UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Sistema de automatización para el monitoreo de una red LAN bajo ambiente Windows y UNIX

TESIS

Que para obtener el título de

Ingeniero en Computación

P R E S E N T A

Emmanuel Enciso Barajas

DIRECTOR DE TESIS

M.I. Marcial Contreras Barrera

Ciudad Universitaria, Cd. Mx., 2004

ÍNDICE

Un sistema de monitoreo es una herramienta fundamental para la administración de redes, sus principales tareas son :

- La interpretación de los datos relacionados con el estado y el desempeño de los dispositivos conectados a la red. Mediante su correcta interpretación y llevando un registro histórico de los acontecimientos que se van dando a lo largo del tiempo, el administrador de red podrá determinar de manera más rápida el comportamiento de los equipos.

- Control estadístico del tráfico existente en la red para poder determinar sus causas y así tomar decisiones al respecto. Dentro de esta estadística se tiene que contabilizar el número de paquetes que pasan a través de la red en un determinado tiempo.

- Tener una bitácora como historial para un análisis minucioso posterior.

- Detectar aquellos usuarios que hagan uso inapropiado de los servicios que se proporcionan para tomar medidas de seguridad.

- Determinar el tiempo de conexión dentro del servidor por cada uno de los clientes así como verificar los servicios que se esta utilizando o en su defecto si el usuario tiene permitido hacer uso de los servicios proporcionados.

- Detectar el tráfico que genera una máquina en particular.

- Determinar los servicios y aplicaciones mas utilizadas por los usuarios.

Hoy en día el desarrollo de redes de computadoras ha marcado una pauta muy importante dentro de lo que es el mundo de las comunicaciones, UNIX es una herramienta indispensable, debido a que es un sistema operativo multiusuario, multiproceso y estándar, que tiene servicios de red integrados y disponibles.

1.1 Objetivos

- Implementar una herramienta funcional para el monitoreo de una red.
- Capacidad de análisis de tráfico en tránsito dentro de un servidor para obtener una estadística de la cantidad de paquetes enviados por parte de los clientes.
- Mostrar todas las transacciones realizadas por los usuarios para evitar el uso indebido por parte de ellos.
- Capturar todos los paquetes TCP/IP que se envían a un servidor para así tener una estadística de los servicios mas utilizados por parte de los clientes.
- Determinar las diferentes direcciones IP que se conectan a un servidor a través de las aplicaciones que utilizan el protocolo TCP/IP.
- Tener un archivo histórico (bitácora) de las transacciones realizadas durante una determinada fecha.

Metodología de desarrollo:

La planeación de los tiempos dentro de cada etapa de la metodología de desarrollo permitirá la correcta distribución de actividades, lo que garantiza el correcto funcionamiento del software a desarrollar.

La figura 1.1 muestra las etapas consideradas para el desarrollo del proyecto.

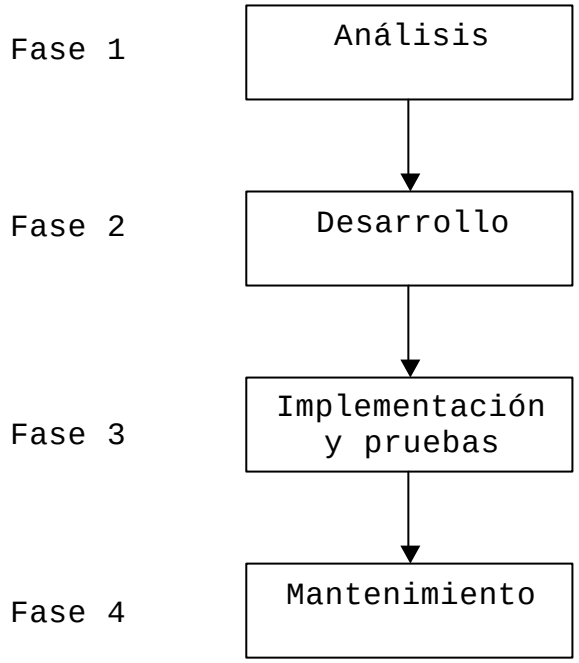


Figura 1.1 Fases del proyecto

1.2 Definición del problema

En la actualidad el mundo de las redes es una de las partes mas importantes en el ámbito de trabajo. Ahora en cualquier empresa por muy pequeña que sea existe una computadora. Para las empresas mas grandes es necesario proteger los servicios que puedan ofrecer como un servidor de correo, servidor Web, servidor de BD, etc.

Existen muchos monitores de redes con gráficos, estadísticas, bitácoras, etc. Por esta razón es imprescindible contar con una herramienta propia que sea funcional, que brinde lo que realmente se necesita y sobre todo que sea muy flexible para cualquier cambio futuro.

Los monitores existentes realizan muchas funciones y tienen características propias como el lenguaje utilizado, limitaciones respecto al funcionamiento, falta de código fuente, etc.

Por estos motivos se tiene que tener un monitor flexible y con adaptaciones fáciles que el administrador necesite, pensando en el crecimiento de los servicios futuros.

1.3 Requerimientos

Dentro de la fase de análisis, es necesario conocer las requerimientos y necesidades de la red. Algunos puntos a considerar son:

- El número de nodos existentes dentro de la red.
- Hardware y software con que actualmente cuenta la red.
- Número de servidores dentro de la red.
- Conocer las aplicaciones y servicios que ofrecen los servidores.
- Conocimiento de los protocolos utilizados dentro de la red.

Por otra parte es necesario conocer los diversos lenguajes de programación que podrán permitir el desarrollo del sistema de monitoreo como pueden ser: Lenguaje C, Java, Delphi, etc.
El conocimiento de un lenguaje en particular nos ayudará a evitar el desarrollo masivo de código, ya que la utilización adecuada de sus estructuras evita el alargamiento de los programas y su difícil entendimiento para versiones posteriores.

Otros puntos importantes a considerar son:

- Investigación avanzada del lenguaje a utilizar.
- Investigación del funcionamiento de protocolos.
- Creación de sockets.
- Habilidad para resolver algoritmos.
- Investigación de redes.
- Investigación de los servicios que provee un servidor.

El desarrollo del proyecto, será posterior a algunas investigaciones que se deben realizar para tener los conceptos firmes y claros, así como para determinar el buen funcionamiento del sistema a través de algunas pruebas tanto teóricas como prácticas.

La implementación del software será consolidada después de diversas pruebas dentro de las cuales se considerarán los siguientes procedimientos:

- Pruebas de conexión cliente-servidor
- Pruebas de conexión a los puertos TCP de un servidor.
- Análisis de paquetes TCP que pasan a través del servidor.

El buen hábito de programación facilitará el mantenimiento del software para poder realizar una siguiente versión de la aplicación en base a la primera así como tener en cuenta el crecimiento de la red y las aplicaciones que se pueden proveer.

Para la etapa de mantenimiento se considerarán los siguientes aspectos:

- Fácil manipulación del código fuente.
- Creación de manual de procedimiento para la utilización del software.
- Crecimiento de las aplicaciones que puede ofrecer un servidor.
- Consideración del crecimiento de la red.

2.1 Modelo OSI

En 1984, la Organización Internacional de Estandarización (ISO) desarrolló el modelo **OSI** (Open System Interconection). Este define términos entendibles y ampliamente utilizados en las comunicaciones de datos. El modelo OSI no es un protocolo, ni define a los protocolos, sino que sirve para definir características y funciones que deben tener los protocolos.
Tiene como finalidad principal describir el uso de datos entre la conexión física de la red y la aplicación del usuario final. Este modelo es el mejor conocido y el más usado para describir los entornos de red.

2.2 Capas de red

El modelo OSI divide las redes en capas. Cada una de estas capas debe tener una función bien definida y debe de relacionarse con sus capas inmediatas mediante unas interfaces también bien definidos. Los creadores del modelo OSI consideraron que eran 7 el número de capas que mejor se ajustaba a sus requisitos.

La figura 2.1 muestra las capas del modelo OSI.

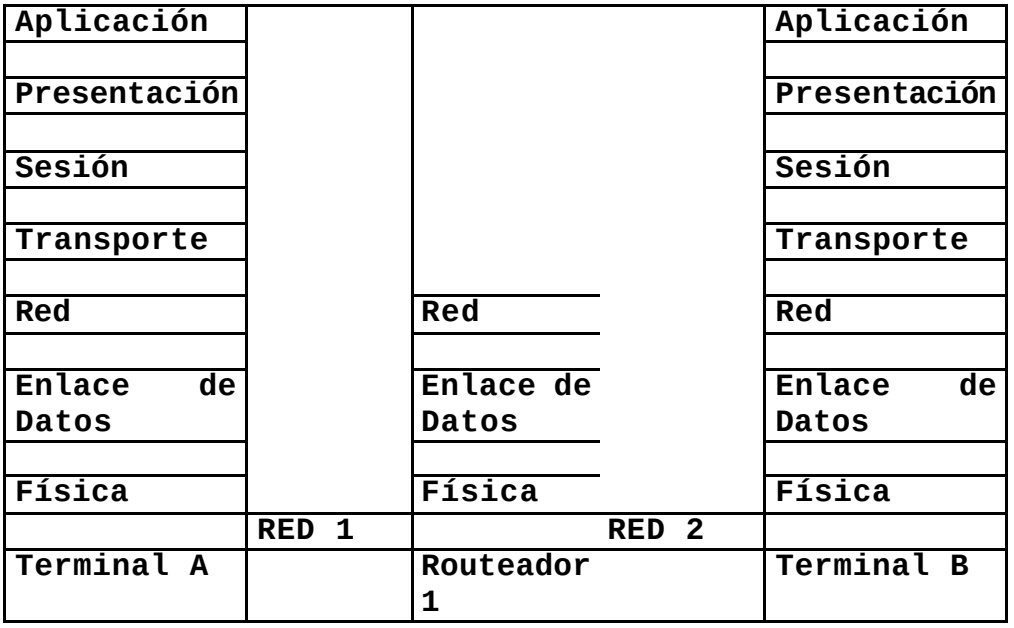


Figura 2.1 Capas del Modelo OSI

La figura 2.1 muestra las capas del modelo OSI. Las tres primeras capas se utilizan para enrutar esto es, mover la información de unas redes a otras. En cambio, las capas superiores son exclusivas de los nodos origen y destino. La capa física está relacionada con el medio de transmisión.

En la figura 2.2 se observa el flujo de información entre cada una de las capas que constituyen este modelo.

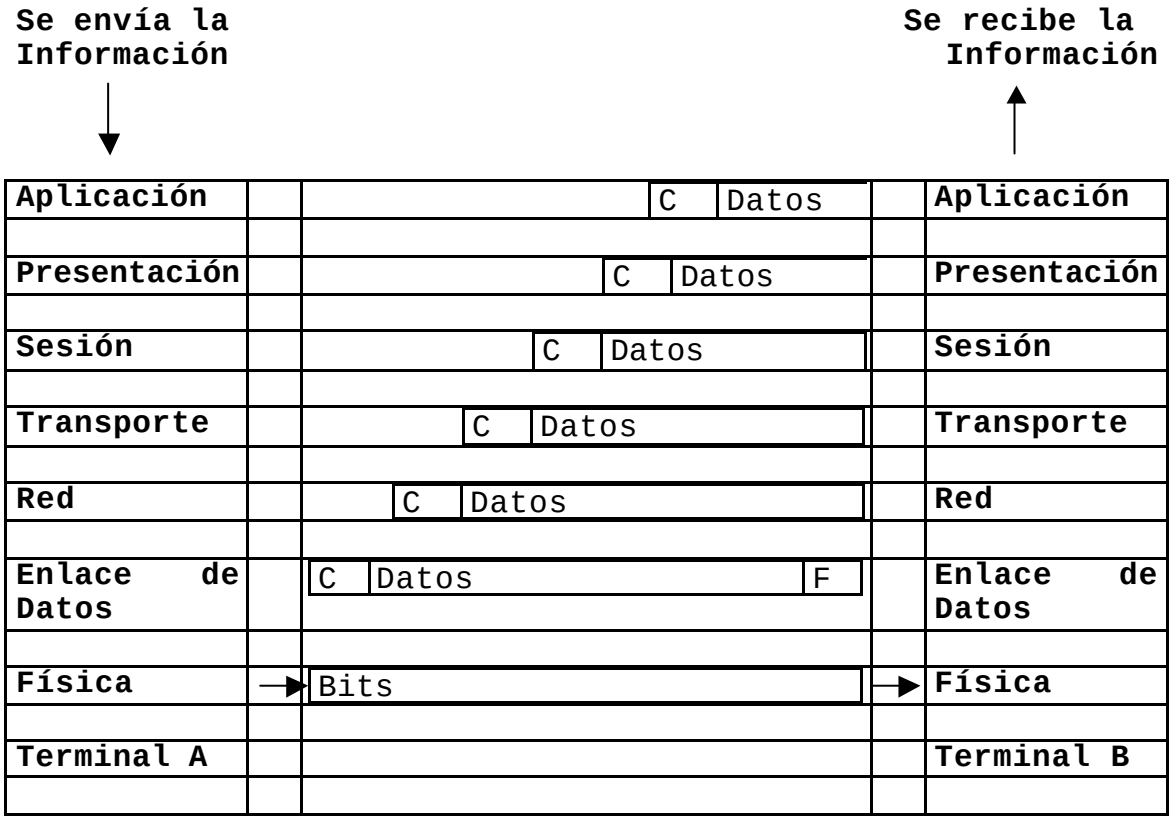


Figura 2.2 Flujo de información entre las capas

La terminal A es el nodo origen y la terminal B, el nodo destino. El mensaje son los "datos" que están situados encima de la capa de aplicación. Estos datos van descendiendo de capa en capa hasta llegar a la capa física de la terminal A. Cada capa añade un encabezado (C = cabecera) a los datos que recibe de la capa superior antes de enviar a su capa inferior. En la capa de enlace de datos se añade también una serie de códigos al final de la secuencia (F = final) para delimitar no sólo el comienzo sino también el final de un paquete de datos.

La capa física no entiende de datos ni de códigos, únicamente envía una secuencia de bits por el medio de transmisión (un cable).

Estos bits llegarán, probablemente pasando por varios encaminadores intermedios, hasta la capa física del destino. A medida que se van recibiendo secuencias de bits, se van pasando a las capas superiores.
Cada capa elimina su encabezado antes de pasarlo a una capa superior. Finalmente los datos llegarán a la capa de aplicación, serán interpretados y mostrados al usuario de la terminal B.

Capa Física

La Capa Física del modelo de referencia OSI, es la que se encarga de las conexiones físicas de la computadora hacia la red, en este nivel están por ejemplo, los estándares de cable de par trenzado que se deben de usar para conectar una red, la forma en que las antenas de microondas deben de estar orientadas para comunicarse, así como las características de propagación de ondas radiales, etc.

Capa de Enlace de Datos

Esta es la responsable de la comunicación de estación a estación en la red, impone la organización lógica de los bits en paquetes de información, también se especifican las estrategias y mecanismos para accesar al medio de comunicación, la forma en que los datos serán transmitidos y la forma en que serán reensamblados en el destino; las funciones de esta capa son:

Detectar y posiblemente corregir errores de la capa física
Poder establecer la comunicación y cerrar ésta

El propósito de esta capa es proveer los medios funcionales para activar, mantener y desactivar una o más conexiones de enlace de datos entre la capa de red.

Es la primera capa que obtiene los datos como paquetes, los ensambla, los revisa, efectúa correcciones de errores y calcula la suma de los paquetes que llegan; si el paquete está dañado es descartado, y si la capa puede determinar de dónde proviene el paquete, éste envía un mensaje de error. SDLC (Synchronous Data Link Control) y HDLC (High Level Data Link Control), son protocolos que operan en esta capa.

Capa de Red

La capa de red proporciona el enrutamiento físico de los datos de un nodo a otro y tiene como función proporcionar una trayectoria de conexión entre una pareja de entidades de la capa de transporte. En este nivel se agrupan protocolos de retorno para el funcionamiento de la red, tales como algoritmos de rotación y control de congestión en la red.

El nivel de red permite la comunicación entre computadoras en diferentes redes. Este tipo de comunicación es llamado interconexión de redes; existen varios protocolos que ayudan a realizar esta tarea en la red como los son Internetwork Datagram Protocol (IDP), Internet Protocol (IP), Internetwork Packet Exchange (IPX) y Datagram Delivery Protocol (DDL.)

