser sustituida por la dirección o el nombre del nodo. La mayoría de los comandos acepta tanto nombres como direcciones; cuando esto no suceda, se hará la aclaración pertinente.

V.2 TELNET

El comando **telnet** permite establecer una sesión en cualquier máquina de la red utilizando una terminal conectada a cualquier otra máquina (**sesión remota**[1]). La manera de utilizar este comando es

% telnet *nodo*

Al usar el comando anterior, telnet establece una sesión en el sistema especificado, simulando que la terminal se encuentra conectada directamente a él. Por ejemplo, la siguiente secuencia muestra una sesión remota en Kelem, iniciada desde Atl:

```
atl> telnet kelem                          Inicio de la conexión
Trying...
Conected to kelem.                         La conexión tuvo éxito
Escape character is '^]'

ULTRIX V4.2 (Rev. 96) (kelem.cecafi.unam.mx)     Mensajes de Kelem

login: nam                                 Cuenta con la que se crea la sesión
Password:                                  El password no se despliega
Last login: Thu Jan 9 13:30
ULTRIX V4.2 (Rev. 96) System #2: Mon Dec 9 18:54:44 CST 1991
...

kelem> cd tesis                            Se usan algunos comandos
```

[1]El calificativo 'local' se usará para cualquier entidad localizada en el sistema al cual se conecta directamente el usuario; 'remoto' se utilizará para una entidad localizada en cualquier otro sistema.

```
kelem> vi codificacion.c
...
kelem> logout                                   Fin de la sesión
Connection closed by foreing host.              Fin de la conexión
atl>                                            De nuevo nos encontramos en Atl
```

Resulta claro que después de iniciar la sesión con telnet, se deberán usar los comandos y caracteres de control del sistema remoto (en el ejemplo anterior, tanto Atl como Kelem son sistemas Unix).

V.3 Comandos Telnet

Telnet cuenta con una serie de comandos que pueden auxiliar al usuario durante su conexión con algún sistema. Estos comandos deben utilizarse desde el llamado **modo de comandos** de telnet. Hay dos formas de entrar a este modo: usando a telnet sin especificar ningún nodo, o usando la secuencia de control ^*]*[2], al estar conectado con el sistema remoto. De cualquiera de las dos maneras, telnet indica al usuario que se encuentra en modo de comandos con el prompt *telnet*>. Dentro de este modo, el usuario puede utilizar diversas órdenes para configurar a telnet, así como para establecer y terminar conexiones.

La tabla 5.1 describe los comandos más usuales de telnet:

Comando	Función
open *nodo*	Establece una sesión en el *nodo* indicado
close	Termina la sesión remota
quit	Termina el comando telnet
z	Suspende temporalmente a telnet
status	Despliega el estado de la sesión de telnet

Tabla 5.1 Comandos telnet

El comando z suspende a telnet, lo cual permite reanudar la sesión en la máquina local temporalmente. Si se usa C Shell o Korn Shell, telnet se puede reanudar con el comando *fg* (en Bourne Shell se puede hacer con el comando *kill -CONT pid*), tal como muestra el

[2]El caracter '^' indica que se debe presionar la tecla <CTRL>.

siguiente ejemplo:

```
atl> telnet kelem                          Conexión con Kelem
Trying...
Conected to kelem.
...

kelem> cd tesis                            Sesión remota (Kelem)
kelem> make
...
kelem> ^]                                  Escape a telnet
telnet> z                                  Escape a la sesión local
Stopped                                    Indica la suspensión de telnet
atl> ls -l                                 Sesión local (Atl)
...
atl> jobs
[1]   + Stopped  telnet kelem
atl> fg %1                                 Se reanuda la sesión remota
kelem> rm *.o                              Algunos comandos
...
kelem> logout                              Fin de la sesión remota
Connection closed by foreing host.         Fin de la conexión
atl>                                       Sesión local
```

V.4 Servicios con telnet

Dentro de la organización de la Internet existe un organismo que se encarga de muchas de las tareas de administración de la red, así como de distribuir información sobre ella. Este organismo, llamado **Network Information Center (Centro de Información de la Red, NIC)**, presta como uno de sus servicios el acceso a un tutorial sobre el uso de la Internet. Este tutorial puede ser usado a través de telnet con el siguiente comando:

```
% telnet nic.ddn.mil
```

Un servicio aún más importante para la Universidad es el acceso a la supercomputadora CRAY YMP, ofrecido por la Dirección General de Cómputo Académico. La supercomputadora CRAY tiene como nombre *sirio.cray.unam.mx* y como dirección *132.248.205.1*, por lo que puede ser alcanzada como se muestra a continuación:

```
% telnet sirio

Trying 132.248.205.1...
Connected to sirio.
Escape character is '^]'.


Cray UNICOS (sirio) (ttyp000)


login: cecafi
...

sirio>
...
sirio> logout

%
```

Los siguientes capítulos se ocupan de las utilerías de transferecia de archivos y correo electrónico.

VI. TCP/IP: Transferencia de archivos

La transferencia de archivos permite obtener o hacer copias de archivos hacia un nodo remoto de la red. A través de este servicio, la Internet permite el acceso gratuito a bases de datos que contienen software de dominio público e información sobre diversos temas.

Las siguientes secciones describen el uso de **ftp**, la utilería de transferencia de archivos que proporcionan la mayoría de las implementaciones de TCP/IP.

VI.1 Ftp

El comando **ftp** permite al usuario hacer copias de archivos hacia o desde cualquier nodo conectado a la red. Las aplicaciones típicas de este comando incluyen: el acceso a archivos almacenados en computadoras centrales desde computadoras personales, el acceso a bases de datos públicas y la distribución de información a través de una red.

El funcionamiento de ftp se basa en un conjunto de comandos que permiten el acceso a la información de una máquina remota, así como la configuración de la misma utilería. El primer paso en una sesión de ftp es establecer una conexión con el nodo remoto (para esto es necesario conocer una cuenta en el sistema remoto con permiso de acceso a la información que se desea accesar):

Conexión con Kelem

```
atl> ftp kelem
Connected to kelem.
220 kelem.cecafi.unam.mx FTP server (Version 4.1 Sun Mar 25
22:59:11 EST 1990) ready.
```

```
Name: nam                                   Cuenta en el sistema remoto
331 Password required for nam.
Password:                                   El password no se despliega
230 User nam logged in.
ftp>                                        Ftp está listo para recibir comandos
```

La misma conexión se puede hacer también de la siguiente manera:

```
atl> ftp
ftp> open kelem                             El comando open inicia la conexión
Connected to kelem.
220 kelem.cecafi.unam.mx FTP server (Version 4.1 Sun Mar 25
22:59:11 EST 1990) ready.
Name: nam
331 Password required for nam.
230 User nam logged in.
ftp>
```

Una vez que ftp despliega el prompt *'ftp>'*, se puede utilizar cualquiera de los comandos descritos en las siguientes secciones.

VI.2 Manipulación de directorios

Ftp cuenta con varios comandos que permiten navegar por el sistema de archivos del nodo remoto; todos estos comandos son muy semejantes a los del *shell* de Unix y se ilustran en la tabla 6.1.