Capa de Transporte

La capa de transporte tiene un significado especial porque es el primer nivel en el conjunto de protocolos que provee una comunicación fin a fin entre estaciones de red; este nivel conduce su conversación con su destino sin importar el número de computadoras intermedias o redes entre la computadora origen y destino. El nivel de transporte usa el ruteo de nivel de red para establecer comunicación entre dos máquinas (los dos puntos finales), la unidad de información en el nivel de transporte es generalmente llamado segmento.

Esta capa establece, mantiene y termina las comunicaciones entre dos máquinas, siendo responsable de comprobar que los datos enviados coincidan con los recibidos, además administra el envío de los datos, y determina su orden y prioridad.

Capa de Sesión

Esta capa organiza y sincroniza el intercambio de datos entre los proceso de la aplicación, trabaja en forma conjunta con la capa de aplicación para proporcionar conjuntos sencillos de datos. Está involucrada en la coordinación de las comunicaciones entre diferentes aplicaciones, y le permite a cada una conocimiento del estado de la otra.

Capa de presentación

Esta capa convierte los datos de la aplicación a un formato común, asegurando que la información enviada por la capa de aplicación de un sistema, será leída por la misma de otro sistema. La capa de presentación procesa datos dependientes de la máquina de la capa de aplicación, en un formato independiente de la máquina, útil para las capas inferiores.

Capa de Aplicación

Es la capa que realiza la función de interfaz del sistema OSI con el usuario final, en ella residen los programas de aplicación, como hojas de cálculo, correo electrónico o los módulos de despliegue de base de datos. Su tarea es desplegar información recibida y enviar los nuevos datos del usuario a las capas inferiores.

Es aquí donde lo visible y lo mas orientado al usuario se genera. A esta capa pertenecen por ejemplo los Web Browsers, FTP, el correo electrónico, telnet, los java applets y demás.

Dentro de estas etapas, suelen utilizar diversos dispositivos como routers, hubs, switches, etc.

Routers: Es un encaminador que acepta la información y puede elegir el mejor camino para entregar la información, además evita embotellamientos de información. Tienen tablas de ruteo que contiene direcciones IP.

Hub: El "Hub" básicamente extiende la funcionalidad de la red (LAN) para que el cableado pueda ser extendido a mayor distancia, es por esto que un "Hub" puede ser considerado como una repetidora

Switch: Es considerado un "Hub" inteligente, cuando es inicializado el Switch, éste empieza a reconocer las direcciones MAC que generalmente son enviadas por cada puerto, en otras palabras, cuando llega información al Switch éste tiene mayor conocimiento sobre que puerto de salida es el *más apropiado*, y por lo tanto ahorra una carga de tráfico a los demás puertos del Switch.

2.3 Conceptos y términos importantes

Protocolo : Un protocolo de comunicación es aquel en el cual se pueden definir las reglas para la transmisión y recepción de información entre todos los nodos existentes dentro de una red, si dos nodos dentro de dicha red se desean comunicar, será necesario que ambos empleen la misma configuración de protocolos.

Entre los protocolos de una red local podemos distinguir a dos principales grupos, por un lado tenemos los Protocolos de los niveles físico y de enlace (Niveles 1 y 2 del modelo OSI), que definen las funciones asociadas con el uso del medio de transmisión: envío de los datos a nivel de bits y trama y el modo de acceso de los nodos al medio. Estos protocolos vienen unívocamente determinados por el tipo de red (Ethernet, Token Ring, etc.).

Por el otro lado tenemos aquellos que realizan la función de nivel de Red y Transporte, niveles 3 y 4 de OSI. Este tipo de Protocolo transmite la información a través de la Red en pequeños segmentos llamados paquetes, es decir, si un ordenador requiere enviar un fichero grande a otro ordenador, dicho fichero es dividido en cierto número de paquetes en el origen y vueltos a ensamblar en el ordenador destino.
La información es ensamblada en paquetes de datos para poder llevar a efecto la transferencia de datos. Cada paquete incluye tres características de información:

Datos de la Carga: La información que se quiere transferir a través de la red.

Dirección : Destino del paquete o de la información, en donde cada uno de los ordenadores que están dentro de un segmento de Red tiene solo una dirección.

Código de Control: Informa la descripción del tipo de paquete y el tamaño de este. También llamados códigos de verificación de errores y de otra información.

Trama : La comunicación entre una estación y otra a través de una red Ethernet se realiza enviando información. El mensaje que se quiere transmitir se descompone en uno o más paquetes, a estos paquetes se le conoce con el nombre de tramas. El formato de cada una de estas es la siguiente:

Encabezado de la Trama	Área de datos del datagrama IP	Final de la trama

Dentro de este formato de Trama, tenemos diversos campos, donde se mencionan los más importantes:

Versión : Indica la versión del protocolo IP que se utilizó para poder crear el datagrama.

Longitud : Es el tamaño expresado en bytes, de cada una de las tramas.

Identificación : Número de secuencia que junto a la dirección origen, dirección destino y protocolo, identifica de manera única dentro de la red a ésta trama.

Tiempo de vida : Es el tiempo máximo expresado en segundos que puede estar un datagrama dentro de la red.

Protocolo : Indica el protocolo utilizado en el campo de datos.
Dirección Origen : Contiene la dirección IP de la máquina que envía la información.

Dirección Destino : Contiene la dirección IP de la máquina que se desea enviar la información.

MTU : El MTU de una red es la mayor cantidad de datos que puede transportar los paquetes en su trama física.

Fragmentación : La fragmentación es la división de un cierto número de datagramas para poder atravesar las redes en camino. Dependiendo del MTU se puede fragmentar o no, ya que la red en que se este trabajando lo determina. El MTU en las redes Ethernet es de 1500 bytes, y en Token Ring es de 8,192 bytes.

Para poder ejemplificar este concepto se tiene la figura 2.3:

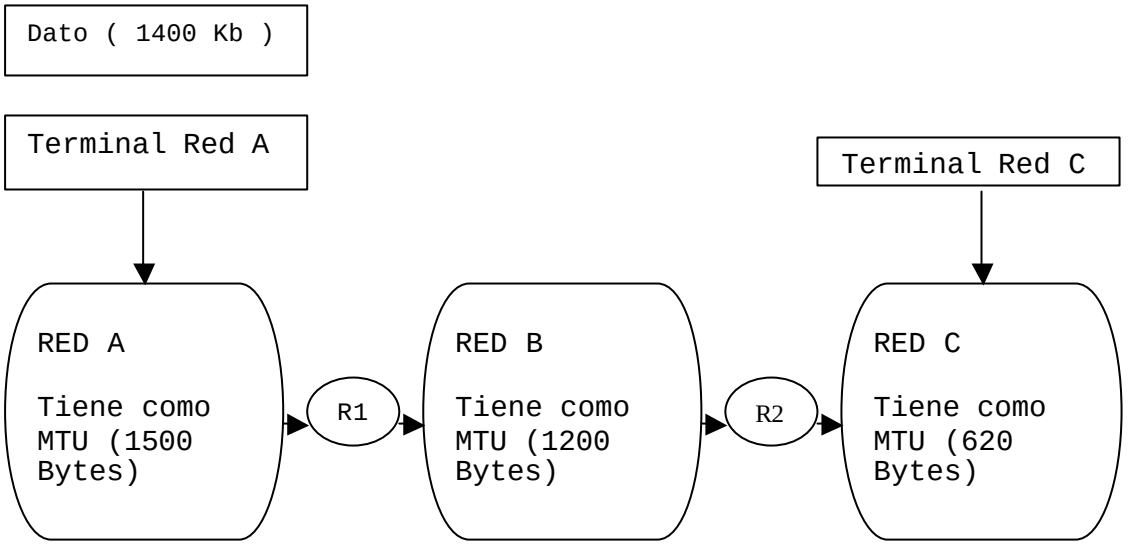


Figura 2.3 Ejemplificación del MTU

En la figura 2.3 se puede apreciar que una estación dentro de la red A desea enviar un datagrama de 1400 bytes a una estación que se encuentra en la red C. Primeramente tenemos que el datagrama puede atravesar sin ningún problema la red A, ya que el tamaño del datagrama de la estación A es menor que el MTU dentro de la red A.

Para poder atravesar a la red B no es posible, ya que el tamaño del datagrama es mayor que el MTU de la red B, por lo cual es necesario que el ruteador R1 fragmente este datagrama de la forma siguiente:

El fragmento 1 tendrá exactamente las mismas características del original exceptuando el número de datagramas y el tamaño de este. En esta primera fase, el fragmento será de 1200 bytes y el fragmento 2 tendrá 200 bytes.
Para poder pasar a la tercera red es necesario volver a fragmentar, es decir, fragmento 1 de 620 bytes, fragmento 2 en 580 bytes y el fragmento 3 en 200 bytes.
Es importante considerar también que dentro de la información de la trama, la bandera de NF (No fragmentar) pudo haber estado activada, por lo que ninguna trama hubiera podido atravesar las redes, por lo tanto el ruteador R1 lo hubiera descartado.

Direccionamiento : Una dirección indica donde se encuentra o donde está esa máquina. Una ruta establecida indica donde llegar ahí. El protocolo IP maneja principalmente direcciones, esto es, las direcciones deben de ser fijas 4 bytes (32 bits) donde esta comienza por un número de red seguido por una dirección local.

Existen tres clases de red; en la clase A el bit más significativo es “0”, los siguientes 7 bits son la red y los restantes 24 bits son la dirección local. Para la clase B tenemos que los bits más significativos son “10”, los siguientes 14 son la red y los 16 restantes son la dirección local. Por último tenemos la clase C, donde tenemos los tres bits más significativos “110”, los 21 bits siguientes identifican a la red y los últimos 8 a la dirección local.

Encaminamiento : Se le conoce como encaminamiento, a la ruta mas adecuada que debe seguir un datagrama enviado por una terminal origen hasta el destino. Dentro de lo que son las redes de redes, estas se conectan a través de routers o encaminadores, donde su principal función es elegir las mejores rutas o el mejor camino para poder llegar a su destino, la información que viaja en la red. Los routers deben conocer un poco respecto a la estructura de la red, esto les permitirá encaminar de forma eficiente cada uno de los mensajes de la información hacia su destino.

La información que requiere el ruteador se almacena en tablas de encaminamiento, dicha información corresponde a los prefijos de las direcciones.
Por ejemplo, si se desea enviar información hacia un host con dirección IP 132.248.67.157 y con una mascará 255.255.0.0, entonces el ruteador o encaminador de la red será 132.248.0.0, esto es porque todas las terminales que empiecen con 132.248.*.* se enviarán hacia el mismo punto.

Puertos : Un puerto es un número, el cual se forma a través de 16 bits, por lo que tenemos 2 ^ 16 número máximo de puertos. Esto quiere decir que en cada terminal o host tenemos 65,536 puertos disponibles. Cada aplicación que nosotros trabajamos para comunicación con las terminales utilizan estos puertos para poder recibir y enviar información. Por lo regular, las aplicaciones que conocemos como clientes utilizan números de puertos superiores a 1024, debido a que los primeros 1024 son utilizados por el sistema.

A continuación se tienen algunos de los puertos comúnmente utilizados por el sistema:

Clave	No. De Puerto	Descripción
Echo	7/UDP	Echo
Echo	7/TCP	Echo
Echo	7/UDP	Echo
Systat	11/UDP	Active User
Systat	11/TCP	Active User
Daytime	13/UDP	Daytime
Daytime	13/TCP	Daytime
ftp	21/TCP	File Transfer
ssh	22/TCP	Ssh
telnet	23/TCP	Telnet
smtp	25/TCP	Simple Mail Transfer
hostname	101/TCP	Host name Server
rtelnet	107/TCP	Remote Telnet Service
rtelnet	107/UDP	Remote Telnet Service
pop3	110/tcp	Post Office Protocol

Cabe mencionar que los puertos tienen una capacidad de almacenamiento, esto se le conoce como buffer, donde las aplicaciones envían información a los puertos, estos reciben la información y la almacenan hasta el momento en que pueden enviarla por la red.

2.4 Conjunto de Protocolos TCP/IP

Los protocolos TCP/IP son semejantes al modelo de referencia OSI, dicho modelo describe a una red ideal de computadoras en la cual la comunicación en la red ocurre entre procesos y niveles que proveen de servicios a los niveles superiores y recibe servicios de los niveles inferiores.

El modelo de niveles permite realizar las funciones de red en cada nivel, solo es necesario conocer el servicio que el nivel necesita para proveérselo. Las aplicaciones desarrolladas por TCP/IP generalmente requieren de varios de los protocolos que forman el conjunto TCP/IP para poder comunicarse.

El conjunto de los niveles de los protocolos también es conocido como protocolo stack o pila y funciona como sigue: el nivel más alto de la computadora fuente pasa información a los niveles más bajos del modelo TCP/IP, hasta llegar al último nivel que es el físico. El nivel físico transfiere la información a su destino.
Los niveles más bajos de la computadora destino pasan la información a sus niveles más altos, los cuales llevan a los datos a la aplicación destino.

Cada protocolo dentro del modelo TCP/IP tiene varias funciones, y cada función es independiente de los otros niveles. Sin embargo, cada nivel espera recibir ciertos servicios de los niveles vecinos.

2.4.1 Protocolo IP (Internet Protocol)

IP es el principal protocolo de la capa de red. Este protocolo define la unidad básica de transferencia de datos entre el origen y el destino, atravesando toda la red de redes. Además, el protocolo IP es el encargado de elegir la ruta más adecuada por la que los datos serán enviados. Se trata de un sistema de entrega de paquetes (llamados datagramas IP).

El datagrama IP, viaja en el campo de datos de las tramas físicas de las distintas redes que va atravesando. Cada vez que un datagrama tiene que atravesar un router, el datagrama saldrá de la trama física de la red que abandona y se acomodará en el campo de datos de una trama física de la siguiente red. Este mecanismo permite que un mismo datagrama IP pueda atravesar redes distintas: enlaces punto a punto, redes ATM, redes Ethernet, redes Token Ring, etc. El propio datagrama IP tiene también un campo de datos será aquí donde viajen los paquetes de las capas superiores.

2.4.2 Protocolo ARP (Protocolo de Resolución de direcciones)

Dentro de la misma red, las máquinas se comunican enviándose tramas físicas. Estas contienen las direcciones físicas de origen y destino donde cada una ocupa 6 bytes. El protocolo ARP (Address Resolution Protocol), obtiene la dirección física del ordenador a partir de su IP, ya que el único dato que se indica en los datagramas es la dirección IP.

En la figura 2.4, se desea enviar un datagrama desde la red A, a través de la Terminal 1 y hasta la Terminal 2 que se encuentra dentro de la red B.