Comando	Función
pwd	Despliega el directorio de trabajo
cd *dir*	Cambia el directorio de trabajo a *dir*
mkdir *dir*	Crea el directorio especificado
rmdir *dir*	Borra el directorio especificado
ls	Despliega el contenido del directorio actual

Tabla 6.1 Comandos de manipulación de directorios

La siguiente secuencia ilustra el uso de esos comandos:

```
ftp> pwd                                    Directorio de trabajo en el nodo remoto
257 "/usr/users/nam" is current directory.
```

```
ftp> cd letras/dire.straits                          Pasar a algún directorio
250 CWD command successful.
ftp> ls
200 PORT command successful.
150 Opening data connection for /bin/ls (132.248.54.3,1046)
(0 bytes).
-rw-r--r--  1 nam  cecafi   8976 Jun 30  1991 brothers.arms
-rw-r--r--  1 nam  cecafi   8016 Jun 30  1991 communique
-rw-r--r--  1 nam  cecafi   7907 Jun 30  1991 dire.straits
-rw-r--r--  1 nam  cecafi   8160 Jun 30  1991 love.over.gold
-rw-r--r--  1 nam  cecafi  12074 Jun 30  1991 making.movies
226 Transfer complete.
ftp>
```

VI.3 Manipulación de archivos

De manera similar, ftp proporciona varios comandos para manipular los archivos del nodo remoto, tal como muestra la tabla 6.2.

Comando	Función
delete *archivo*	Borra el *archivo especificado*
rename *n1 n2*	Cambia el nombre de *n1* a *n2*
chmod *perm archivo*	Cambia los permisos del *archivo*

Tabla 6.2 Comandos de manipulación de archivos

VI.4 Transferencia de archivos

Los comandos *get/put* permiten copiar un archivo desde/hacia cualquier sistema remoto. La sintaxis de dichos comandos es la siguiente:

```
ftp> get archivo-remoto [archivo-local]
ftp> put archivo-local [archivo-remoto]
```

Cuando no se especifica el segundo argumento, ftp asume que el nombre de los archivos remoto y local es el mismo. A continuación se ejemplifica el uso de estos comandos:

```
ftp> get love.over.gold                    Copia el archivo a la máquina local
200 PORT command successful.
150 Opening data connection for love.over.gold
(132.248.54.3,1060) (8160 bytes).
226 Transfer complete.
8388 bytes received in 0.18 seconds (46 Kbytes/s)
ftp> put one.world                         Copia el archivo a la máquina remota
200 PORT command successful.
150 Opening data connection for one.world (132.248.54.3,1062).
226 Transfer complete.
500 bytes sent in 0.16 seconds (45 Kbytes/s)
ftp> ls                          Ahora, hay una copia de one.world en el nodo remoto...
200 PORT command successful.
150 Opening data connection for /bin/ls (132.248.54.3,1064)
(0 bytes).
-rw-r--r--  1 nam  cecafi   8976 Jun 30  1991 brothers.arms
-rw-r--r--  1 nam  cecafi   8016 Jun 30  1991 communique
-rw-r--r--  1 nam  cecafi   7907 Jun 30  1991 dire.straits
-rw-r--r--  1 nam  cecafi   8160 Jun 30  1991 love.over.gold
-rw-r--r--  1 nam  cecafi  12074 Jun 30  1991 making.movies
-rw-r--r--  1 nam  cecafi    742 Jan 15  1992 one.world
226 Transfer complete.
ftp> !ls -l love.over.gold          ...y una copia de love.over.gold en el nodo local
-rw-r--r--   1 nam group    8160 Jan 15  1992 love.over.gold
ftp>
```

El caracter **!** permite ejecutar un comando del sistema operativo en el nodo local.

VI.5 Transferencias múltiples

Aunque los comandos get y put proporcionan un gran potencial a ftp, en algunas ocasiones (por ejemplo cuando se necesita transferir un conjunto numeroso de archivos) pueden resultar poco prácticos. Los comandos *mget* y *mput* permiten el uso de las expresiones regulares del shell de Unix para especificar sus argumentos; la tabla 6.3 muestra los metacaracteres permitidos y su función:

Metacaracter	Función
*	Sustituye a cualquier cadena de caracteres
?	Sustituye a cualquier caracter
$[c_1c_2...c_n]$	Sustituye a cualquier caracter entre $c_1, c_2, \ldots, c_n$

Tabla 6.3 Metacaracteres válidos en ftp

Enseguida se ejemplifica el uso de estos comandos:

```
ftp> cd /usr/lib/DXM/demos/motifgif          Una demostración de Motif
250 CWD command successful.
ftp> ls                                      Muestra los archivos
200 PORT command successful.
150 Opening data connection for /bin/ls (132.248.54.3,1077)
(0 bytes).
-rwxrwxr-x  1 root  system  136030 Nov 13  1990 challenger.gif
-rwxr-xr-x  1 root  system  1996604 Dec 19 15:44 motifgif
-rwxrwxr-x  1 root  system    10695 Nov 13  1990 motifgif.c
-rwxrwxr-x  1 root  system     5656 Nov 13  1990 motifgif.h
-rw-r--r--  1 root  system    42780 Dec 19 15:43 motifgif.o
226 Transfer complete.
ftp> mget *.[ch]                  Solo copia los fuentes de la demostración
mget motifgif.c? y                Se debe confirmar la transferencia
200 PORT command successful.
150 Opening data connection for motifgif.c (132.248.54.3,1081)
(10695 bytes).
226 Transfer complete.
11106 bytes received in 0.3 seconds (36 Kbytes/s)
mget motifgif.h? y
200 PORT command successful.
150 Opening data connection for motifgif.h (132.248.54.3,1083)
(5656 bytes).
226 Transfer complete.
5870 bytes received in 0.13 seconds (45 Kbytes/s)
ftp> !ls                                     Revisa la copia
motifgif.c
motifgif.h
ftp>
```

Como puede observarse del ejemplo, cuando se utiliza alguno de estos comandos, es necesario confirmar la transferencia de cada uno de los archivos. El comando **prompt** permite que las transferencias se realicen sin necesidad de confirmación.

Por otro lado, para poder utilizar los metacaracteres es necesario que ftp reconozca expresiones regulares. El comando **glob** habilita/deshabilita esta característica (por default se encuentra habilitada).

VI.6 Formatos de transferencia

Por default, ftp únicamente permite la transferencia de archivos de texto (caracteres ASCII 0-127); cuando se desea transferir archivos binarios (por ejemplo archivos ejecutables o textos en ascii extendido) se debe utilizar antes el comando **binary**. El comando **ascii** regresa a ftp a modo ASCII.

VI.7 Redireccionamiento de entrada/salida

El nombre de cualquier archivo que se especifique en ftp puede ser sustituido por el caracter -; en este caso, el archivo que se utiliza es la entrada estándar (*stdin*) o la salida estándar (*stdout*), dependiendo de si el nombre sustituido representaba un archivo de entrada o de salida.

Un ejemplo común del empleo de esta característica es desplegar un archivo remoto sin tener que copiarlo a la máquina local:

```
ftp> cd letras/pink.floyd
250 CWD command successful.
ftp> get goodbye.blue.sky -                    Utiliza la salida estándar
200 PORT command successful.
150 Opening data connection for goodbye.blue.sky
(132.248.54.3,1101) (418 bytes).
Goodbye Blue Sky
------- ---- ---
Oooooooooooooooooh
Did you see the frightened ones
Did you hear the falling bombs
Did you ever wonder
Why we had to run for shelter
When the promise of a brave new world
Unfouled beneath a clear blue sky
Oooooooooooooooooh
Did you see the frightened ones
Did you hear the falling bombs
The flames are all long gone
But the pain lingers on
Goodbye Blue Sky
Goodbye Blue Sky
Goodbye
226 Transfer complete.
436 bytes received in 0.11 seconds (3.8 Kbytes/s)
ftp>
```

Por otro lado, si el nombre de un archivo inicia con el caracter |, se interpreta como un comando de Unix y la salida de la transferencia se redirecciona a la entrada de ese comando, es decir se hace un *pipe*. Si el archivo del ejemplo anterior hubiese resultado más largo, el comando

```
ftp> get goodbye.blue.sky |more
```

obtendría mejores resultados (el comando *more* de Unix despliega un archivo página por página).