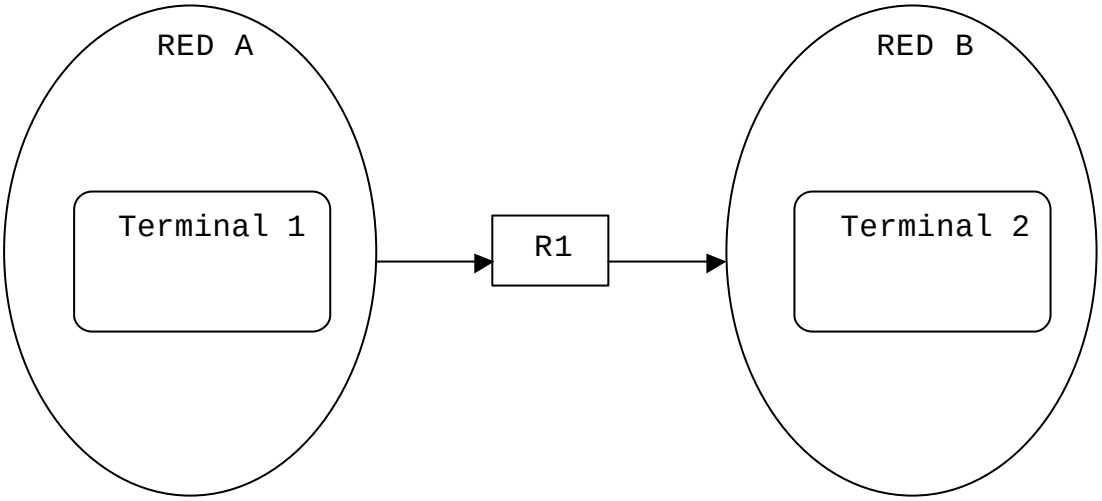


Figura 2.4 Envío de datagramas

Funcionamiento del protocolo ARP.
La terminal A tiene las siguientes características:
Pertenece a la Red A
Dirección Física : 00-04-06-08-0A-0B
Dirección IP : 132.248.67.157

La terminal B tiene las siguientes características:
Pertenece a la Red B
Dirección Física : 00-04-0C-0D-0E-4F
Dirección IP :200.38.75.87

El Routeador R1 tiene las siguientes características:
Dirección Física : 00-43-4F-07-10-0B
Dirección Física : 00-54-4D-09-A3-BF
Dirección IP : 132.248.67.1
Dirección IP : 200.38.75.1

En el momento en que la Terminal 1 envía un paquete hacía la Terminal 2, verifica que la Terminal 2 no se encuentra dentro de esta red, por lo tanto deberá de atravesar el router R1. La Terminal 1 envía un mensaje ARP a todas las máquinas de su red, pidiendo la dirección física de la IP (200.38.75.87), por lo tanto el routeador R1, contesta que ese datagrama esta dirigido hacia el y le contesta a la Terminal 1 con su dirección física (00-43-4F-07-10-0B). Como consecuencia de esta petición, la Terminal A envía una trama con origen 00-04-06-08-0A-0B y destino 00-43-4F-07-10-0B, conteniendo dentro del datagrama la IP origen 132.248.67.157 y la IP destino 200.38.75.87.
Una vez que el datagrama es entregado al routeador por parte de la red A se efectúa el mismo proceso para entregar el datagrama a la Terminal 2.

Las tablas ARP se almacenan dentro de cada uno de los ordenadores que contienen principalmente IP´s y Direcciones Físicas. La tabla se va formando cada vez que se va preguntando y se va respondiendo, si es la primera vez que pregunta, entonces se ingresa a su tabla, las peticiones siguientes no será necesario ingresarlas nuevamente ya que se guarda la dirección física. Cabe destacar que cada vez que se ingresa a una tabla, se le asigna un tiempo de vida de cierto número de segundos, cuando este tiempo se termine, entonces se elimina de la tabla.

2.4.3 RARP (Reverse Address Resolution Protocol)

Es el protocolo que mapea las direcciones físicas a direcciones lógicas.
Es una variante de ARP. RARP también convierte direcciones, pero al contrario de lo que hace ARP. Este convierte direcciones Ethernet a direcciones IP.

RARP ayuda a configurar sistemas diskless, para que estas conozcan su dirección IP, mandando su dirección física y esperando de regreso una dirección IP. La respuesta que contenga la dirección IP se envía mediante un servidor RARP, una máquina que puede suministrar dicha información. A pesar de que el dispositivo de origen envía el mensaje en forma de difusión las reglas de RARP estipulan que solamente el servidor RARP podrá generar una respuesta.

2.4.4 Protocolo ICMP (Internet Control Message Protocol)

Este protocolo es el encargado de informar al host de origen si se ha producido algún error durante la entrega de los paquetes. También suele transportar distintos mensajes de control.
Los mensajes ICMP (Protocolo de Mensaje de Control) son enviados en varias circunstancias:

- Cuando los paquetes de información no pueden alcanzar el host destino.
- Cuando una pasarela (Gateway) no tiene la capacidad para poder almacenar de forma temporal los paquetes de información.
- Cuando la pasarela puede dirigir al host para enviar el tráfico por una ruta más corta.

La información que se envían de solicitud y respuesta (eco), se utilizan para poder comprobar la conexión entre dos hosts a nivel de la capa de red. Esta información comprueba el buen funcionamiento y configuración de las capas de red, como son: la capa física, capa de acceso al medio y capa de red.

Un ejemplo del uso de este protocolo es el comando PING. El cual consiste en hacer una petición a un host remoto a través de un eco, donde el host remoto responde la petición solicitada.
Los tipos de mensajes dentro de este protocolo son importantes, ya que cada uno de ellos tiene un significado.

En la tabla 2.5 vemos los tipos de mensajes:

TIPO	Mensaje (ICMP)
0	Respuesta de eco
3	Destino inaccesible
4	Disminución del tráfico
5	Redireccionar
8	Solicitud de eco
11	Tiempo excedido para un datagrama
12	Problema de Parámetros
13	Solicitud de marca de tiempo
14	Respuesta de marca de tiempo
15	Solicitud de información
16	Respuesta de información
17	Solicitud de máscara
18	Respuesta de máscara

2.5 Tabla de mensajes del protocolo ICMP

Los datagramas IP tienen un campo llamado TTL, el cual funciona como un reloj en forma descendente, es decir, tiene un tiempo límite de vida para ese datagrama, esto con la finalidad de que la información no este dando vueltas infinitamente en la red. Cuando la bandera TTL llega a ser “0”, este paquete se descarta y envía un mensaje de tiempo excedido para que el host origen este informado de ese paquete.

2.4.5 Protocolo UDP (Protocolo de Datagramas del Usuario)

Este protocolo tiene características muy similares al protocolo IP, ya que ofrece comunicaciones muy sencillas entre hosts. Las características principales son las siguientes:

- No se puede establecer una conexión antes con otro host para poder enviar información UDP, esto se le conoce como protocolo No Orientado a Conexión.
- La información que se envía a través de la red puede llegar errónea, dañada o pueden perderse en el trayecto, por eso es No Fiable.

El formato del mensaje UDP es el siguiente:

16 Bits	**16 Bits**
Puerto Origen	Puerto Destino
Longitud	Verificación
Datos	

Puerto Origen : Es el número de puerto de la máquina origen.
Puerto Destino: Es el número de puerto de la máquina destino.
Longitud : Es la longitud medida en bytes del mensaje.
Verificación : Esto es la suma de verificación de error del mensaje.
Datos : Es la información que se desea enviar.

2.4.6 Protocolo TCP (Protocolo de Control de Transferencia)

Las principales características de este protocolo son:

- Es necesario antes que nada, establecer una conexión entre las dos terminales que se desean comunicar, ya establecida dicha conexión se puede empezar la transmisión de información. Con esto tenemos garantizado que la información va ha llegar a la terminal destino de forma efectiva y ordenada. También se garantiza la integridad de la información, es decir, sin duplicidad de esta. Si al protocolo UDP se le llama No Orientado a Conexión, a este protocolo se le conoce como Orientado a Conexión.

- La información que se entrega a la terminal destino llega de forma correcta, por lo tanto se le llama que es fiable.

Con estas dos características, podemos concluir que el protocolo TCP, nos permite la comunicación confiable entre dos terminales o aplicaciones, por lo tanto estas aplicaciones pueden dar por hecho que toda la información que reciben es correcta.

2.4.6.1 Establecimiento de conexión

Cuando un cliente decide establecer una comunicación con un servidor, es necesario que ambos estén de acuerdo en participar, de lo contrario la comunicación no se podrá llevar a cabo. La conexión entre terminales se determina a través de peticiones y contestaciones, esto se realiza con una secuencia específica.

Una conexión TCP requiere de un proceso denominado ***"tres fases"***, en el cual se distinguen tres etapas diferenciadas.

La figura 2.6 muestra las etapas de la conexión TCP.

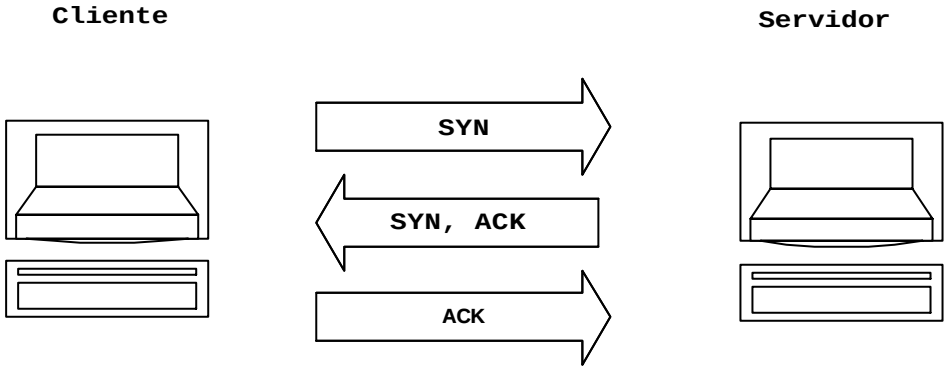


Figura 2.6 Petición de una conexión TCP

Cuando un cliente genera una llamada connect(), envía un segmento al servidor solicitando abrir un circuito TCP. Dicho segmento lleva activado el flag SYN, para indicar que el circuito está en proceso de sincronización. Este segmento de datos no suele llevar ningún tipo de dato, únicamente la cabecera IP, la cabecera TCP y algunas posibles opciones de TCP.

El servidor responde enviando un segmento de aceptación al segmento anterior. Para ello activa el flag ACK y en campo número de secuencia ACK coloca el valor correspondiente al campo número de secuencia del segmento recibido pero incrementado en una unidad. El flag SYN viaja activado para indicar que el proceso de sincronización no ha finalizado aún. En este momento, cuando el cliente recibe el segmento, sabe que el servidor ha validado su petición, pero el servidor es ahora el que espera se valide su segmento.

El cliente envía un segmento validando el enviado por el servidor, colocando el valor correspondiente en el campo número de secuencia ACK y activando el flag ACK. El flag SYN no viaja activo en esta ocasión.

A continuación se muestra un ejemplo en el que un cliente y un servidor se intercambian mensajes de 1000 bytes. Cuando la conexión se ha establecido y el cliente ha elegido el número 1023 como número de secuencia, el servidor por su parte ha elegido el número 5000 como número de secuencia.

La figura 2.7 muestra el proceso de intercambio de mensajes, teniendo en cuenta las confirmaciones:

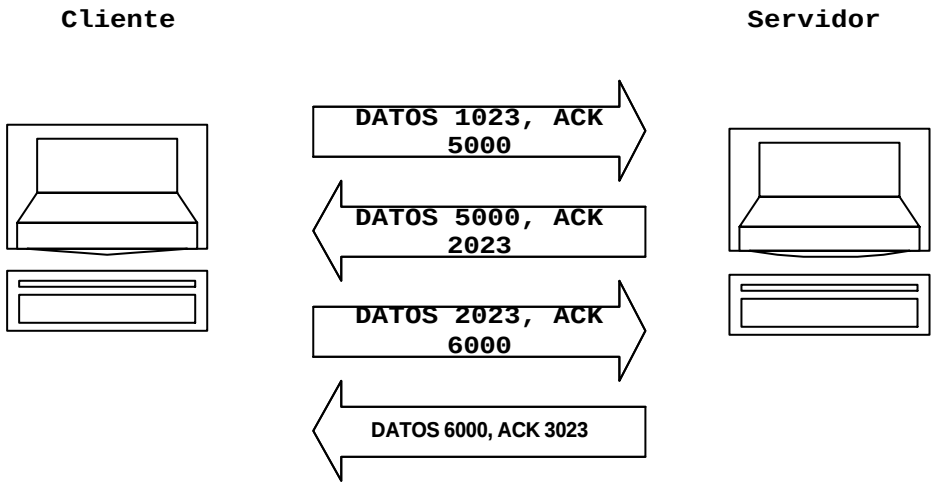


Figura 2.7 Proceso de intercambio de mensajes.

El cliente envía un primer bloque de datos, habiendo identificado el primer byte con el número de secuencia previamente elegido al azar, que es el 1023. En ese mismo mensaje, informa al servidor, que está preparado para recibir el byte identificado por el número de secuencia 5000. Hay que recordar que ambos hosts han sido informados de los números de secuencia del otro extremo en el proceso de establecimiento de conexión.

El servidor, al recibir el mensaje, envía otro segmento de datos confirmando los datos recibidos del cliente, al indicar que está preparado para recibir el byte con número de secuencia 2023. El proceso se repite durante el intercambio de información hasta que se produzca la desconexión.

2.4.6.2 Finalización de la conexión

Cuando un cliente desea finalizar de manera ordenada la conexión genera una llamada a close(), ese mecanismo genera el envío de un segmento con el flag FIN activado, dando a entender la intención de finalizar la conexión.

La figura 2.8 muestra estas cuatro fases para realizar la finalización de transmisión:

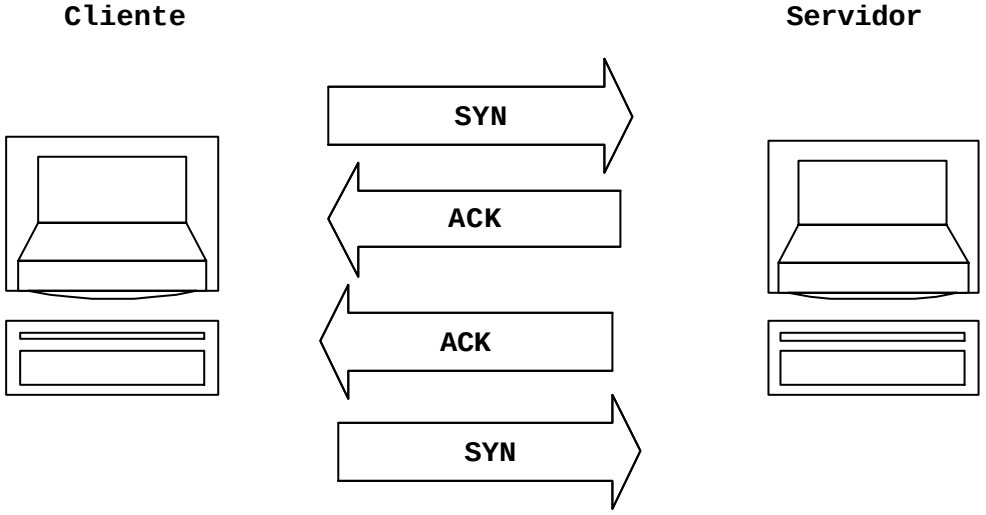


Figura 2.8 Finalización de una conexión TCP.

El servidor es el encargado en ese momento de validar cualquier información previa mediante el envió de un segmento con el bit ACK activado. En ese punto, el cliente esta preparado para cerrar la conexión, sin embargo no es posible, ya que el servidor aún no ha enviado ningún segmento con el flag FIN activado.

La figura 2.9 muestra el formato del mensaje TCP:

<table>
<tr><td colspan="2">Puerto TCP - Origen</td><td colspan="2">Puerto TCP – Destino</td></tr>
<tr><td colspan="4">Número de la Secuencia
Número de acuse de recibido</td></tr>
<tr><td>HLEN</td><td>Reservado</td><td>Bits de código</td><td>Ventana</td></tr>
<tr><td colspan="2">Suma de Verificación</td><td colspan="2">Puntero de Urgencia</td></tr>
<tr><td colspan="3">Opciones</td><td>Relleno</td></tr>
<tr><td colspan="4">Datos</td></tr>
</table>

Figura 2.9 Formato de mensaje TCP

Puerto Origen : Es el puerto del host origen de donde se envia la información.

Puerto Destino : Es el puerto del host destino a donde se enviará la información.

Secuencia : Es el número de secuencia del primer byte que transporta el segmento.

HLEN : Longitud de la cabecera, debe de ser medida en múltiplos de 32 bits.

ACK : En un mismo segmento puede transportar los datos de un sentido y las confirmaciones en el otro sentido, esto es la señal ACK.

SYN : Es la sincronización del número de secuencia.

FIN : Indica que la terminal ya no tiene mas secuencias para enviar

2.5 Direccionamiento IP

El Protocolo Internet mueve datos entre servidores en forma de datagramas. Cada datagrama se entrega a la dirección que contiene la dirección destino del encabezado del datagrama. La dirección destino es una dirección IP estándar de 32 bits que contiene información suficiente para identificar únicamente a una red y a un servidor específico en esa red.
Cada dirección IP tiene dos identificadores: “Host” y “Red”. Se pueden tener 5 clases de red (A, B, C, D y E), las cuales se diferencian en los primeros 4 bits y en el número de bits que almacena el “ID Host” y el “ID Red”.