VI.8 Ftp anónimo

Muchos de los nodos de la Internet distribuyen de manera gratuita información de dominio público basándose en ftp. Para permitir el acceso a su información, cada uno de esos nodos proporciona una cuenta pública llamada **anonymous** que generalmente tiene como password **guest** o **ident** (algunos nodos aceptan cualquier password, mientras que otros solicitan la dirección de correo electrónico del usuario[1]). Este método de acceso es llamado **ftp anónimo**.

La información que se distribuye de esta manera incluye temas como compiladores, ambientes gráficos, virus, juegos, música, documentación de la red (RFCs), literatura, imágenes digitalizadas, etc. Un reporte hecho en septiembre de 1991 estimaba el total de código fuente distribuido en la Internet en 1,500 Megabytes. La siguiente tabla 6.4 muestra algunos nodos y la información que distribuyen.

Los administradores de esos sistemas han solicitado a la comunidad de la Internet que las sesiones de ftp anónimo sólo se efectúen cuando la carga de la red sea baja, aproximadamente de 7 PM a 6 AM (hora local del sistema que se accesa). Por favor, respeta esta petición.

La lista completa (**ftp-list**) puede ser obtenida, entre otros, de los nodos

```
enh.nist.gov (129.6.16.1)
pine.circa.ufl.edu (128.227.8.7)
```

[1] El correo electrónico es el tema del siguiente capítulo.

En el Centro de Cálculo se mantiene una copia de este archivo en el directorio **/pub** de Atl, por lo que sus usuarios pueden analizar la lista con el comando

```
atl> more /pub/ftp-list
```

Nodo	Descripción
ads.com	Listas de correo
aeneas.mit.edu	Kerberos (autentificación de redes)
aisun1.ai.uga.edu	Utilerías para MS-DOS
allspice.lcs.mit.edu	SNMP (administración de redes)
argus.standford.edu	Información de la Internet
arisia.xerox.com	Código fuente de TCP/IP
chalmers.se	RFCs (documentación de la Internet)
cs.toronto.edu	Aplicaciones X (aplicaciones gráficas de red)
cs.uwp.edu	Música clásica y popular
ento.tamu.edu	Juegos para VAX/VMS
expo.lcs.mit.edu	Código fuente de X (sistema de ventanas para red)
nnsc.nsf.net	Guía de recursos de la Internet
nic.ddn.mil	RFCs
pine.circa.ufl.edu	Información sobre el "gusano" de la Internet
sh.cs.net	Mapas de la red

Tabla 6.4. Nodos con servicio de ftp anónimo

Muchos de los archivos que se mantienen en estas máquinas se encuentran en formato comprimido o empacados en formato de cinta (formato **tar**). En el primero de los casos (el archivo tiene extensión **Z**), el comando **uncompress** realiza la decompresión; en el segundo caso (el archivo tiene extensión **tar**) se debe utilizar el comando **tar xvf**. Por ejemplo:

```
atl> ls
bind.4.8.tar.Z
atl> uncompress bind.4.8.tar.Z
atl> ls
bind.4.8.tar
atl> tar xvf bind.4.8.tar
...
```

Para finalizar esta sección, el siguiente ejemplo muestra una sesión de ftp anónimo:

```
atl> ftp chalmers.se
Connected to 129.16.1.1.
220 chalmers FTP server (Version 5.2 Mon Sep 10 20:37:06 MET
ST 1990) ready.
Name (129.16.1.1:nam): anonymous                    Ftp anónimo
331 Guest login ok, send ident as password.
Password:                                           El password es ident
230 Guest login ok, access restrictions apply.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> cd rfc
250 CWD command successful.
ftp> get rfc1118.txt                    El RFC 1118 es una guía a la Internet
200 PORT command successful.
150 Opening BINARY mode data connection for rfc1118.txt
(61446 bytes).
226 Transfer complete.
61446 bytes received in 32 seconds (1.8 Kbytes/s)
ftp> quit                                           Salir de ftp
atl>
```

VII. TCP/IP: Correo electrónico

El servicio de correo electrónico permite que dos usuarios puedan intercambiar mensajes a través de la red sin importar la localización de los nodos que utilizan para ello. La conexión de la RedCECAFI con la Internet permite que la comunidad de la Facultad de Ingeniería pueda comunicarse mediante correo electrónico a los organismos educativos más importantes de todo el mundo, además de algunas empresas gubernamentales y comerciales de diferentes países.

Este capítulo presenta una pequeña introducción al uso de mail, la utilería de correo electrónico de Unix.

VII.1 MAIL

Bajo Unix, el correo electrónico es proporcionado por la utilería **mail**, la cual se invoca de la siguiente manera:

% mail *dirección*

en donde *dirección* tiene la forma

usuario

si la persona a la que se desea enviar el mensaje se encuentra en el mismo nodo o[1]

usuario@nodo

si esa persona se encuentra en otro nodo.

Una vez que se ha invocado, mail pide al usuario el propósito

[1]El nodo debe especificarse con un nombre.

(*Subject*) del mensaje y el texto del mismo. Al terminar de escribirlo (lo cual se indica con la secuencia ^*D*) mail pregunta si se desea enviar una copia del mensaje a otro usuario (*Cc*), tal como se ilustra a continuación:

```
atl> mail jramirez                                  Mensaje para jramirez
Subject: Curso Oracle                               Asunto del mensaje

        Por favor confirma mi asistencia al         Texto del mensaje
    curso de Oracle V6.  Gracias.
^D
Cc: aamador                                         Se hará una copia a aamador
atl>
```

La siguiente vez que el usuario *jramirez* use su cuenta, el sistema operativo se encargará de desplegar el letrero

You have mail

para indicarle que ha recibido un nuevo mensaje de correo. Los mensajes recibidos pueden leerse usando el comando mail sin argumentos:

```
ULTRIX V4.2 (Rev. 96) (atl.cecafi.unam.mx)
login: jramirez                                     Jramirez entra a su cuenta
Password:
Last login: Thu Jan 9 13:30
ULTRIX V4.2 (Rev. 96) System #2: Mon Dec 9 18:54:44 CST 1991
...
You have mail                                       Hay mensajes nuevos

atl> mail                                           Mail sin argumentos permite leer los mensajes
Mail (version 3.2)  Type ? for help.
>N  1 nam    Tue Jan 28 14:57   11/300   Curso Oracle
& t                                                 Leer los mensajes recibidos
Message  1:
From nam Tue Jan 28 14:57:15 1992
From: nam@atl (Norberto Arrieta M.)                 Remitente
To: jramirez                                        Destinatario
Subject: Curso                                      Asunto
Date: Tue, 28 Jan 92 14:57:12 CST
Message-ID:  <9201281457.aa00560@aztlan.UUCP>
Status: RO

     Por favor confirma mi asistencia al curso de
     Oracle V6.
```

```
& q                                          Terminar la sesión de mail
atl>
```

Las dos secciones siguientes se encargan de describir con mayor detalle la forma de enviar y recibir mensajes con mail.