Clase A
Se asigna para redes con un gran número de estaciones de trabajo. El bit de mayor orden de la clase A es siempre puesto en 0. Los siguientes 7 bits del primer campo representan el direccionamiento de la red. Los 24 bits restantes representan la dirección del host. Esto nos permite contar con 126 redes y aproximadamente 17 millones de nodos de red.

Clase B
El direccionamiento de la clase B es para redes medianas. El bit de mayor orden es puesto en 1-0. Los siguientes 14 bits(los primeros dos campos) representan el direccionamiento de la red. Los restantes 16 bits (los últimos 2 campos) representan el direccionamiento del nodo. Esto nos permite tener 16,384 redes y aproximadamente 65,000 nodos de red.

Clase C
Utilizada en redes tipo LAN. El bit de mayor orden es puesto en 1-1-0. Los siguientes 21 bits (primeros tres campos) representan el direccionamiento de red. Los últimos 8 bits representan la dirección del nodo. Esto permite contar con aproximadamente 2 millones de redes y 254 nodos de red.

Clase D
Se emplea para multicasting a un número de nodos. Los paquetes son pasados a un subconjunto seleccionado de usuarios en la red. Solamente estos nodos son registrados y serán quienes reciban dichos paquetes. El bit de mayor orden es puesto en 1-1-1-0. Los bits restantes son para el reconocimiento del nodo.

Clase E
Es una dirección experimental y está reservada para uso futuro, el bit de mayor orden es puesto en 1-1-1-1-0

Clase	Rango
A	1-127
B	128-191
C	192-223
D	224-254

3.1 Sistema Operativo

Un sistema operativo es un programa que actúa como intermediario entre el usuario y el hardware de una computadora y su propósito es proporcionar un entorno en el cual el usuario pueda ejecutar programas. El objetivo principal de un sistema operativo es entonces, lograr que el sistema de computación se use de manera cómoda, y el objetivo secundario es que el hardware de la computadora se emplee de manera eficiente.

Un sistema operativo es una parte importante de cualquier sistema de computación. Un sistema de computación puede dividirse en cuatro componentes: el hardware, el sistema operativo, los programas de aplicación y los usuarios. El hardware Unidad Central de Procesamiento (UCP), memoria y dispositivos de entrada/salida (E/S) proporciona los recursos de computación básicos. Los programas de aplicación (compiladores, sistemas de bases de datos, juegos de vídeo y programas para negocios) definen la forma en que estos recursos se emplean para resolver los problemas de computación de los usuarios.

3.1.1 Características de un Sistema Operativo

- Conveniencia. Un sistema operativo hace más conveniente el uso de una computadora.

- Eficiencia. Un sistema operativo permite que los recursos de la computadora se usen de la manera más eficiente posible.

- Habilidad para evolucionar. Un sistema operativo deberá construirse de manera que permita el desarrollo, prueba o introducción efectiva de nuevas funciones del sistema sin interferir con el servicio.

- Administración del hardware. El sistema operativo se encarga de manejar de una mejor manera los recursos de la computadora en cuanto a hardware se refiere, esto es, asignar a cada proceso una parte del procesador para poder compartir los recursos.

- Relacionar dispositivos (gestión a través del kernel). El sistema operativo se debe encargar de comunicar a los dispositivos periféricos, cuando el usuario así lo requiera.

- Organizar datos para acceso rápido y seguro.

- Manejar las comunicaciones en red. El sistema operativo permite al usuario manejar con alta facilidad todo lo referente a la instalación y uso de las redes de computadoras.

- Procesamiento por bytes de flujo a través del bus de datos.

- Facilitar las entradas y salidas. Un sistema operativo debe hacerle fácil al usuario el acceso y manejo de los dispositivos de Entrada/Salida de la computadora.

- Técnicas de recuperación de errores.

- Evita que otros usuarios interfieran. El sistema operativo evita que los usuarios se bloqueen entre ellos, informándoles si esa aplicación esta siendo ocupada por otro usuario.

- Generación de estadísticas.

- Permite que se puedan compartir el hardware y los datos entre los usuarios.

3.1.2 Shell

El shell permite ejecutar un conjunto de comandos que se encuentran almacenados en un archivo. En esta forma se puede configurar un archivo que contenga una determinada secuencia de comandos para posteriormente ejecutar todos los comandos del archivo con una sola instrucción.

Este es llamado *Interprete de comandos*. El proceso de ejecución de un comando es el siguiente:

1.- Tipo de shell a emplear.
2.- Escritura del comando.
3.- Ejecución del comando.
4.- El shell ejecuta el comando ordenado, encontrando el archivo dentro de un directorio del sistema utilizando el contenido de este.
5.- El shell espera la orden para otro comando.

3.2 Arquitectura del Sistema Unix

Unix es el núcleo de un SO de tiempo compartido. El núcleo del sistema siempre esta residente en memoria y entre otras cosas brinda los siguientes servicios:

- Controla todos los recursos del hardware.
- Controla los dispositivos periféricos (discos, terminales, impresoras, etc).
- Permite a distintos usuarios compartir recursos y ejecutar sus programas.
- Proporciona un sistema de archivos que administra el almacenamiento de información (programas, datos, documentos, etc.).

Unix también abarca un conjunto de programas estándar como son:

- Compilador de lenguaje C.
- Editor de texto.
- Intérprete de ordenes.
- Programa de gestión de archivos.

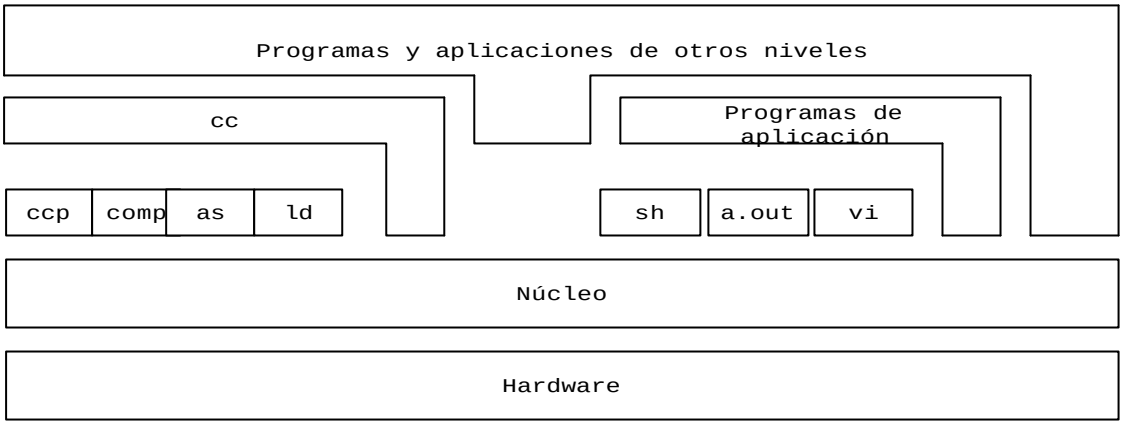


Figura 3.1 Arquitectura del Sistema Unix

En la figura 3.1 se observan los distintos niveles dentro de la Arquitectura del Sistema Unix. En el nivel más interno tenemos al hardware, es decir tenemos la máquina física en la cual esta implementado el sistema y contiene los recursos que deseamos gestionar. En el segundo nivel tenemos el *núcleo* del sistema, el cual esta escrito en su mayoría en lenguaje C y otra parte en código ensamblador.

En el tercer nivel se encuentran los programas estándar que contiene cualquier sistema Unix (vi, who, grep, tail, etc.) así como programas generados por el propio usuario. Cabe mencionar que este nivel nunca actuará sobre el hardware en forma directa, es decir, debe existir algún mecanismo para decirle al *núcleo* del sistema que se desea trabajar sobre algún recurso del hardware, esto se le conoce como *llamadas al sistema*[1].

Por encima de este nivel tenemos aplicaciones que sirven a otros programas ya creados para llevar a cabo su función las cuales no se comunican directamente con el núcleo. Un ejemplo de este tipo de aplicaciones es el compilador de C. Cuando se compila un programa en C, este invoca de forma secuencial los programas:

- Preprocesador de C (cpp)
- Compilador de C (comp).
- Ensamblador (as).
- Enlazador (Id).

[1] *Cualquier programa que se esté ejecutando bajo el control de Unix y necesite hacer uso de algún recurso, efectuará una llamada a alguna de las system calls*

Las llamadas al sistema y a sus bibliotecas, es la unión entre los usuarios y el núcleo del sistema. Esta biblioteca asociada es un mecanismo mediante el cual podemos invocar llamadas al sistema desde un simple programa escrito en C. Como podemos observar, los programas realizados en lenguaje C necesitan una biblioteca intermedia para poder hacer las llamadas, por lo contrario un programa escrito en código ensamblador puede invocar directamente llamadas al sistema sin utilizar alguna biblioteca intermedia.

Estas llamadas al sistema se ejecutan en *modo kernel*[2], y para entrar a este modo se tiene que ejecutar una sentencia en código máquina conocida como *trap*[3].

La figura 3.2 muestra el diagrama de bloques del núcleo.

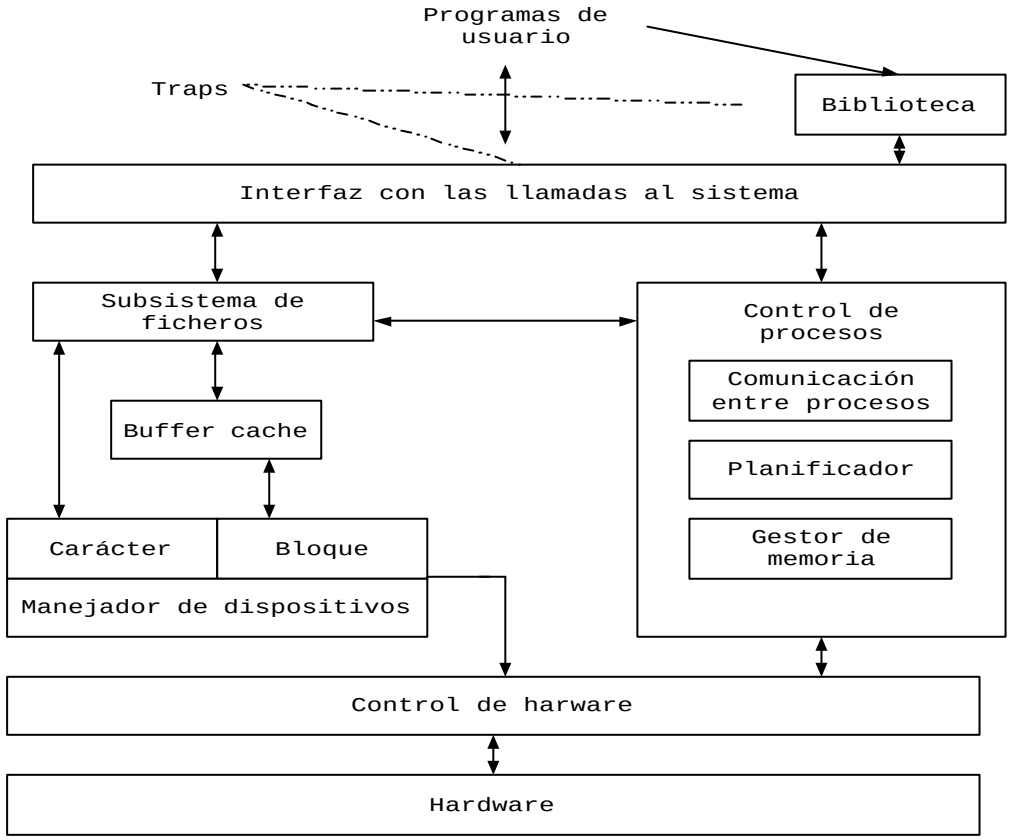


Figura 3.2 Diagrama de bloques del núcleo

[2] *Para muchos microprocesadores se le conoce modo supervisor*
[3] *Conocida también como interrupción software.*

Los dos conceptos principales en los cuales se basa el Sistema Operativo Unix, es el sistema de archivos y los procesos tal como se muestra en la figura 3.2.

El subsistema de ficheros es el encargado de administrar los recursos del sistema de ficheros así como reservar espacio para los ficheros, administrar el espacio libre, acceso a los ficheros, etc. Otra tarea es comunicarse con los dispositivos de almacenamientos secundarios como son: las unidades de disco duro, unidades de cinta, etc. esto puede llevarse a cabo mediante los manejadores o drivers de dispositivos que se encargan de establecer el protocolo de comunicación (*handshake*) entre el núcleo y los periféricos. Existen dos tipos de dispositivos dependiendo la forma de acceso, estos son por bloques y por carácter.

El dispositivo modo bloque es aquel en donde el acceso se realiza con la intervención del buffer que mejora considerablemente la transmisión.
El dispositivo modo carácter no utiliza el buffer para la transferencia es decir, se realiza de forma directa.

El subsistema de control de procesos es el que se encarga de planificar los procesos (scheduler[4]), la sincronización que debe existir entre ellos así como la comunicación entre ellos y el control de la memoria. Algunas llamadas para controlar los procesos son: *fork, signal, exec, etc.*

El gestor de memoria, es el encargado de controlar los procesos que están almacenados en la memoria principal, si en un determinado instante no existe memoria suficiente, el gestor tiene que recurrir a un proceso llamado *swapping,* esto es para que todos los procesos tengan derecho a un tiempo mínimo de ocupación en memoria y de esta forma se puedan ejecutar.

Por último tenemos el módulo de control de hardware, que se encarga del manejo de interrupciones así como la comunicación con la máquina. Los dispositivos pueden interrumpir al CPU cuando está ejecutando un proceso, si esto ocurre el núcleo del sistema debe ser capaz de reanudar el proceso que se estaba ejecutando después de atender la interrupción.

[4] *El Scheduler, es el que decide que programa de usuario se ejecutara, cuál y cuánto tiempo se deberá de ejecutar.*

3.3 Ficheros

El sistema de ficheros de Unix, permite hacer una abstracción de los dispositivos de almacenamiento de información, esto con la finalidad de que se puedan tratar a nivel lógico y poderlos utilizar de forma más sencilla que la estructura de hardware.

3.3.1 Características de ficheros

Dentro de las características de ficheros tenemos las siguientes:

- Posee una estructura jerárquica.
- Hace un tratamiento consistente de los datos de los ficheros.
- Puede crear y borrar ficheros.
- Controla y permiten el crecimiento dinámico de los ficheros.
- Protege la información de los ficheros.
- Manejar todos los dispositivos como si fuesen ficheros como son (terminales, unidades de cinta, etc.).

El sistema de ficheros esta organizado a nivel lógico con un nodo principal conocido como raíz. Cada nodo dentro de raíz es un directorio donde a su vez puede tener más nodos o subdirectorios, los cuales pueden contener ficheros normales o ficheros de dispositivo.
La ruta de un fichero determinado se puede realizar de dos formas, relativa (referido al directorio actual) y absoluta (referido al directorio raíz).

3.3.2 Estructura de ficheros

El sistema Unix puede manejar uno o varios discos físicos, donde cada uno de estos puede contener uno o más sistemas de ficheros. El núcleo trabaja con este sistema de ficheros pero a un nivel lógico es decir, no lo trabaja directamente con los discos a nivel físico.
Cada uno de los discos es considerado como dispositivo lógico dentro del sistema, los cuales tiene asociado un número de dispositivos llamados *minor number y major number*.

Estos números se utilizan en unas tablas de funciones, los cuales son indexados para poder utilizar el acceso al manejador de discos. El trabajo del manejador de disco es básicamente la transformación de las direcciones lógicas a las direcciones físicas de los discos.

La estructura del sistema de ficheros se observa de forma siguiente:

Bloque de Boot	Superbloque	Lista de nodos -i	Bloque de datos

Bloque de arranque: Comúnmente este se encuentra en el primer sector del sistema de ficheros, el cual puede contener el código de arranque. Este código es un programa que se encarga de buscar el sistema operativo para después cargarlo en memoria y así poder inicializarlo.

Superbloque: El superbloque es el encargado de describir el estado de un sistema de ficheros. Contiene información como el tamaño que ocupa, la cantidad total de ficheros que puede contener, los espacios libres, etc.

Lista de nodos i: Esta lista de nodos contiene una entrada por cada uno de los ficheros, la cual almacena una descripción de cada uno de estos. Esta descripción suele ser el tamaño del fichero, el propietario del fichero, el estado y los permisos de acceso a dicho fichero, etc.

Bloque de datos: El bloque de datos se encuentra enseguida de la lista de nodos, en esta parte es donde se encuentra el contenido de los ficheros a los que hace referencia la lista de nodos.

3.4 Procesos

3.4.1 Programas y procesos

Un programa es una serie de instrucciones y datos que se encuentran almacenados en un fichero. Este fichero tiene atributos dentro de la lista de nodos que lo identifica como ejecutable. Este puede ser ejecutado por el propietario o por otras personas ajenas al grupo dependiendo de los permisos asignados al programa.

Para poder generar un programa ejecutable es necesario realizar un mecanismo de generación, el cual se basa primeramente en crear un fichero de texto que contenga las instrucciones a realizar, conocido como código fuente. El compilador (para este caso C) deberá de traducir el código fuente a código objeto, esto para que las instrucciones de nuestro código fuente la entienda nuestra máquina, el compilador creará un fichero de salida el cual tendrá como atributo ser ejecutable.

Cuando el programa es leído por el núcleo y este es cargado en memoria para poder ejecutarse, este programa se convierte en un proceso, dentro del cual no solo existe una copia del programa si no que el núcleo añade información para poder manipular el proceso.

Cada uno de los procesos se componen de tres bloques principales los cuales se conocen como segmentos: *segmento de texto, segmento de datos y segmento de pila.*

Segmento de texto: Dentro de este segmento, se tienen las instrucciones entendibles para la máquina.

Segmento de datos: Dentro de este segmento, se tienen las diferentes variables que podrán ser inicializadas cuando se arranque el programa. Para el compilador de C, tenemos la inicialización de las variables locales y variables estáticas.

Segmento de pila : El segmento de pila es creado y administrado dinámicamente por el núcleo cuando es arrancado el proceso, este segmento está compuesto por marcos de pila (bloques lógicos), que son introducidos cuando se llama a una función y que son sacados cuando se vuelve la función.

Debido a que existen dos formas para ejecutar los procesos (usuarios y supervisor), el sistema maneja el segmento de pila por separado. El segmento de pila para usuarios, contiene los argumentos, variables globales, variables locales y algunos otros datos o funciones que solo se pueden ejecutar en modo usuario. La pila de modo supervisor contiene funciones que solo se ejecutan en modo supervisor, estas son llamadas al sistema.

3.4.2 Estado de un proceso

El tiempo de vida de un proceso se puede dividir en un conjunto de estados, donde cada uno de los estados tiene características diferentes y muy bien definidas. Los estados por los cuales debe de pasar un proceso son:

1. El proceso se está ejecutando en modo usuario.

2. El proceso se está ejecutando en modo supervisor.

3. El proceso no se está ejecutando, pero esta listo para entrar en cuanto lo indique el núcleo.

4. El proceso está cargado en memoria pero esta durmiendo.

5. El intercambiador debe cargar el proceso en memoria ya que el proceso esta listo para entrar, esta transición debe estar preparada antes de que el núcleo indique que debe ejecutarse.

6. El proceso está durmiendo y el intercambiador ha descargado el proceso para alojarlo en memoria secundaria, esto con la finalidad de crear espacio en la memoria principal y así poder cargar otros procesos.

7. El proceso está volviendo de modo supervisor a modo usuario, pero el núcleo se apropia del proceso y hace un cambio de contexto, pasando otro proceso a ejecutarse en modo usuario.

8. El proceso ha sido creado y se encuentra en una etapa de transición, el proceso existe pero no está preparado para entrar ni durmiendo. Esto pasa para todo proceso excepto para el proceso 0.

9. El proceso ejecuta la llamada ***exit*** y pasa a estado zombi, el proceso ya no existe pero deja para su proceso padre un registro que contiene el código de salida y algunos otros datos. Este estado es el estado final de un proceso.

La figura 3.3 muestra los estados de un proceso:

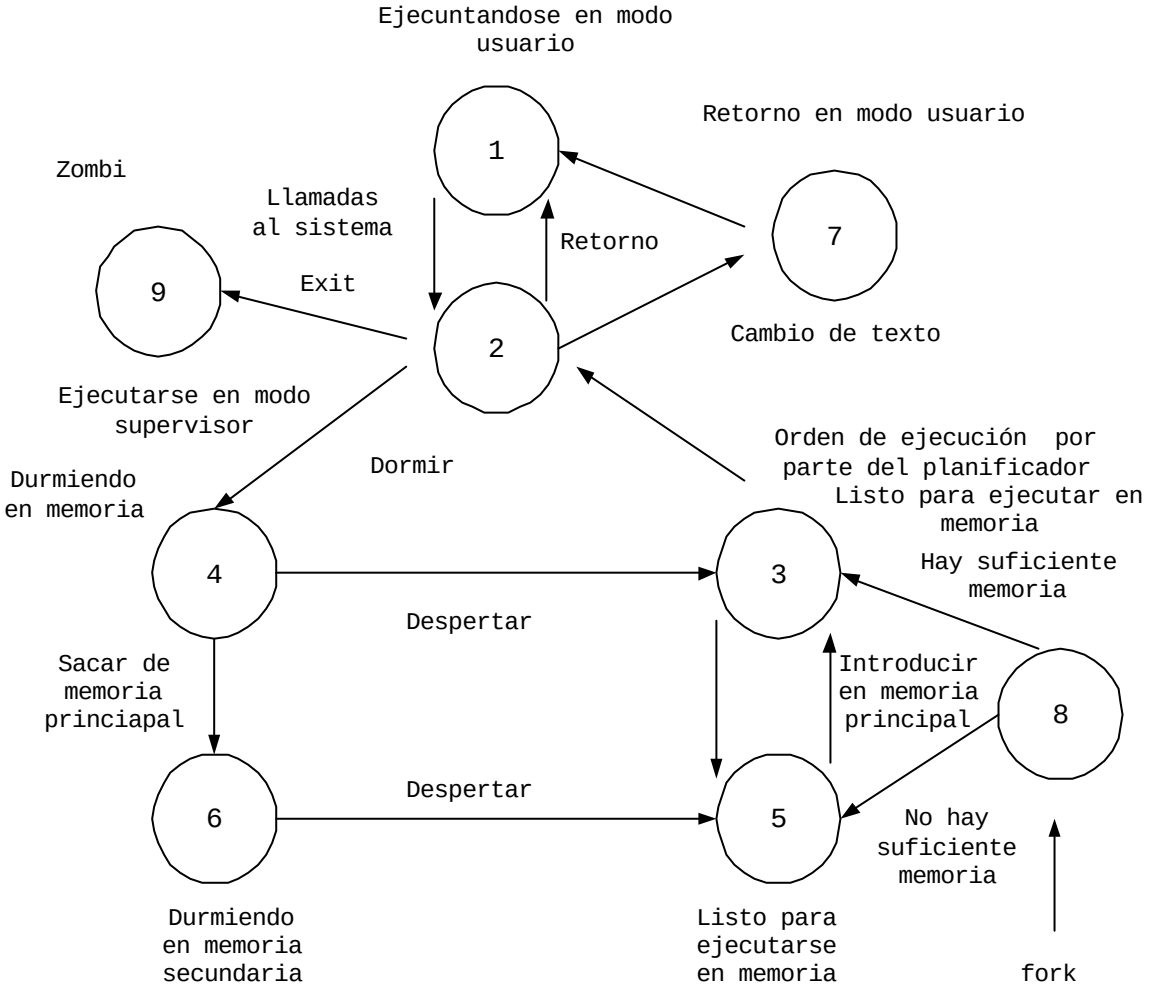


Figura 3.3 Estado de un proceso

3.4.3 Comunicación entre procesos

La comunicación entre los procesos habilita un mecanismo para que los procesos se puedan comunicar y así compartir los datos de forma sincronizada, donde el canal de transmisión será la memoria principal.

Las tuberías son una de las primeras formas de comunicación, donde éstas se pueden considerar como canales de comunicación entre procesos. Existen dos tipos: *con nombre (fifo) y sin nombre.*

Las tuberías sin nombre se crean con la llamada *pipe* y sólo podrá ser utilizada por el proceso que la creo y sus descendientes, pipe tiene la siguiente estructura:

int pipe (int fildes [2])

La tubería creada se puede manipular con el array *fildes,* los dos elementos de este array se comportarán como descriptores de archivo donde los utilizaremos para escribir y leer en la tubería. Al escribir en el array fildes[1] vamos a introducir datos en la tubería y al leer de fildes[0] vamos a extraer datos o información de la tubería. De manera análoga tenemos que fildes[1] se comporta como un fichero de solo escritura mientras que fildes[0] se comporta como un fichero de solo lectura.

La sincronización para escribir y leer de las tuberías la lleva a cabo el núcleo, de tal forma que las llamadas *read* para sacar información de la tubería no devolverá el control hasta que existan datos escritos por el otro proceso (*write*). También el núcleo es el encargado de administrar la tubería para dotarla de un mecanismo de acceso *FIFO*[5], esto para garantizar que el proceso de lectura extraiga la información en el mismo orden que el emisor haya escrito los datos.

Los datos escritos en las tuberías se gestionan en el buffer, sin que lleguen a almacenarse en el disco, por lo que al realizar la transferencia a través de memoria, las tuberías pasan a ser un mecanismo de comunicación mucho más rápido que el acceso a ficheros ordinarios.

[5] *First in first out --- primero en entrar –primero en salir*

3.4.3.1 Comunicación bidireccional

Para poder implementar esta comunicación es necesario otra tubería entre el proceso transmisor y receptor. Podríamos utilizar la misma tubería anterior, pero no es conveniente ya que nos enfrentaríamos a un problema de sincronización y tendríamos que ayudarnos de señales o semáforos para el acceso a la tubería.

La forma de implementarlo sería crear un protocolo entre emisor y receptor, de tal forma que el proceso emisor no afectara el envió de otro mensaje hasta que el receptor haya procesado totalmente el último mensaje recibido.

3.5 Shells en Unix

3.5.1 ¿Qué es un Shell?

El shell proporciona muchas de las características que hace al sistema Unix potente y flexible. Es un intérprete de órdenes, un lenguaje de programación entre otras cosas. Como intérprete el shell lee las órdenes que se le introducen y dispone de lo necesario para poder ejecutar esas sentencias. Además se puede utilizar como un lenguaje de programación para poder crear programas denominados guiones.

3.5.2 Tipos de Shells

Shell	***Descripción***
Bash	Versión GNU del shell estándar con características añadidas del shell C.
Csh	C shell
Jsh	Es una extensión del shell estándar
Ksh	Korn shell
Sh	El shell estándar de UNIX
Tcsh	Versión mejorada de shell C
zch	Un shell mejorado que se asemeja a ksh

3.5.3 ¿Que hace un Shell?

A continuación se presenta la secuencia repetida de un shell:

- El shell le solicita una orden cuando ésta dispuesta para la entrada y espera a que el usuario la introduzca.
- El usuario introduce una orden tecleando una línea de orden.
- El shell procesa la línea de orden para determinar las acciones que debe llevar a cabo.
- Después de finalizar el proceso, el shell le vuelve a solicitar una entrada al usuario para poder reiniciar este proceso.

En general una línea de orden contiene un nombre y unos argumentos, donde cada orden finaliza con un retorno.

3.5.4 Korn shell

El C shell (csh) y el Korn shell (**ksh**) fueron diseñados para poder disponer de mejores capacidades y características que sh carece, tanto el csh como ksh proporcionan un número importante de mejoras respecto al sh, tales como la edición de archivos de gran tamaño, la lista de órdenes, alias de ordenes para poder darle nombre adecuado a las órdenes, entre otras características.

3.5.5 C shell

El csh fue desarrollado como parte del sistema operativo, el cual fue el primero de los shell mejorados. Las características más importantes del csh incluyen el control de los trabajos, las listas históricas, alias de comandos para facilitar su contexto y una sintaxis similar a C para sus características de programación.
Cuando se conecta al sistema, csh lee dos archivos denominados *.login y .cshrc.* El contenido del archivo *.login* debe contener aquellas órdenes y definiciones de la variables que sólo necesitan ser ejecutadas al comenzar la sesión.

El archivo *.cshrc* contiene las órdenes de ejecución y fija parámetros. La diferencia entre estos dos archivos es que *.login* solo se lee cuando se inicia una sesión, mientras que *.cshrc* se lee desde el inicio de una sesión como cuando se le invoca desde el shell de la sesión.

3.5.5.1 Variables de C shell

El C shell proporciona variables tanto estándares definidos por el sistema y variables definidas por uno mismo. El C shell define variables con la orden **set**, la cual se utiliza de forma siguiente:

set variable = valor1

La orden para eliminar una variable es con la orden **unset.**

3.5.5.2 Variables de conmutación

Dentro de este shell, existen variables denominadas de conmutación, esto es para activar o desactivar algunas características. Algunas características son el redireccionamiento de órdenes hacia un archivo de salida y donde este archivo existe en el directorio actual, abandonar accidentalmente la sesión con alguna mezcla de teclas o si se desea un mensaje de finalización de un trabajo subordinado.

3.5.5.3 Variables de entorno

Una variable de ambiente, es una variable que está disponible a las órdenes como parte del entorno que mantiene el shell. En csh utiliza una orden para poder definir las variables de entorno tal y como lo es **setenv**. La sintaxis es:

Setenv variable valor

Como podemos observar, la diferencia entre set y setenv, es que setenv no necesita el signo (=) para unir la variable con su valor.
Para eliminar una variable de entorno, se tiene que utilizar **unsetenv**.

3.5.6 Korn shell

El Korn shell ha ido evolucionando a través de una serie de versiones que ha aumentado en potencia, añade sus propias versiones de la mayoría de mejoras que se encuentran en el C shell, así como otras muchas características potentes. Proporciona características potentes de historia y formatos de edición de línea que es compatible con los editores **vi** y **ed**. Alguna de las ventajas de trabajar con ksh, es que todos los programas realizados en este shell generalmente no requieren de modificaciones bajo sh, mientras que para ser ejecutados bajo csh, es necesario realizar algunas adaptaciones para una buena ejecución.

Similar al C shell, Korn shell utiliza dos archivos para iniciar una sesión. Ksh utiliza un archivo llamado *.profile* para poder buscar las órdenes que se quiere ejecutar así como las variables que se utilizarán dentro de cada una de las sesiones.

También lee un archivo análogamente al csh (.cshrc) que no necesariamente debe estar en alguna ruta definida ni se debe de llamar de un solo nombre, este archivo se define dentro del archivo *.profile,* el cual podemos definir el nombre y su localización.

3.5.6.1 Variables de Korn Shell

Dentro de las variables de ksh, se puede definir o redefinir dichas variables. Este shell utiliza muchas de las variables de shell sistema V, entre las que se incluyen las más importantes:

env: Donde la variable indica a ksh, donde encontrar el archivo de entorno de inicio de sesión.

histsiz: Le informa a ksh cuantas órdenes almacenar en el archivo histórico.

tmout: Esta variable define el tiempo máximo de espera antes de producirse *fuera de tiempo*[6] si no se introduce alguna orden.

[6] *Fuera de tiempo, provoca la salida del sistema después de un determinado tiempo.*

3.6 Unix

3.6.1 Versiones de Unix

La mayoría de los vendedores principales que ofrecen versiones de Unix comparten muchas de las características importantes con la Versión 4 de Unix y se ajustan a los estándares para sistemas abiertos basados en Unix Sistema V versión 4. A pesar de que existen muchas de las variantes de Unix, comparten muchas de las características.

La figura 3.4 muestra el diagrama cronológico a partir de los años 80's, donde se muestra la evolución de las variantes más importantes del sistema Unix.