VII.2 Enviando mensajes

Como se mencionó en la sección anterior, una carta se envía utilizando el comando

% mail *dirección*

para después especificar el asunto y el texto del mensaje. Cuando el usuario de mail escribe el texto del mensaje, puede hacer uso de una serie de comandos que lo auxilian en su tarea; todos esos comandos inician con el caracter '~', por lo que son llamados **comandos tilde**, y deben escribirse en la primer columna de la pantalla. La siguiente tabla 7.1 describe los más relevantes:

Comando	Función
~c *usuarios*	Envía copias a los *usuarios* especificados
~s *cadena*	Usa *cadena* como asunto (subject) del mensaje
~r *archivo*	Incluye el *archivo* en el mensaje
~m *mensajes*	Incluye en el texto los *mensajes* especificados
~q,~Q	Sale de *mail*; salva el texto en *dead.letter*
~x	Sale de *mail*, no salva el texto
~d	Incluye en el mensaje al archivo *dead.letter*
~v	Edita el mensaje con *vi*
~w *archivo*	Escribe el mensaje en el archivo *especificado*
~!*command*	Ejecuta un comando del shell sin salir de mail
~?	Despliega la lista completa de comandos tilde

Tabla 7.1 Comandos tilde

A manera de ejemplo, a continuación se muestra una forma equivalente de enviar el mensaje que se usó en la sección anterior:

```
atl> cat mensaje                    El mensaje se encuentra en un archivo

     Por favor confirma mi asistencia al curso
de Oracle V6.  Gracias.
```

```
atl> mail jramirez
Subject:                                    El usuario olvida dar el asunto
~s Curso                                    El asunto se da con el comando ~s
~!ls                                        Listar los archivos del directorio
mensaje
tarea.c
tarea
~r mensaje                                  Toma el texto de la carta del archivo mensaje
~c aamador                                  Se hará una copia a aamador
^D
Cc:                                         Ya se especificó quienes reciben copias
atl>
```

El archivo *dead.letter* al que se refieren algunos de los comandos de la tabla 7.1 es generado por mail cuando por alguna razón se interrumpe el proceso en el momento en que se está escribiendo una carta. La especificación de los mensajes sobre los que actúa el comando *~m* se hace con el número de identificación de los mismos (la siguiente sección define los números de identificación).

VII.5 Leyendo el correo

Cuando un usuario recibe un mensaje nuevo por medio de mail, el sistema operativo se encarga de notificarselo la siguiente vez que se hace uso de su cuenta con el mensaje

```
You have mail
```

Si el usuario se encuentra en sesión cuando recibe el mensaje, el sistema operativo no dará aviso de ello a menos que se haya usado el comando

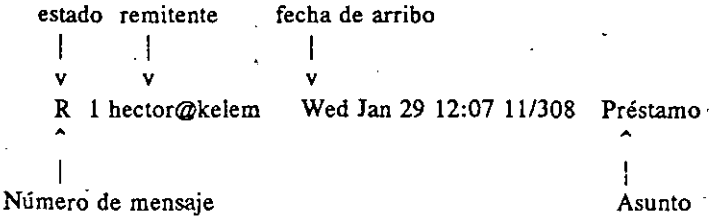
```
atl> biff y
```

Si éste es el caso, la notificación se hace en cuanto termina el comando que se esté ejecutando en el momento en que se recibe el mensaje.

Un usuario puede leer el correo que ha recibido utilizando el comando mail sin argumentos. Cuando se usa de esta manera, mail despliega la lista de mensajes recibidos y el prompt '&', indicando que está listo para recibir comandos:

```
atl> mail                                                    Leer el correo
Mail (version 3.2)  Type ? for help.
"/usr/spool/mail/nam": 3 messages
 N  3 aamador          Wed Jan 29 12:08 11/308   C++
>U  2 jramirez         Wed Jan 29 12:08 10/290   Re: Curso
 R  1 hector@kelem     Wed Jan 29 12:07 11/308   Préstamo
&
```

La lista de mensajes desplegada por mail incluye los siguientes campos:



El estado de un mensaje puede tomar tres valores:

N - El mensaje acaba de ser recibido
U - El mensaje fue recibido anteriormente, pero no ha sido leído
R - El mensaje ya fue leído

El caracter '>', llamado cursor, indica el mensaje actual (el que será leído enseguida). Los comandos más relevantes de mail se muestran en la tabla 7.2.

Observe que la lista de mensajes es opcional; cuando no se especifica, sólo se incluye al mensaje actual (el apuntado por el cursor). Una lista de mensajes puede especificarse como un rango (por ejemplo 1-5 para los primeros cinco mensajes) o con un asterisco (todos los mensajes). La mayoría de los comandos pueden abreviarse con su primer letra.

Comando	Función
type [*lista*]	Despliega los mensajes de la *lista*
<*Return*>	Despliega el mensaje actual
header	Despliega los encabezados de los mensajes
next	Se salta el mensaje actual (avanza el cursor)
vi [*lista*]	Edita los mensajes de *lista*
delete [*lista*]	Borra los mensajes especificados
undelete [*lista*]	Recupera los mensaje borrados
write [*lista*] *a*	Graba los mensajes en el archivo *a*
reply [*lista*]	Responde a los mensajes especificados
mail *usuario*	Envía correo al *usuario* especificado
quit	Sale de mail; salva los mensajes no leídos
exit	Sale de mail; salva todos los mensajes
!*comando*	Ejecuta el *comando* de shell especificado
?	Despliega la lista de comandos completa

Tabla 7.2 Comandos de mail

A continuación se muestran algunos ejemplos del uso de estos comandos:

```
& header                                                    Desplegar los encabezados
 N  3 aamador        Wed Jan 29 12:08 11/308   C++
>U  2 jramirez       Wed Jan 29 12:08 10/290   Re: Curso
 R  1 hector@kelem   Wed Jan 29 12:07 11/308   Préstamo
& type 1                                                    Leer el primer mensaje
Message  1:
From hector@kelem Wed Jan 29 12:07:53 1992
From: hector@kelem (Hector Rojas Sosa)                      Remitente
To: nam                                                     Destinatario
Subject: Libro Compiladores                                 Asunto
Date: Wed, 29 Jan 92 12:07:52 CST                           Fecha de arribo
Status: U                                                   Estado
                                                            Texto
     Guarde el libro de compiladores en tu escritorio.
     Gracias.
& d 1-2                                                     Borrar los mensajes 1 y 2
& w 3 cpp                                                   Graba el mensaje 3 en el archivo cpp
& quit                                                      Sale de mail
```

VII.5 Listas de correo

La Internet proporciona un servicio de distribución de correo electrónico a nivel mundial. Con él es posible mantener foros de discusión sobre diversos temas, hacer consultas relacionadas con esos temas o proporcionar acceso a diferentes boletines electrónicos (BBS, Bulletin Board Service).

Este servicio de correo electrónico se basa en las llamadas **listas de correo**, que consisten en grupos de usuarios que desean intercambiar correo relacionado con un tema específico. Existen cientos de listas de correo en todo el mundo, que incluyen temas como música, literatura, bases de datos, programación orientada a objetos, deportes, sistemas operativos y muchos otros; cualquier usuario de la Internet puede unirse a ellas de manera gratuita.

En Atl se mantiene un archivo (*/pub/interest-groups*) que contiene una relación de las listas de correo de la Internet y otras redes, además de una pequeña descripción de cada una de ellas. Esa descripción incluye la manera de suscribirse a cada lista. Como ejemplo de esto, enseguida se muestra la manera de suscribirse a la lista de correo para C++:

```
atl> vi /pub/interest-groups          Consultar la información de las listas
"/pub/interest-groups" [Read Only] 18730 lines 800407 chars
/C++                                  Buscar la cadena C++
TCPLUS-L%UCF1VM.BITNET@CUNYVM.CUNY.EDU   Texto encontrado por vi
TCPLUS-L on LISTSERV@UCF1VM
TCPLUS-L is a new list intended for discussion of the
Borland product Turbo C++.
To join the list, please send the following command to
LISTSERV@UCF1VM (Internet users send mail only, to
LISTSERV%UCF1VM.BITNET@CUNYVM.CUNY.EDU) either via an
interactive message or as the body of a piece of mail:

   SUBSCRIBE TCPLUS-L <firstname> <lastname>

Please substitute in your own first and last names for
<firstname> and <lastname>.

For information on the content of the list and other
non-technical matters, please contact Don Cross at
CROSS@UCF1VM. For technical help or assistance in joining
the list, please contact Lois Buwalda at LOIS@UCF1VM.
:q!                                   Salir de vi
```

La descripción de la lista indica que la suscripción se tramita enviando el mensaje