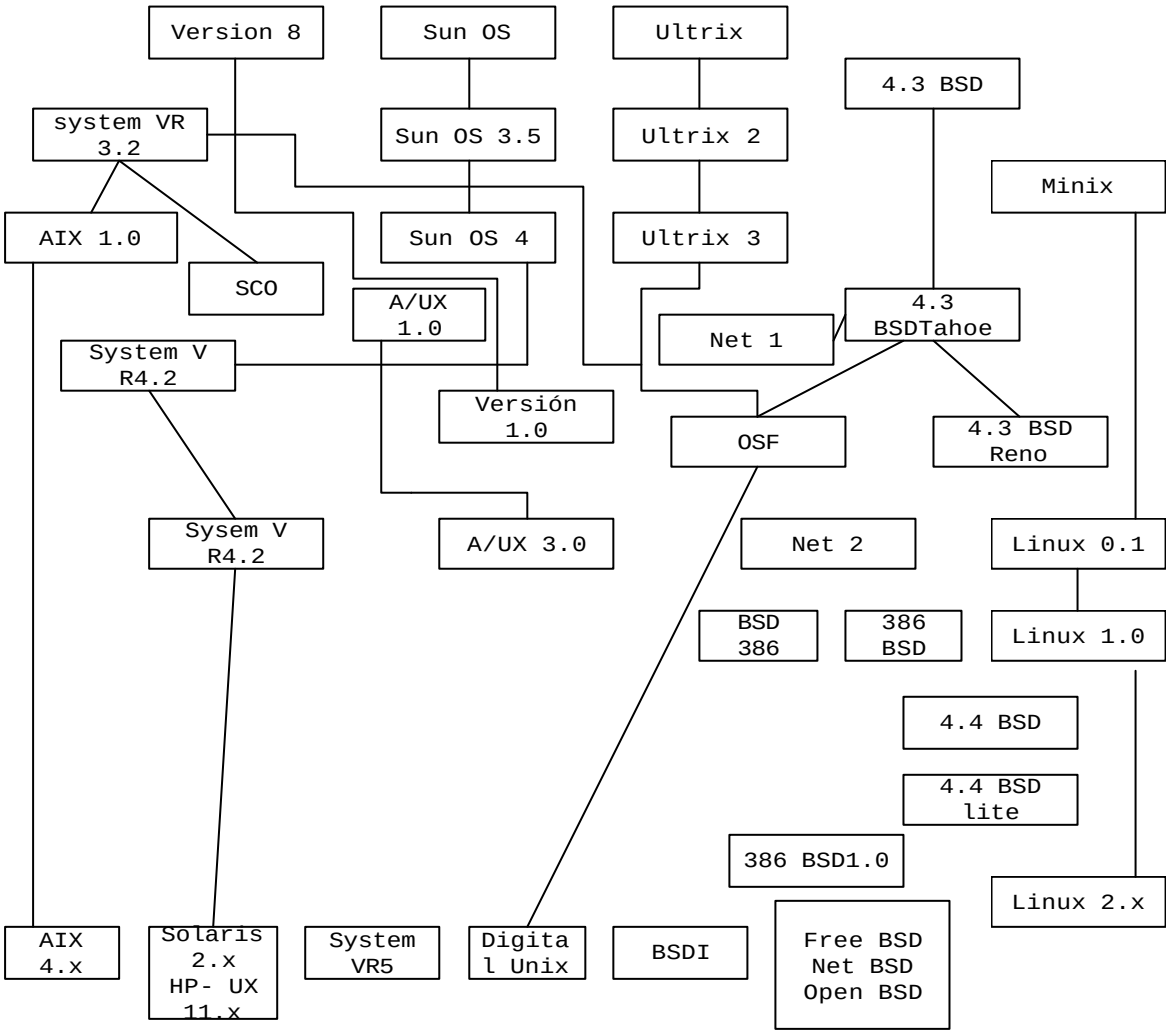


Figura 3.4 Versiones de UNIX

3.6.1.1 Unixware

UnixWare ha sido la denominación utilizada por Novell para sus productos basados en Unix Sistema V. La última versión es UnixWare 2, la cual esta basada en la integración de Unix sistema V versión 4.2 y Novell NetWare, soportando el modelo cliente/servidor.
Existen solo dos versiones; Unixware y UnixWare 2. En la actualidad Unixware sólo está disponible para procesadores Intel y para versiones posteriores que estarán basadas en arquitectura RISC.

3.6.1.2 Solaris

El sistema operativo original de Sun Microsystem fue denominado SunOS. Esta basado en Unix Sistema V versión 2 y BSD 4.3. La primera versión de SunSoft de Unix fue Solaris 1.0. Con Solaris 2.4, SunSoft se dirigió hacia un sistema basado en SVR4, la versión actual de solaris es la 5.8. En la actualidad existen versiones para procesadores Intel y SPARC.

3.6.1.3 IRIX

Irix es la versión propietaria de Unix Sistema V versión 4 que proporciona Silicon Graphics para estaciones de trabajo basadas en MIPS. La versión actual de Irix es Irix 5.3 la cual incorpora lo esencial de Unix sistema V versión 4 y 4.2. Irix fue diseñado para proporcionar funciones adicionales en muchas áreas, incluyendo soporte servidor, lanzamiento de aplicaciones y soporte de medios digitales.

3.6.1.4 HP-UX

HP-UX es la versión desarrollada por Hewlett-Packard para usar en las computadoras y estaciones de trabajo, la cual esta basada en Unix Sistema V versión 2. La versión mas reciente es HP-UX 10.0. Esta versión ha sido diseñada conforme a los estándares de Unix y HP, la cual planea ofrecer una futura versión que cumpla con Spec 1170.

3.6.1.5 DEC OSF/1

Digital Equipment Corporation (DEC), tiene una versión del sistema operativo Unix llamado Ultrix. Con la llegada de nuevas computadoras basadas en procesadores Alpha se ha desarrollado una versión variante de Unix, DEC OSF/1, basada en el sistema operativo OSF/1.
Proporciona entre otras cosas soporte para tiempo real, gestión de memoria incrementada, multiprocesamiento simétrico y rápido establecimiento del sistema de archivos.

3.6.1.6 AIX

La versión de IBM del sistema operativo Unix se denomina AIX, desarrollada para utilizar sobre estaciones de trabajo IBM. La versión 4 es la más reciente de AIX. Esta versión se basa en Unix sistema V versión 3 y tiene característica de BSD 4.3. AIX incluye soporte para MOTIF, la interfaz gráfica de usuario desarrollada por OSF e incluye una implementación parcial del Common Deskyop Enviroment.

3.6.1.7 Linux

La variante Linux del Sistema Operativo Unix fue diseñada para ser ejecutada en computadoras personales basadas en procesadores Intel 80x86. Al contrario de algunas variantes de Unix, Linux es de libre uso. Ha llegado a ser una versión popular muy extendida para uso de computadoras personales, ya que cumple con muchas características de Unix sistema V y tiene muchas ampliaciones.

3.6.1.8 Unix SCO

Santa Cruz Operation (SCO) basa sus sistemas operativos en Unix Sistema V/386 version 3.2, una versión de Unix sistema V versión 3 diseñada para procesadores Intel 80386. Los dos entornos operativos de SCO son Open Desktop y Open Server. Open Desktop ejecuta la mayoría de las aplicaciones DOS, Windows Xenin. Open Server incluye grandes capacidades de red, entre otros TCP/IP, NFS así como varios protocolos LAN.

3.7 Sistema Operativo DOS

Este SO es el más utilizado por las computadoras personales. Cuando IBM fabricó la PC, hizo que el usuario antes de cargar algún SO, realizara lo que se llamó el POST (Power On Self Test), que determinaba los dispositivos disponibles (teclado, vídeo, discos, etc.) y luego buscaba un disco de arranque. Estas funciones eran realizadas por un conjunto de instrucciones incorporadas en la máquina mediante una ROM. Luego quedó escrito que siempre hubiera algún tipo de software en el sistema aún sin ser cargado el SO.

Entre las rutinas del POST tenemos las de revisión del sistema, inicialización y prueba de teclado, habilitación de vídeo, chequeo de la memoria y la rutina de inicialización que preparaba a la máquina para ejecutar DOS. Después que las pruebas de arranque han sido ejecutadas y el sistema está cargado, la ROM aún sigue siendo importante debido a que contiene el soporte básico de entrada y salida (BIOS).

La BIOS provee un conjunto de rutinas que el SO o los programas de aplicación pueden llamar para manipular el monitor, teclado, discos duros, discos flexibles, puertos COM o impresoras.

El sistema operativo DOS es similar al sistema Unix en el diseño de su sistema de archivos, del intérprete y del lenguaje de órdenes así como el funcionamiento igual para el manejo de archivos.

3.7.1 Intérprete de órdenes

El intérprete de órdenes de DOS, command.com es equivalente al shell del sistema Unix en cuanto a que se introducen las órdenes a través de sus nombres y opciones, redireccionamiento de entrada y salida, comodines, etc. Dentro de estas, existen órdenes para crear directorios, copia de archivos y lectura de archivos.

3.8 Diferencia entre DOS y Unix

Existen algunas diferencias entre el DOS y Unix. DOS es fundamentalmente un sistema monousuario a lo que Unix es un sistema multiusuario, lo que implica que en este último sistema tenga seguridad respecto al acceso a directorios y archivos mediante los permisos.

DOS no está orientado a las comunicaciones y a redes, debido a que este sistema no soporta envío de correo o transferencia de archivos. Para poder proporcionar estos servicios es necesario comprar e instalar paquetes de software especiales para cada uno de los servicios.

Otra gran diferencia es cuando se desea interactuar directamente con el hardware, como las unidades C: y A: el caso de DOS.

La distinción del nombre de los archivos entre mayúsculas y minúsculas es algo que no es capaz de realizar DOS y trata los archivos de forma diferente y los archivos con extensión EXE tienen un tratado especial.

4.1 Concepto de socket

¿ Qué son los sockets ?

Los sockets son puntos de enlace entre máquinas, donde podemos constituirlos de la dirección IP y un número de puerto. Puede existir comunicación entre los procesos, de tal forma que los procesos pueden recibir y enviar información a través de los sockets. Existen diferentes tipos de sockets, cada uno de estos describen la forma en que se puede transferir la información. Las características principales de los sockets son el tipo de socket que se va ha crear, el dominio al que pertenece y el tipo de protocolo que se va a utilizar, también dentro de estas características entran tanto en estado en que se encuentran (si esta conectado, en estado de espera, recibiendo o enviando datos, etc.), como los apuntadores a datos tanto de recepción como de transmisión.

Para poder lograr la comunicación entre las computadoras son necesarios los sockets, los cuales trabajan en la capa de sesión del modelo OSI, estos nos permite tener un alto grado de confiabilidad de los mensajes originados en la capa de aplicación de la misma forma que la capa de enlace asegura la comunicación entre los nodos.

4.2 Tipos de Sockets

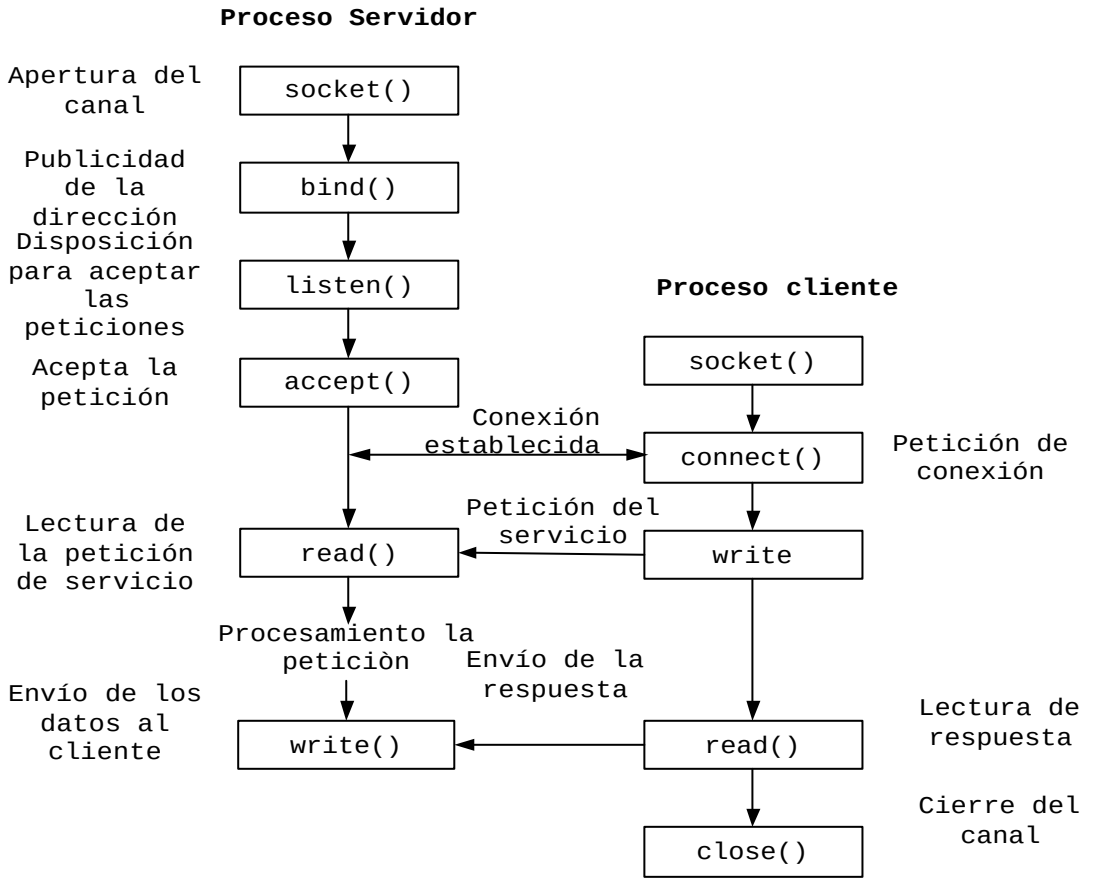
Dentro de los sockets en Internet hay muchos tipos, los cuales tienen sus propias características como son:

- Fiabilidad de datos.
- Mantenimiento del orden de los datos.
- No duplicidad en los datos.
- El modo conectado en la comunicación.
- Envió de mensajes urgentes.

4.2.1 Sock - Stream

Este primer tipo de socket, es un servicio orientado a conexión, donde los datos se transfieren sin encuadrarlos en registros o bloques. Establece comunicación a través del protocolo TCP, por lo que es necesario establecer la conexión entre dos sockets. La forma en que trabaja el socket stream es la siguiente: un socket realiza la petición de conexión (cliente) y el otro socket atiende esa petición (servidor), una vez que la conexión entre estos dos socket fue exitosa, el intercambio de información se puede llevar a cabo en ambas direcciones.

La figura 4.1 muestra la secuencia de llamadas para las conexiones TCP.

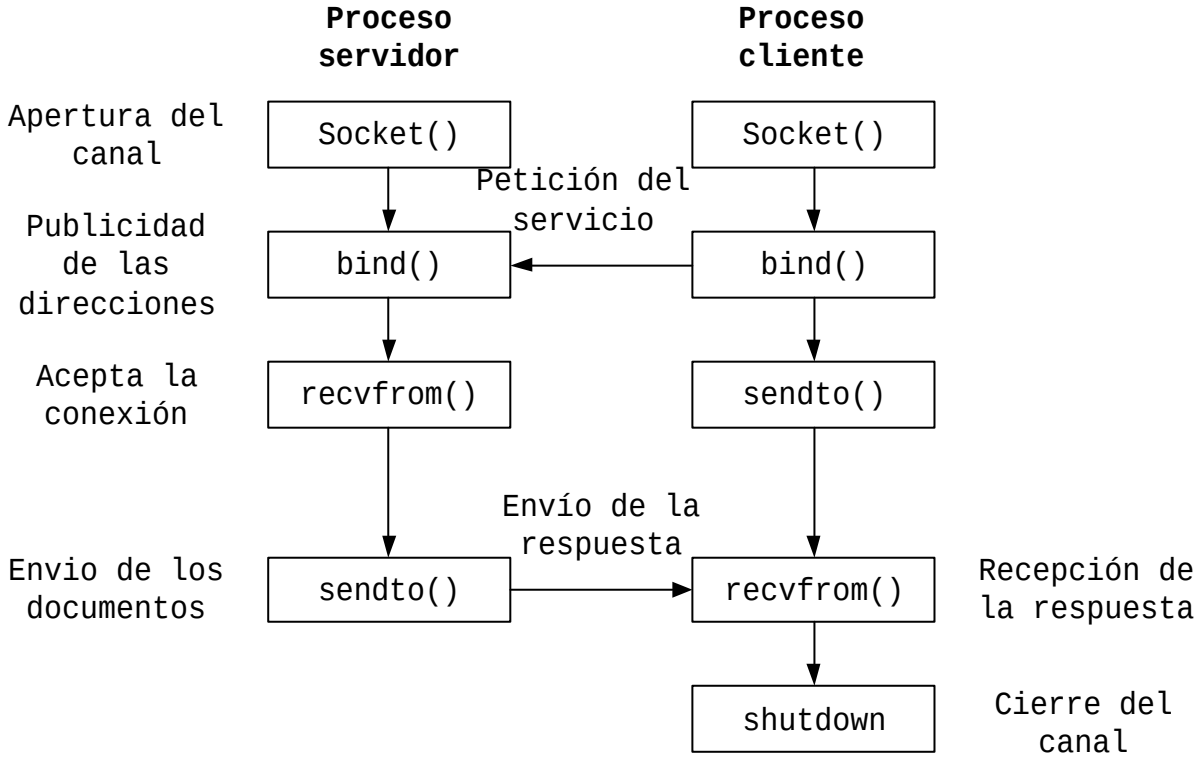


4.1 Comunicación orientada a conexión.

4.2.2 Sock - Dgram

A comparación del Socket Stream es un servicio sin conexión, la transmisión se transmite en paquetes y se recibe de la misma forma, por lo cual la transmisión de información no es confiable o los paquetes no llegan de la misma manera en que se enviaron, donde los paquetes pueden seguir rutas distintas para llegar a su destino.