SUBSCRIBE TCPLUS-L <nombre> <apellido>

a la dirección LISTSERV%UCF1VM.BITNET@CUNYVM.CUNY.EDU. Los mensajes dirigidos a la lista se envían a la dirección TCPLUS-L%UCF1VM.BITNET@CUNYVM.CUNY.EDU.

```
atl> mail listserv%ucf1vm.bitnet@cunyvm.cuny.edu      Suscripción
Subject: Subscribe                  No importa; puede ser cualquier cadena
SUBSCRIBE TCPLUS-L Norberto Arrieta
^D
Cc:
```

Una vez que el administrador de la lista acepta la solicitud de suscripción, la realiza y contesta al usuario con un mensaje de bienvenida. Al recibir este mensaje, se puede empezar a hacer uso de la lista:

```
atl> mail                                   Leer el correo recibido
...
&                                           <Return> (leer mensaje)
Received: from UCF1VM.CC.UCF.EDU by ucf1vm.cc.ucf.edu (IBM
VM SMTP V2R1) with BSMTP id 1630; Tue, 10 Dec 91 18:12:30

Received: by UCF1VM (Mailer R2.08) id 8609; Tue, 10 Dec 91
18:12:29 EST
Date:         Tue, 10 Dec 1991 18:12:28 -0500
From:    Revised    List    Processor    (1.7a)
<LISTSERV%UCF1VM.BITNET@ucf1vm.cc.ucf.edu>
Subject:      Your subscription to list TCPLUS-L
To: NORBERTO ARRIETA <nam@ATL.CECAFI.UNAM.MX>

Dear networker,                             La solicitud fue aceptada

  Your subscription    to list    TCPLUS-L  (TURBO C++
Discussion group.)  has been accepted.

...
& quit
...
atl> mail tcplus-l%ucf1vm.bitnet@cunyvm.cuny.edu
...                                         Enviar algún mensaje a la lista
atl>
```

VII.6 Gateways

Como se mencionó anteriormente la Internet cuenta con conexiones a las redes de mayor importancia en el mundo, lo que permite a sus usuarios intercambiar correo electrónico con usuarios fuera de ella.

Las conexiones se realizan utilizando computadoras conocidas como **gateways** que se encargan de tomar todos los mensajes de la Internet que estén dirigidos a destinos fuera de ella y de enviarlos a la red correcta.

Cuando se desea enviar correo electrónico a un destino fuera de la Internet, únicamente es necesario especificar la dirección de un gateway que permita la conexión con la red en la que se encuentra el destino. La tabla 7.3 muestra algunos de los gateways más importantes y la forma de especificarlos en la dirección del destinatario:

Red	Formato de la dirección
ACSNET	*usuario*@*nodo*.oz.au
BITNET	*usuario*%*nodo*.bitnet@cunyvm.cuny.edu
EASYNET	*nodo*::*usuario*@decwrl.dec.com
JUNET	*usuario*%*nodo*.junet%utokyo-relay@relay.cs.net
NASAMAIL	*usuario*@nasamail.nasa.gov
SPAN	*usuario*@span.nasa.gov
UUCP	*usuario*%*nodo*.uucp@uunet.uu.net

Tabla 7.3 Gateways para comunicación fuera de la Internet

Así, por ejemplo, un usuario de la Internet puede enviar correo electrónico al usuario *rgallardo* conectado al nodo *mxo* de la *EasyNET* de la siguiente manera:

```
atl> mail mxo::rgallardo@decwrl.dec.com
```

El proceso inverso (enviar un mensaje de alguna red hacia la Internet) también es posible, usando los formatos que se muestran en la tabla 7.4.

Red	Formato de la dirección
ACSNET	*usuario*%*nodo.dominio*@munnarloz
BITNET	*usuario*@*nodo.dominio*
EASYNET	decwrl::*usuario*@*dominio*
JUNET	*usuario*@*nodo.dominio*.arpa
NASAMAIL	*usuario*@*nodo*
SPAN	ames::*usuario*@*dominio*
UUCP	uunet!*nodo.dominio*!*usuario*

Tabla 7.4 Gateways para comunicación con la Internet

Si ahora un usuario de la EasyNET quisiera enviar un mensaje al usuario *nam* del nodo *atl* (recordemos que los nodos del CECAFI pertenecen al dominio *cecafi.unam.mx*), lo haría de la siguiente manera:

```
mxo> mail decwrld::nam@atl.cecafi.unam.mx
```

Con la descripción de mail terminan las utilerías básicas de TCP/IP. El siguiente capítulo dicute algunos otros comandos que pueden resultar también útiles al usuario de la red.

VIII. TCP/IP: Otros comandos

Los comandos que se describieron en los capítulos anteriores (telnet, ftp y mail) representan los tres servicios básicos de TCP/IP, por lo que la mayoría de las implementaciones de este protocolo incluyen a esas tres utilerías.

Sin embargo, muchas implementaciones proporcionan también otra serie de utilerías que facilitan el uso de la red, tales como *talk*, *finger*, *rsh*, *rcp*, *rlogin*, *netstat*, *NFS*, *Yellow Pages* y *Bind*. En este capítulo se describen algunos de los comandos que resultan más útiles para el usuario común y corriente de Unix.

VIII.1 Comunicación interactiva entre usuarios

Aunque mail permite que dos usuarios puedan establecer comunicación entre sí mediante correo electrónico, en algunas ocasiones es necesario entablar la comunicación de manera interactiva (en modo conversacional).

El comando *talk* permite realizar este tipo de comunicaciones. Para esto, los dos usuarios que desean establecer la conversación deben estar en sesión al mismo tiempo y uno de ellos debe iniciar la comunicación con el comando

atl> **talk jramirez**

el cual limpia el contenido de la pantalla, la divide a la mitad con una serie de guiones y despliega el mensaje

Waiting for your party to respond

El destinatario de talk verá desplegarse en su terminal el mensaje

```
Message from Talk_Daemon@atl.cecafi.unam.mx at 20:10 ...
talk: connection requested by nam@atl
talk: respond with: talk nam@atl
```

y deberá contestar de la siguiente manera:

```
atl> talk nam
```

Una vez que esto se hace, se establece una conexión entre los dos usuarios y cada carácter que se capture se desplegará en las terminales de los dos usuarios. La conexión debe terminarse con la secuencia *^C*.

Talk permite que los dos usuarios estén localizados en máquinas diferentes, en cuyo caso deberá emplearse la sintaxis

```
% talk usuario@nodo
```

Recuerde que para usar talk, los dos usuarios deben estar en sesión. Para revisar que esta condición se cumpla, se pueden usar los comandos *who* (que despliega los usuarios conectados a la máquina local) y *rwho* (que despliega los usuarios conectados a máquinas en la red local):

```
atl> who
acf      console Jan 29 19:25
aaron    tty00   Jan 27 21:32 (SERV03)
ara00    tty01   Jan 28 16:18 (SERV04)
acf      tty02   Jan 29 14:35 (SERV04)
nam      ttyp0   Jan 29 20:39 (aztlan)
atl> rwho
aaron    atl.cecafi.unam.mx:tty00    Jan 27 21:32 :48
acf      atl.cecafi.unam.mx:console  Jan 29 19:25 :03
nam      aztlan.cecafi.unam.mx:tty01 Jan 29 17:30 :40
moa00    aztlan.cecafi.unam.mx:tty02 Jan 29 19:00 :01
...
```

El comando *finger* es semejante a who, pero proporciona algunos campos extras:

```
atl> finger
Login       Name                TTY Idle    When    Office
luigi       luis felipe rivera a co   39 Sat 18:53  Soporte
yolanda     yolanda b. perez gut 01 5:18 Sat 14:09  DIPASG
aaron       aaron arcos tapia    05 9:50 Thu 16:13  DIPASG
nam         norberto arrieta mar p0       Sat 16:01  Soporte

atl>
```

El comando **finger** permite también obtener la información de un usuario en particular, aún si se encuentra en otra máquina:

```
atl> finger nam@aztlan
[aztlan]
Login name: nam(messages off) In real life: Norberto Arrieta
Office:  Soporte Te,  ext.5734
Directory: /usr/nam                          Shell: /bin/ksh
On since Feb  8 15:40:46 on tty01
No Plan.
atl>
```

VIII.2 Ejecución de comandos en nodos remotos

Unix proporciona una utilería muy parecida a telnet llamada *rlogin*; sin embargo esta utilería sólo puede ser usada para establecer sesiones en sistemas Unix, a diferencia de telnet, que puede ser usada aún en ambientes heterogéneos.