La figura 4.2 muestra la secuencia de llamadas para las conexiones UDP.



4.2 Comunicación no orientada a conexión.

4.2.3 Sock-Raw

Este tipo de socket permite el acceso a protocolos de más bajo nivel como lo es IP (nivel de red).

4.3 Dominio de un sockets

El dominio de un socket, indica el formato de las direcciones que podrán tomar los sockets así como los protocolos que soportan dichos sockets.

4.3.1 AF_UNIX

El cliente y el servidor debe estar en la misma máquina. Se debe incluir la cabecera "/usr/include/sys/un.h". La estructura de la dirección de este dominio es:

```
Struct sockaddr_un {
     short sun_family;
     char sun_data;
     }
```

4.3.2 AF_INET

El cliente y el servidor pueden estar en cualquier parte de la red. Se deben de incluir las cabeceras "/usr/include/netinet/in.h"," /usr/include/arpa/inet.h" y "/usr/include/netdb.h".
La estructura de este dominio es :

```
struct sockaddr_in{
short   sin_family;          /*Familia de protocolos*/
u_short  sin_port;           /* Puerto de 16 bits*/
struct  in_addr   sin_addr;      /* Dirección IP de 32
bits*/
char  sin_zero[8];           /*Relleno de 8 ceros*/
};
```

donde la estructura in_addr se define de manera siguiente:

```
struct in_addr{
     u_long  s_addr;
};
```

Existen otros dos dominios, los cuales no son muy utilizados, por lo que solo se mencionarán. Estos son AF_NS, donde el cliente y el servidor tienen que estar en una red XEROX y AF_CCITT, para protocolos CCITT y X25.

4.4 Sockets en Unix

Se puede crear un socket llamando a la función socket, en donde su valor será un descriptor, sobre el cual se podrán realizar operaciones de lectura y escritura.
Al utilizar esta función, se lleva a cabo la inicialización de entradas a las diferentes tablas del sistema de manejo de archivos: la tabla de descriptores de procesos, la tabla de archivos y estructuras de datos conteniendo las características de los sockets de igual manera que los nodos para los archivos.

4.4.1 Función socket

La función socket se define de manera siguiente:

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

Familia debe de ser alguno de los dominios mencionados anteriormente como son:

AF_INET
AF_UNIX
…
…
…
Etc.

El servicio determina el tipo de servicio de transporte que se desea y debe de ser proporcionado por el tipo de protocolo que se encuentra dentro de ésta familia, estos pueden ser Sock-Stream, Sock-Dgram y Sock-Raw para el dominio AF_INET.

El parámetro *protocolo* determina, en el caso en que varios protocolos dentro de la misma familia presten el servicio especificado, cual de ellos utilizará el socket. Un valor nulo (0) hará que la función seleccione el valor adecuado. Los más utilizados son : IPPROTO_TCP, IPPROTO_UDP y IPPROTO_ICMP

4.4.2 Función bind

La función bind nos permite la unión de un conector con una dirección de red determinada. Su declaración es la siguiente:

*int bind (int sfd, const void *direccion, int longitud)*

Cuando un socket es creado, no tiene nombre. Hasta que un nombre es ligado al socket, los procesos no tienen forma de referenciarlo, en consecuencia ningún mensaje puede ser recibido en él. La comunicación de procesos es ligada por una asociación. En el dominio internet, una asociación está compuesta de nombre de direcciones locales y remotas, puertos locales y remotos, mientras en el dominio Unix, una asociación esta compuesta de nombre y directorios locales y remotos (la frase “nombres y directorios remotos”, significa que son creados por procesos remotos y no en sistemas remotos).

4.5 Comunicación entre sockets

Cuando se abre un conector orientado a conexión, el programa servidor indica que está disponible para recibir peticiones de conexión mediante la llamada a *listen.*

4.5.1 Función listen

La llamada a listen la suele ejecutar el proceso servidor después de la llamada a socket y bind. La sintaxis de esta función es la siguiente :

int listen (int sfd, int log)

Listen habilita una cola asociada al conector descrito por *sfd*. Esta cola se encarga de alojar peticiones de conexión procedentes de los procesos cliente. La longitud de esta cola se especifica en el argumento *log*. Para que esta función tenga sentido, el conector debe de ser de tipo SOCK-STREAM (conector orientado a conexión).
La cola de conexiones es muy importante para los servidores de tipo interactivo, por que mientras están atendiendo a un cliente pueden llegar peticiones procedentes de otros clientes.

4.5.2 Función connect

Para que un proceso cliente inicie la conexión con un servidor a través de un conector, es necesario que haga una llamada a *connect*. La forma de declarar esta función es la siguiente:

*int connect (int sfd,const void *direc, int lon)*

Donde sfd es el descriptor del conector que da acceso al canal y direc es un puntero a una estructura que contiene la dirección del conector remoto al que queremos conectarnos.

Si el tipo de conector es SOCK-DGRAM, connect especifica la dirección del conector remoto al que se le enviará información, pero no se conecta con él.

Si el tipo de conector es SOCK-STREAM, connect intenta conectar con el ordenador remoto con el objetivo de realizar una conexión entre el conector remoto y local especificado por *sfd*.

4.5.3 Función accept

Los procesos del servidor van a leer peticiones de servicio mediante la llamada *accept*. La declaración de esta función es la siguiente :

*int accept (int sfd, void *direc, int *lo)*

Esta llamada se usa con conectores orientados a conexión como lo es SOCK-STREAM. El argumento *sfd* es un descriptor del conector creado por la función s*ocket* y unido a una dirección mediante *bind*. La función *accept* extrae la primera petición de conexión que existe en la cola de peticiones pendientes creada por la llamada *listen*. Una vez extraída la petición de conexión, *accept* crea un nuevo conector con las mismas características y propiedades de *sfd,* después reserva un nuevo descriptor (*sfd*) para él.

El conector original (*sfd*) permanece abierto y puede aceptar nuevas conexiones, pero el recién conector creado (*nsfd*) no puede usarse para aceptar más conexiones.

4.5.4 Función close

Una vez que un proceso no necesita realizar más peticiones a un conector, puede desconectarme del mismo. Para ello y aprovechando que un conector es tratado sintácticamente como un fichero, podemos utilizar la llamada *close.* La definición de esta llamada es la siguiente:

int close(int sfd)

4.6 Puertos de comunicación

El puerto es un valor entero de 16 bits, e identifica un canal de comunicación para un proceso específico. Estos habilitan la comunicación entre procesos en la red TCP/IP. Los encabezados de los protocolos TCP contienen un puerto origen y destino. Los puertos son direccionados por procesos que pueden ser identificados.

4.6.1 Tipos de puertos

Los puertos no son equivalentes a otros. Ellos son clasificados en diferentes grupos con diferentes capacidades. La tabla 4.1 presenta las categorías de los puertos, su rango y una breve descripción de su categoría. Existen dos parámetros para poder comunicarse con un proceso: la dirección Internet del host y un número de puerto único asignado por el sistema.

La tabla 4.1 muestra la asignación de los puertos.

Descripción	Puertos
Puerto asignado por el sistema	0
Puertos reservados	256-1024
Puertos conocidos	1-255
Autoasignados (resvport)	512-1023
Puerto libres	1024-65535

Tabla 4.1 Descripción de puertos.

4.7 Sockets en Windows

Los winsocks fueron diseñados para ser muy similares a los sockets Berkeley. Se puede decir que los sockets Windows son sockets Berkeley con un número de extensiones Windows específicas.

Para proporcionar los servicios TCP/IP sin escribir código para una implementación de un vendedor en particular, el winsock API es llamado para la creación de una librería Winsock.DLL que los vendedores hacen disponible para las comunicaciones Windows con TCP/IP.

4.7.1 Función socket

Descripción Crea un socket.

```
#include <winsock.h>

SOCKET PASCAL FAR socket(int af, int type, int
protocol);
```

af Un formato de especificación de dirección. El único formato actualmente soportado es PF_INET, que es el formato de direcciones Internet ARPA.

type Una especificación de tipo para el nuevo socket.

protocol Un protocolo en particular para ser usado con el socket, o “0” si la llamada no desea especificar un protocolo.

4.7.2 Función bind

Descripción Asocia una dirección local con un socket.

```
#include <winsock.h>

int PASCAL FAR bind(SOCKET s, const struct sockaddr
FAR * name, int namelen);
```

s Un descriptor identificando un socket sin liga.

name La dirección que se asigna al socket.

4.7.3 Función listen

Descripción Habilita a un socket para aceptar nuevas conexiones.

```
#include <winsock.h>

int PASCAL FAR listen(SOCKET s, int backlog);
```

s Un descriptor identificando un socket sin liga y sin conexión.

backlog La máxima longitud de la cola de conexiones pendientes.

4.7.4 Función connect

Descripción Establece una conexión a un socket remoto.

```
#include <winsock.h>

int PASCAL FAR connect(SOCKET s, const struct
sockaddr FAR * name, int namelen);
```

s Un descriptor identificando un socket sin conexión.

name El nombre del socket remoto al cual el socket va a ser conectado.

namelen La longitud de name.

4.7.5 Función accept

Descripción Acepta una conexión en un socket.

```
#include <winsock.h>

SOCKET PASCAL FAR accept(SOCKET s, struct socaddr
FAR * addr,
int FAR * addrlen);
```

s Un descriptor identificando un socket el cual está escuchando conexiones después de un listen().

addr Un apuntador opcional a un buffer que recibe la dirección de la conexión. El formato exacto del argumento addr es determinado por la familia de direcciones establecida cuando el socket fué creado.

addrlen Un apuntador opcional a un entero que contiene la longitud de la dirección addr.

5.1 Modelo cliente-servidor

Dentro del mundo de las comunicaciones, el Modelo cliente/servidor es el más común para poder construir las aplicaciones en red.

Servidor. El servidor es una aplicación que se está ejecutando continuamente en un ordenador dentro de una red, el cual realizará una tarea designada o un servicio cuando un proceso cliente lo solicite.

Cliente. Es cualquier proceso que se ejecute en cualquier ordenador, este debe de estar dentro de la red y debe de ser capaz de hacer una petición a un ordenador. El mismo ordenador puede ser un cliente, es decir el ordenador puede ser al mismo tiempo cliente y servidor.
Con lo que respecta a los servidores, existen dos tipos de ellos, los interactivos y los concurrentes.

Los servidores interactivos son aquellos que se encuentran en estado de espera de una petición por parte de los clientes, en donde si le llega alguna solicitud, la atiende el mismo servidor.

Esto no es tan conveniente para los servidores que tienen varias peticiones al mismo tiempo, ya que cuando el servidor reciba más de una petición este estará atendiendo la primera, por lo cual provocaría retardos para los clientes. Los servidores concurrentes, son aquellos que crean nuevos procesos para cuando existen peticiones de parte de los clientes, así, cuando llega una petición el servidor genera un nuevo proceso, donde éste atiende a dicha petición, mientras que el servidor vuelve a estar en espera de otra solicitud.

La elección entre un tipo de servidor y otro, depende de la complejidad del servicio y el número de peticiones esperadas que vayan a producirse.

La figura 5.1 muestra el proceso para poder llevar a cabo el modelo cliente-servidor:

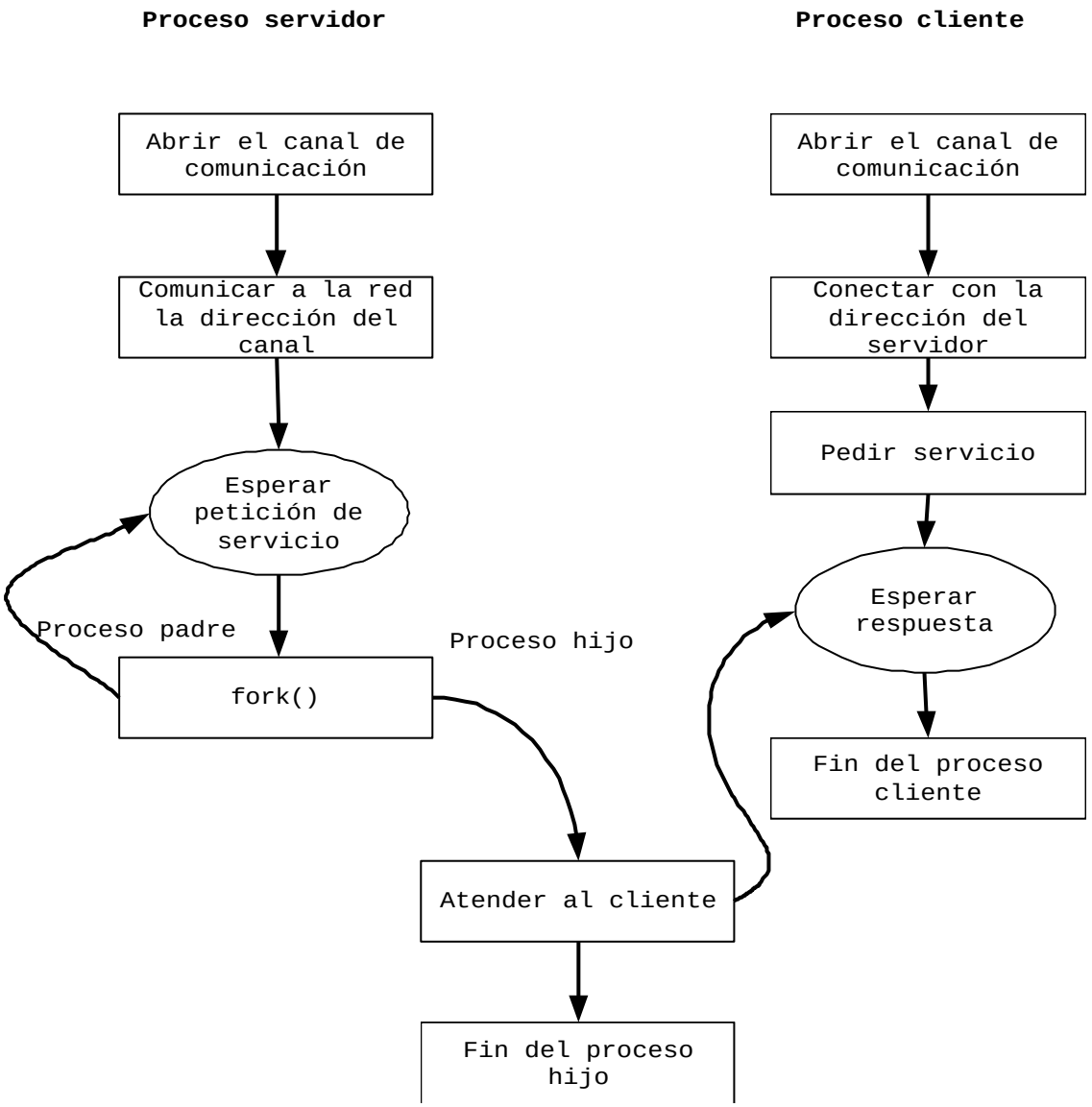


Figura 5.1 Modelo cliente-servidor

5.2 Telnet

El término Telnet designa distintos componentes que funcionan de forma conjunta, para que sea posible acceder a un ordenador remoto en modo textual vía Internet. Por lo tanto, principalmente, el protocolo Telnet proporciona el intercambio de datos entre el cliente y el servidor como si estuviera físicamente ante él mismo. Este protocolo ofrece todas las posibilidades de disponer un usuario local con el mismo derecho de acceso.