Una característica de rlogin que no posee telnet es que una persona puede autorizar a un grupo de usuarios de otras máquinas a utilizar su cuenta sin proporcionar password. Para lograrlo, se debe crear el archivo *.rhosts* en el directorio raíz de la cuenta (*home*); cada renglón de este archivo contiene los nombres del nodo y del usuario autorizados para usar la cuenta.

Por ejemplo, supongamos que el usuario *nam* tiene cuentas en los sistemas Atl y Kelem y desea conectarse del primero hacia el segundo sin proporcionar password. Entonces, deberá crear el archivo de autorización en Atl:

```
atl> pwd
/usr/users/nam                          El archivo se debe crear en el home
atl> cat > .rhosts
kelem      nam                  Autorización para el usuario nam en el sistema kelem
...

kelem> rlogin atl                       No es necesario dar password
Last login: Thu Jan 9 13:30
ULTRIX V4.2 (Rev. 96) System #2: Mon Dec 9 18:54:44 CST 1991
...
atl> ^D                                 Fin de la sesión
kelem>
```

Un comando asociado con rlogin es *rsh*[1], que permite ejecutar un sólo comando en un nodo remoto. La sintaxis de ese comando es

```
% rsh nodo comando
```

[1]En algunos sistemas este comando es llamado *rcmd*.

El usuario de rsh debe tener autorización para ejecutar comandos en el sistema remoto (esa autorización se especifica, de igual manera que para rlogin, en el archivo .rhosts). El siguiente ejemplo muestra como obtener un listado del directorio */etc* del nodo Kelem:

```
atl> rsh kelem ls /etc
```

Es importante notar que el redireccionamiento de entrada/salida se hace en el nodo local a menos que el metacaracter de redireccionamiento se "escape" del shell:

```
atl> rsh kelem ls /etc > listado        "listado" se crea en la máquina local

atl> rsh kelem ls /etc '>' listado      ahora se crea en la máquina remota
```

Un uso muy común de rsh es ejecutar una aplicación de X-Windows (u otro ambiente de ventanas) en un nodo remoto, pero desplegando las ventanas en el nodo local:

```
atl> rsh kelem /usr/bin/X11/xterm -ls -display atl:0:0
```

VIII.3 Transferencia de archivos

El comando *rcp* presta un servicio muy parecido al de ftp, pues permite copiar archivos a través de la red, aunque su interface es mucho más sencilla. Rcp sigue la sintaxis

```
% rcp nodoFuente:archivoFuente nodoDestino:archivoDestino
```

Si alguno de los nodos no se especifica, se asume el nodo local:

```
atl> rcp aztlan:/etc/passwd /tmp     Hace una copia del archivo /etc/passwd
                                     de Aztlan en el directorio /tmp de Atl
atl> rcp tesis/codificacion.c kelem:pruebas/cod.c  Ahora la copia
                                     es del nodo local hacia Kelem
atl> rcp aztlan:mail.txt kelem:mail.txt  Y ahora de Aztlan hacia Kelem
```

Al igual que con rlogin, el usuario de rcp debe tener autorización en el archivo .rhosts de los nodos que desea accesar.

IX. DECNET en VMS

En los capítulos anteriores se describieron algunas de los servicios que ofrecen las implementaciones de TCP/IP para la manipulación de correo electrónico, la transferencia de archivos y la conexión con sistemas remotos; este capítulo y el siguiente tienen como propósito hacer lo mismo para el protocolo DECNET en sus implementaciones para VMS y Ultrix.

Las utilerías de DECNET proporcionan servicios muy similares a los ya descritos; esto, unido al hecho de que la interface de esas utilerías es muy sencilla, facilitarán en gran medida su descripción.

IX.1 Implementaciones de DECNET

Existen implementaciones de DECNET para diversos sistemas operativos; el Centro de Cálculo cuenta con las versiones para VMS, Ultrix y MS-DOS en sus equipos VAX 6210 (Atzin), MicroVAX 3400 (Atl) y DECStation 3100 (Kelem), además de una serie de computadoras personales 286. Aquí sólo se describen las versiones para VMS y Ultrix; la versión para MS-DOS (PATHWORKS) se describe en otro documento.

IX.2 Acceso a archivos

El desarrollo de DECNET se llevó a cabo en un ambiente VMS, por lo que las utilerías de red en este sistema operativo presentan una interface muy intuitiva que facilita en gran medida su uso. Los comandos de red de VMS son los mismos que se utilizan para manipular el sistema de archivos de ese ambiente; de hecho, DECNET únicamente extiende la notación para nombrar los archivos de tal modo que incluya a los diferentes nodos de la red.

```
$ dir atl"nam xxx"::"letras"                    Despliega el contenido del
Directory ATL"nam password"::                   directorio remoto 'letras'

"/usr/users/nam/letras/."
"/usr/users/nam/letras/.."
"/usr/users/nam/letras/american_pie"
"/usr/users/nam/letras/green"
"/usr/users/nam/letras/hyacinth_house"
"/usr/users/nam/letras/white.album"
...
Total of 24 files.
$
```

Esta misma sintaxis es válida también con otros comandos como *append*, *open*, *delete*, *backup*, etc. Observe que la ruta del archivo siempre debe especificarse de acuerdo a las convenciones del sistema operativo remoto (en los ejemplos anteriores, Ultrix) y que se especifica de manera relativa al directorio raíz (*home*) de la cuenta utilizada.

IX.3 Correo electrónico

Aunque las funciones que ofrece el comando mail de VMS son muy semejantes a las que se describieron en el capítulo VII, su interface sí presenta algunas diferencias. Bajo VMS, mail se invoca de la siguiente manera:

```
$ mail
```

entonces, la utilería despliega el prompt *'MAIL>'* y espera algún comando. La tabla 9.1 muestra algunos de los comandos más importantes de mail.

Enseguida se muestra como ejemplo una sesión de mail (observe que la dirección del destinatario sigue el formato *nodo*::*usuario*):

```
$ mail
MAIL> send                                      Enviar mensaje
To:       atl::jramirez                         Destinatario
Subj:     Curso Oracle                          Asunto
Enter your message below. Press CTRL/Z when complete, or
CTRL/C to quit:
                                                Texto
     Confirma mi asistencia al curso de Oracle V6, por
favor.
^Z
```

Comando	Función
SEND	Enviar mensaje
SEND/EDIT	Enviar mensaje; escribir el texto con el editor
SEND *a*	Enviar mensaje; tomar el texto del archivo *a*
DIRECTORY	Desplegar los encabezados de los mensajes recibidos
<*Return*>	Desplegar el mensaje actual
READ *mensaje*	Desplegar el *mensaje* especificado
REPLY	Responder al mensaje actual
BACK	Desplegar el mensaje anterior
DELETE *mensaje*	Marcar para ser borrados los mensajes en la *lista*
PURGE	Borrar los mensajes marcados
EXTRACT *a*	Grabar el mensaje actual en el archivo *a*
EXIT	Salir de mail
HELP *comando*	Obtener ayuda sobre el *comando* especificado

Tabla 9.1 Algunos comandos de mail VMS

```
MAIL> dir                                   Encabezados de los mensajes recibidos
     # From                    Date          Subject

     1 ATZIN::HECTOR          25-JAN-1992   Libro compiladores
     2 ATZIN::ACF             28-JAN-1992   Tesis
     3 ATZIN::JRAMIREZ        31-JAN-1992   C++

MAIL>                                       <Return> (desplegar mensaje actual)
#1          25-JAN-1992 10:04:28.88         Fecha en que se recibió
From:       ATZIN::HECTOR                   Remitente
To:         NAM                             Destinatario
CC:                                         Quienes recibieron copia
Subj:       Libro compiladores              Asunto

     Guarde el libro de compiladores en tu escritorio.
Gracias.
MAIL> read 3                                Leer el mensaje 3 (saltar el 2)
#3          31-JAN-1992 14:05:50.42
From:       ATZIN::JRAMIREZ
To:         NAM
CC:
Subj:       C++

     Sabes en que fecha empieza el curso de C++?

MAIL> delete 3                              Borrar el mensaje 3

MAIL> exit                                  Salir de mail
$
```

IX.4 Sesiones remotas

Las sesiones remotas en DECNET se establecen con el comando *set host*, de acuerdo a la sintaxis

$ set host *nodo*

Por ejemplo:

```
$ set host kelem
ULTRIX V4.2 (Rev. 96) (kelem.cecafi.unam.mx)        Mensajes de Kelem

login: nam                                  Cuenta con la que se crea la sesión
Password:                                   El password no se despliega
Last login: Thu Jan 9 13:30
ULTRIX V4.2 (Rev. 96) System #2: Mon Dec 9 18:54:44 CST 1991
...

kelem> cd tesis                             Se usan algunos comandos
kelem> vi codificacion.c
...
kelem> logout                               Fin de la sesión
%REM-S-END, control returned to node _ATZIN::
$                                           De nuevo nos encontramos en Atzin
```

Por supuesto, DECNET cuenta con muchas otras facilidades que las que aquí se describieron. El lector interesado puede consultar las referencias que se indican al final de este documento.

X. DECNET en Ultrix

Como se mencionó anteriormente DECNET fue desarrollado en un ambiente VMS. Por ello, la implementación de este protocolo para el ambiente Ultrix no cuenta con todas las características de la primera; sin embargo al nivel de esta descripción, las facilidades en los dos ambientes son muy parecidas; enseguida se describe la interface de DECNET para Ultrix.

X.1 Manipulación de archivos

La interface de DECNET en el ambiente Ultrix no es tan transparente como la de VMS; sin embargo, sus comandos son muy parecidos a los del shell de Unix, lo que facilita su manejo. La tabla 10.1 muestra dichos comandos y su función.

Comando	Función
dcat	Despliega el contenido de un archivo remoto
dcp	Transferencia de archivos
dlogin	Sesión remota
dls	Despliega el contenido de un directorio remoto
drm	Borra un archivo remoto
mail	Correo electrónico

Tabla 10. Comandos de manipulación de archivos de DECNET Ultrix

De manera semejante a su equivalente en VMS, DECNET para Ultrix extiende la notación para la ruta de un archivo de la siguiente manera:

nodo/cuenta/password::*ruta*

en donde *ruta* se escribe siguiendo las convenciones del nodo remoto.

A continuación se muestran ejemplos del uso de DECNET en Ultrix:

```
atl> dcat kelem/nam/yyy::demos/motifgif.c        Despliega el archivo
#ifdef REV_INFO                                  demos/motif.gif almacenado en Kelem
#ifndef lint
static char SCCSID[] = "OSF/Motif: @(#)motifgif.c 1    .    5
90/08/01";
#endif /* lint */
...
atl> dcp atzin/nam/www::'[.redes]m.txt;-1' kelem/nam/yyy::m
     Copia la penúltima versión del archivo '[.redes]m.txt' almacenado en Atzin al nodo kelem
atl> dcp atzin/nam/www::'[.redes]m.txt;-1' m     Ahora la copia
                                                 al nodo local
atl> dls atzin/system/ttt::'dua0:[nam]'          Muestra el contenido del
DUA0:[NAM]C.DIR;1                                directorio remoto 'dua0:[nam]'
DUA0:[NAM]DATOS.DIR;1
DUA0:[NAM]ESTADISTICAS.DIR;1
DUA0:[NAM]FORTRAN.DIR;1
DUA0:[NAM]INFO.;1
DUA0:[NAM]LOGIN.COM;3
DUA0:[NAM]NETSERVER.LOG;8
DUA0:[NAM]REDES.DIR;1
DUA0:[NAM]TMP.DIR;1
atl> drm kelem/nam/yyy::basura.txt        Borra el archivo remoto 'basura.txt'
```

X.2 Correo electrónico

El comando mail de DECNET bajo Ultrix presenta la misma interface que se describió en el capítulo VII[1], por lo que en esta sección únicamente se mostrará una sesión ejemplo de esta utilería. El único cambio de mail reside en la manera de especificar la dirección del destinatario:

nodo::*usuario*

Por ejemplo:

```
atl> mail atzin::jramirez
Subject: Reunión
      La reunión programada para mañana se suspendió.
^D
Cc:
atl>
```

[1]Realmente hay un sólo comando mail que "entiende" varios protocolos, entre ellos TCP/IP y DECNET.

X.3 Sesiones remotas

Por último, el comando para establecer sesiones remotas en DECNET Ultrix es *dlogin*:

```
atl> dlogin atzin                                   Conexión con nodo Atzin

          Centro de Calculo de la Facultad de Ingenieria
                    Sistema VAX/VMS 6210

 Username: NAM                                      Usuario NAM
 Password:

                                                    Mensajes de Atzin
```

C	E	C	A	F	I

```
                    SISTEMA VAX/VMS 6210
                        BIENVENIDO


 Last interactive login on Friday, 31-JAN-1992 17:17
 Last non-interactive login on Friday, 31-JAN-1992 17:16

 $ set def [.notas]                                 Cualquier comando
 $ dir

 Directory DISK$USUARIOS1:[NAM.TMP]

 PASSWD.;3              PASSWD.;2              PASSWD.;1
 Y.COM;3                Y.COM;2                Y.TXT;6

 Total of 6 files.
 ...

 $ lo                                               Terminar la sesión
 NAM          logged out at 31-JAN-1992 17:19:59.57
 dlogin -- session terminated
 atl>                                               De nuevo estamos en Atl
```

De nuevo, el lector interesado puede encontrar referencias para ampliar este tema al final de este texto.

XI. Gateway DECNET-Internet

Cómo se mencionó anteriormente, aunque el Centro de Cálculo cuenta con el protocolo DECNET, su uso se encuentra prácticamente restringido a aplicaciones internas del Centro debido a que la RedUNAM se basa en el protocolo TCP/IP. Sin embargo, esto no quiere decir que los usuarios de un nodo DECNET no puedan hacer uso de los servicios proporcionados por la RedUNAM y la Internet.

Atl y Kelem cuentan con una utilería llamada *Gateway DECNET-Internet* que realiza la conversión de mensajes DECNET a mensajes TCP/IP, permitiendo de esta manera la comunicación entre nodos que cuentan protocolos diferentes (los nodos que proporcionan dicha conversión son llamados *gateways*). Así, los mensajes dirigidos a un nodo que usa un protocolo diferente al del nodo origen, deben dirigirse a un gateway para que éste realice la conversión de protocolos y los envíe a su destino correcto.

Las siguientes secciones describen la manera de utilizar el Gateway DECNET-Internet.

XI.1 Uso del Gateway desde un nodo DECNET

Cualquiera de los comando de DECNET para manipulación de archivos puede emplearse para accesar a un nodo TCP/IP con la siguiente sintaxis:

comando gateway"*usuario@nodo password*"::"*archivo*"

Como ejemplo se muestra la manera de usar el servicio de ftp anónimo de la Internet (Recuerde que los nodos que proporcionan la conversión de protocolos en CECAFI son Atl y Kelem):

```
$ dir atl"anonymous@cs.uwp.edu guest"::"pub/music/classical"
```
Desplegar el directorio de música clásica usando ftp anónimo

```
Directory ATL"anonymous@cs.uwp.edu password"::

"pub/music/classical/Index"
"pub/music/classical/Chamber.lat"
"pub/music/classical/Concerto.lat"
"pub/music/classical/Keyboard.lat"
"pub/music/classical/Orchstrl.lat"
"pub/music/classical/Chamber.brq"
"pub/music/classical/Keyboard.brq"
"pub/music/classical/Orchstrl.brq"
"pub/music/classical/Guide"
...
Total of 22 files.

$ copy atl"anonymous@cs.uwp.edu guest"::-
_$ "pub/music/classical/Guide" guide.txt
```
Obtener el archivo 'Guide'
(En VMS, '-' indica que el comando continúa en la siguiente línea)

Por otro lado, el formato de las direcciones de correo electrónico se modifican de acuerdo a la sintaxis

gateway::"*usuario@nodo*"

Por ejemplo, la suscripción a la lista de C++ se haría de la siguiente manera:

```
$ mail
MAIL> send
To: atl::"listserv%ucf1vm.bitnet@cunyvm.cuny.edu"
Subj: Subscribe
Enter your message below. Press CTRL/Z when complete.
SUBSCRIBE TCPLUS-L Norberto Arrieta
^Z
Exit
MAIL> exit
$
```

Por último, el servicio de sesión remota se utiliza conectándose con un gateway y especificando después el nodo TCP/IP al que se desea conectar, seguido del carácter '!':

```
$ set host kelem                                    Usar a Kelem como gateway
ULTRIX V4.2 (Rev. 96) (kelem.cecafi.unam.mx)        Mensajes de Kelem

login: sirio!                                       La conexión real es con Sirio

Trying 132.248.205.1...
Connected to sirio.
Escape character is '^]'.


Cray UNICOS (sirio) (ttyp000)                       Mensajes de Sirio (supercómputo)


login: cecafi
Password:
...
sirio>                                              Sesión en Sirio
...
sirio> ^D
$                                                   De nuevo estamos en Atzin
```

XI.2 Uso del Gateway desde un nodo TCP/IP

Desde un nodo TCP/IP el Gateway se utiliza con los comandos telnet, ftp y mail descritos anteriormente, aunque la sintaxis del destino varía ligeramente. Por ejemplo, una sesión remota en un nodo DECNET se establece conectandose con telnet a un gateway y después especificando el nombre del nodo seguido de los caracteres '::':

```
aztlan>                                             Aztlan es una PC 386, no cuenta con DECNET
aztlan> telnet kelem                                Usar a Kelem como gateway
Trying 132.248.54.7...
Connected to kelem.
Escape character is '^]'.


ULTRIX V4.2 (Rev. 96) (kelem.cecafi.unam.mx)

login: atzin::                                      Crear una sesión en Atzin

        Centro de Calculo de la Facultad de Ingenieria
                    Sistema VAX/VMS 6210

 Username: NAM
 Password:
```

C	E	C	A	F	I

SISTEMA VAX/VMS 6210
BIENVENIDO

```
Last interactive login on Friday, 31-JAN-1992 18:18
Last non-interactive login on Friday, 31-JAN-1992 18:02
$ cd [.tesis]                                    Algún comando
...
$ lo                                             Terminar la sesión
NAM          logged out at 31-JAN-1992 18:21:17.07
dlogin -- session terminated
aztlan>                                          De nuevo estamos en Aztlan
```

El comando ftp se usa de manera parecida: se establece la conexión con un gateway para después especificar el nombre del nodo DECNET con el que se desea la conexión real, seguido de los caracteres '::' y el nombre de la cuenta con la que se establece la sesión:

```
aztlan> ftp atl                                  Usar a Atl como gateway
Connected to atl.
220 atl.cecafi.unam.mx FTP server (Version 4.1 Sun Mar 25
22:59:11 EST 1990) ready.
Name (atl:nam): atzin::nam
331 Password required for gateway access atzin::nam.
Password:                                        No se despliega
230 Access control info received.
ftp> cd [.tmp]          Cambiar de directorio (se usa la sintaxis del sistema remoto)
250 CWD command successful.
ftp> ls                                          Desplegar el directorio
200 PORT command successful.
150 Opening data connection for dls (132.248.54.2,1129).
PASSWD.;3   rwxrwxr-x---   31-JAN-92 12:47:14 12075   [50,0]
PASSWD.;2   rwxrwxr-x---   31-JAN-92 12:45:59 12075   [50,0]
CMPY.COM;3  rwxrwxr-x---   31-JAN-92 12:51:19  5242   [50,0]
CMPY.TXT;6  rwxrwxr-x---   31-JAN-92 12:41:13   262   [50,0]
226 Transfer complete.
ftp> quit                                        Salir de ftp
221 Goodbye.
aztlan>
```

Para hacer uso del Gateway DECNET-Internet con mail, la dirección del destinatario sigue la sintaxis

usuario%nodo@gateway

Por ejemplo:

```
atzin> mail jramirez%atzin@atl
Subject: Reunión
      La reunión programada para mañana se suspendió.
^D
Cc:
atzin>
```

Por último, es importante recordar al lector que aquí solo se describieron las utilerías básicas de TCP/IP y DECNET. Las referencias que se listan en la bibliografía tratan los mismos temas que este texto de una manera mucho más profunda; el lector puede obtener ayuda también escribiendo por correo electrónico a info@kelem.cecafi.unam.mx.

Bibliografía

A continuación se hace una lista de algunos de los textos en los que el lector puede ampliar los temas que aquí se trataron. La mayor parte de ellos pueden ser consultados en el Centro de Cálculo.

Redes de computadoras

Network Concepts and Architectures
Bill Hanconck
Digital Press
- Conceptos generales de redes. Texto a nivel introductorio -

Protocolos TCP/IP

A TCP/IP tutorial, RFC 1180[1]
T. Sacolofsky, C. Cale
Disponible a través de la Internet
- Conceptos generales de TCP/IP. Descripción de las utilerías básicas -

Internetworking with TCP/IP: Principles, protocols and architecture, vol. 1 & 2
Douglas E. Comer
Prentice Hall
- Descripción de los protocolos TCP/IP. Texto avanzado para lectores interesados en la implementación de TCP/IP -

[1]Este documento, el RFC 1118 y el Internet-tour pueden ser consultados en CECAFI en el volumen titulado 'Guía de acceso a la Internet'.

Internet

The Hitchhker's Guide to the Internet, RFC 1118
Ed Krol
Disponible a través de la Internet
- Historia, configuración actual, servicios y otra información sobre la Internet -

Internet-tour
Disponible a través de la Internet
- Información diversa sobre la Internet. Incluye una lista de centros de supercómputo y otros servicios -

Unix

El Entorno de programación Unix
Brian W. Kernighan, Rob Pike.
Prentice-Hall Hispanoamericana
- Descripción de las utilerías de Unix. Excelente para programadores en este ambiente -

Unix Administration Handbook
Evi Nemeth, Garth Snyder
Prentice-Hall
- Configuración de sistemas Unix. Incluye un excelente capítulo sobre redes -

DECNET

VMS - Guide to DECnet-VAX networking
Digital Equipment Corporation
- Descripción de las utilerías de DECNET en VMS -

DECnet-Ultrix
Digital Equipment Corporation
- Descripción de las utilerías de DECNET en Ultrix -

VMS

VMS - User's guide
Digital Equipment Corporation
- Descripción de las utilerías de VMS -