Por otro lado tenemos el software del cliente mediante el cual un usuario toma contacto con un equipo remoto y se comunica con él. Además está, naturalmente, el software del servidor que alojado en el ordenador remoto espera tomar contacto con un cliente de Telnet.

5.2.1 Establecimiento de conexión

Con lo que se refiere al establecimiento de conexión, es necesario poner a disposición la distinta información necesaria para el contacto con el servidor Telnet deseado vía TCP.
Las definiciones para llegar a tener contacto con un servidor Telnet son las siguientes:

- La dirección IP del host quién prestará el servicio Telnet
- El número de puerto al cuál queremos conectarnos.
- La terminal que se va a emular.

La dirección IP es necesaria para identificar el host adecuado. Si no existe la dirección IP será imposible establecer una conexión TCP.
La selección de una conexión consiste en definir el puerto TCP mediante el cual tiene que producirse el servicio. Si se desea utilizar el servicio Telnet, será necesario configurarlo para entrar por el puerto 23.
La selección del tipo de terminal, sirve para establecer estándares en la asignación de teclas y la determinación de determinados carácteres, por lo regular se establece como VT100.

5.2.2 Software del servidor

En el otro extremo de la conexión, el servidor tiene un servicio Telnet, conocido como “Telnet-Daemon”, donde se basa en una filosofía “Remote-Login”, es decir, una conexión remota. Unix fue diseñado especialmente para este tipo de tareas (sistema multitarea y multiusuario).
Entendemos por un sistema multiusuario, aquel equipo en el cuál pueden trabajar diversos usuarios conectados a él mediante una conexión en red.

Para poder realizar una conexión con el servidor, es necesario identificar al usuario que realice esta petición, primero solicita el nombre del usuario y su contraseña. Cuando el usuario remoto termina de identificarse, el servidor busca en una base de datos gestionada por el administrador del sistema, el usuario y su contraseña.

A menos que el usuario sea reconocido y su contraseña sea la correcta, se abrirá el intérprete de comandos. Si por el contrario el usuario y su contraseña es diferido por el sistema, por lo general, el servidor repite la solicitud hasta tres veces y si en alguna de estas peticiones no hay coincidencias, solo cierra la conexión.

5.2.3 Protocolo Telnet

Tras los clientes y servidores Telnet se encuentra el protocolo Telnet, con el que ambos se comunican entre sí. Existen esencialemente , cuatro aspectos de la comunicación:

- El intercambio de carácteres entre cliente y servidor (en ambas direcciones). En este punto se refiere a que el usuario escribe desde el cliente, la información debe ser conducido a través del software del cliente hacía el servidor vía TCP. El servidor realiza las peticiones para posteriormente procesar esta información y devolver uno o varios caracteres al software cliente, que lo muestra en la pantalla del usuario.

- Definición de un “terminal virtual”. En este punto se deriva la necesidad de contar equipos distintos vía Telnet: mainframes de IBM, Workstations de Sun, PC´s, Macs, etc.

El problema radica en que los sistemas trabajan en parte con grupos de caracteres que difieren mucho entre sí, así como la definición de las posibilidades de presentación en pantalla.

- Definición de códigos de control de significado especial. En este punto se definen algunos códigos de control para el servidor o cliente, estos no se muestran en pantalla si no que ejecutan funciones dentro del servidor o cliente .

- El manejo y configuración de operaciones de conexión. Este punto solo determina las características de rendimiento, como por ejemplo, el control de pantalla, la colocación del cursor, la definición de los campos de entrada, etc.

Para el control de este protocolo es necesario que exista comunicación con otro tipo de información entre el cliente y el servidor, por ejemplo, que el interlocutor del otro lado cancele algún proceso. Para realizar esto es necesario utilizar la combinación de algunas teclas. En Telnet, existen algunos códigos de control especiales comprendidos entre los códigos 127 y el 255, a fin de que no coincidan con los caracteres derivados del grupo de caracteres ASCII, y de hecho sólo están comprendidos entre 0 y 127. Este tipo de caracteres los puede enviar el protocolo gracias a que emplea el protocolo TCP como sistema de transporte donde este protocolo transfiere siempre flujos de bytes.

La figura 5.2 muestra el establecimiento de conexión para el protocolo Telnet.

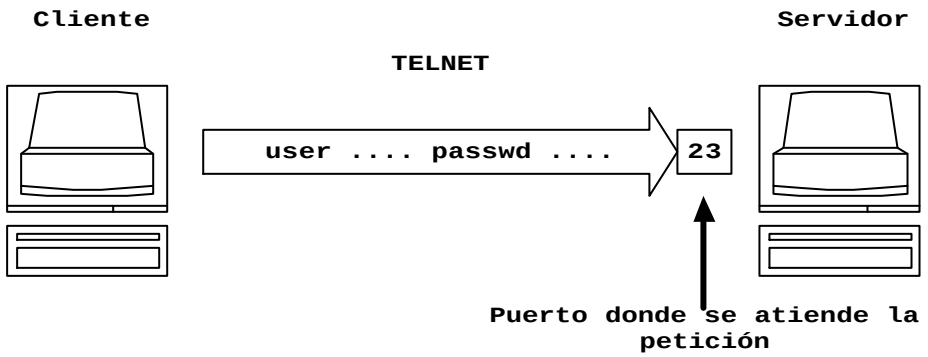


Figura 5.2 Establecimiento de conexión Telnet

5.3 FTP

El protocolo FTP (File Tranfer Protocol), está basado en la filosofía cliente-servidor. El servidor, de modo similar a un disco duro, permite cargar y guardar archivos, pero sólo cuando un cliente entra en conexión con él. Para cargar los archivos de un servidor FTP se necesita un programa especial. Este programa accede a los servicios del servidor a través de las conexiones TCP que garantiza una transmisión de datos segura. Debido a que cliente y servidor se comunican entre sí, a través de caracteres ASCII, se ha creado un protocolo que se puede implementar de manera relativamente sencilla.

5.3.1 Acceso al servidor

Después de iniciar el programa (cliente ftp), el usuario debe establecer una conexión con el servidor introduciendo un comando especial. Para ello el usuario debe de conocer la dirección IP a la cual requiere hacer la conexión, este comando se llama *open*. Una vez establecida la conexión hay que indicar el usuario y la contraseña con que se desea ingresar.

Algunos de los comandos más importantes dentro de este protocolo son los siguientes:

Comando	Descripción
ascii	Configuración del modo de transmisión ASCII.
binary	Modo de transmisión para archivos binarios.
bye	Finalizar la conexión con el servidor y salir del programa.
cd	Cambiar de directorio de trabajo en el servidor.
get [nombre de archivo]	Extraer el archivo del servidor y guardarlo en el disco local con el mismo nombre.
lcd	Cambiar de directorio de trabajo local.
open	Establecer una conexión con un servidor
quit	Finalizar la sesión FTP y salir del programa.
put [nombre de archivo]	Enviar un archivo local al servidor.
pwd	Verificar el directorio actual en el servidor.
User	Entrar al servidor con otro nombre.

5.3.2 Protocolo FTP

El protocolo FTP se apoya en el protocolo TCP para la transferencia de archivos, el cual ofrece un camino seguro y eficaz para la transferencia de paquetes e información. El protocolo FTP utiliza para la transmisión de archivos dos canales por separado, el canal de comandos sirve, exclusivamente, para la transmisión de comandos y permanece abierto durante toda la sesión FTP. Si el cliente FTP solicita un archivo, se abre una conexión adicional para datos por la que se envía el contenido de los archivos.

En la figura 5.3 se muestra la utilización de los dos canales:

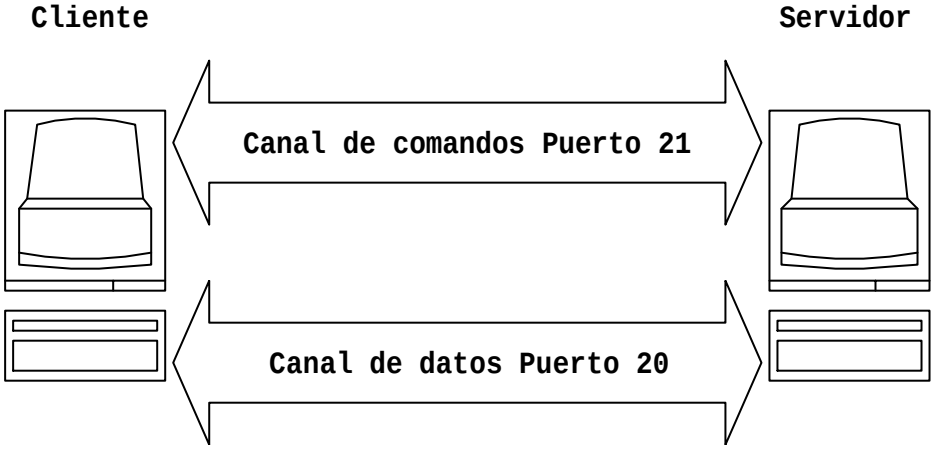


Figura 5.3 El protocolo FTP utiliza dos canales de comunicación

Los servicios de un servidor FTP solo se pueden utilizar si anteriormente se ha establecido una conexión, para esto el servidor espera las conexiones por el puerto TCP 21. Una vez establecida la conexión se abre un canal en el cual se envían los comandos que se han de ejecutar y a los que responderá el servidor.

En caso de que el cliente requiera un archivo, el servidor abre una segunda conexión TCP con el cliente, pero solo se transmitirá el contenido del archivo y cierra el archivo después de enviar el último byte.

Este protocolo define algunos comandos que sirven para distintas tareas como son:

- Registro de un usuario.
- Recepción y envió de archivos.
- Mostrar el contenido de directorios.
- Configurar los parámetros de transmisión.

5.3.3 Respuestas del servidor

El servidor contesta todos los comandos solicitados a través de una respuesta, la cual notifica al cliente si fue exitosa o no su petición. Esta respuesta está formada por un código de tres posiciones, seguido de un texto aclaratorio. Las tres cifras que preceden al texto aclaratorio permiter codificar las respuestas posibles, el protocolo FTP solo ocupa una parte de todas las posibles; estas cifras son :

- La primera cifra de una respuesta indica si el servidor confirma un comando FTP (código 2xx), si lo rechaza (código 5xx) o si el servidor necesita más información (código 3xx).
- La segunda cifra da información sobre el motivo o causa por la cual se rechazó o acepto la petición del cliente.
- La tercer cifra se utiliza para diferenciar códigos de respuesta que tienen idénticas las primeras cifras.

En la figura 5.4 se muestra la estructura de los códigos de respuesta:

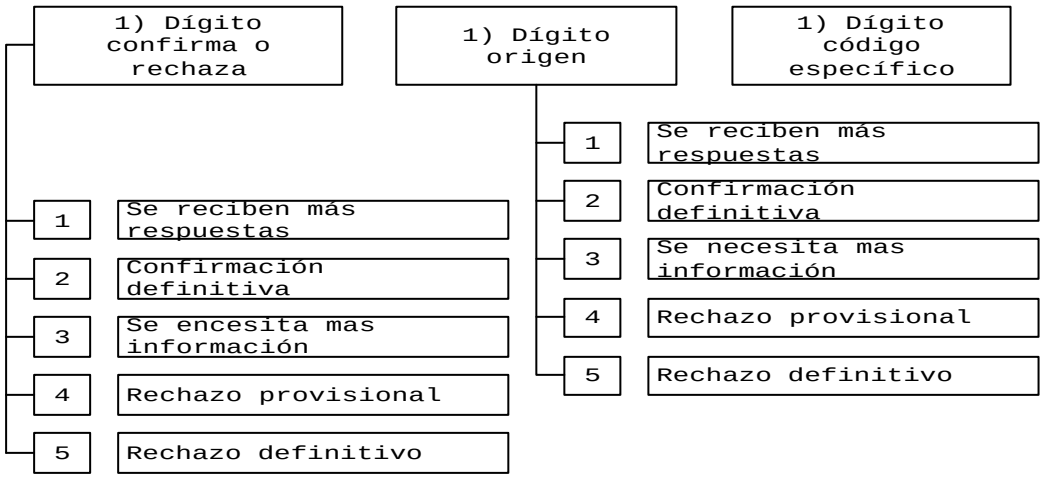


Figura 5.4 Estructura de los códigos de respuesta

La figura 5.5 muestra el establecimiento de conexión para el protocolo FTP.

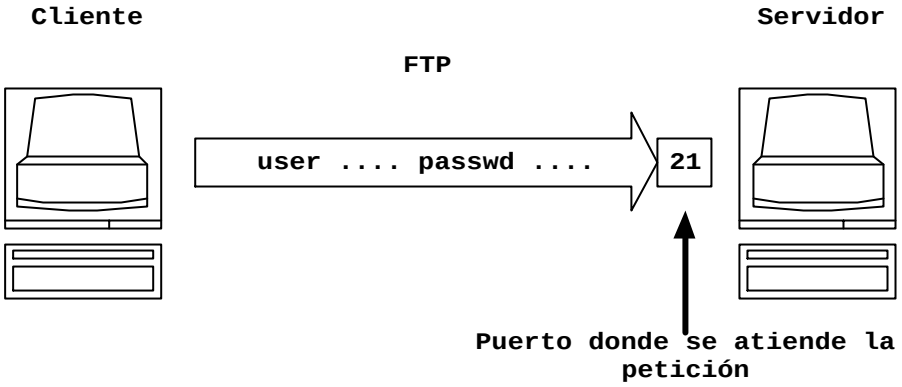


Figura 5.5 Establecimiento de conexión FTP

5.4 SMTP

El objetivo de SMTP (Simple Mail Transfer Protocol) es enviar correo de una manera fiable y eficiente. SMTP es independiente del sistema de transmisión y únicamente requiere un canal de envío de datos fiable y ordenado. Una de las características de SMTP es la capacidad de funcionar a través de los entornos de servicios de transporte. Un servicio de transporte proporciona un entorno de comunicación entre procesos, este entorno puede cubrir una o varias redes.

5.4.1 Funcionamiento de SMTP

Como resultado de una petición de envío de correo por parte del usuario, el emisor SMTP, que actúa como cliente, establece una conexión TCP con el receptor SMTP, que hace función de servidor. Este último puede ser el equipo final al cual va dirigido el mensaje o bien un sistema intermedio. El puerto definido para llevar a cabo una conexión SMTP es el 25.

Los comandos SMTP son generados por el cliente y son enviados hacia el servidor, las respuestas a dichos comandos, por su parte, funciona de manera inversa es decir, son enviadas del servidor hacia el cliente.

Cuando un host recibe un mensaje de correo proveniente de otro equipo, debe de verificar si el destinatario es él mismo o por si el contrario debe encaminar el mensaje hacia el siguiente host indicado en la lista para que éste a su vez lo encamine hasta que llegue a su destino.
Cundo un host recibe un mensaje y detecta cualquier anomalía en la dirección destino, impide así la entrega del correo, por lo que debe de generar un mensaje de error y enviarlo al emisor utilizando para ello el canal inverso utilizando el comando MAIL. Este camino puede identificar a todos los host por los cuales ha ido atravesando el mensaje hasta llegar al emisor del mensaje.

Los comandos para el protocolo SMTP son los siguientes:

Comandos	Descripción
HELO	Identificar el remitente al destinatario.
MAIL FROM	Identifica una transacción de correo e identifica al emisor.
RCPT TO	Se utiliza para identificar un destinatario individual.
DATA	Permite enciar una serie de líneas de texto.
RSET	Aborta la transacción de correo actual.
NOOP	No operación.
QUIT	Pide al otro extremo que envíe una respuesta positiva.
VRFY	Pide al receptor que confirme que nombre identifica a un destinatario válido.
TURN	El emisor solicita que se inviertan los papeles, para actuar como receptor.
SOML	Si el destinatario esta conectado, entrega el mensaje directamente a la terminal.
SAML	Entrega del mensaje en el buzón del destinatario.
SEND	Si el destinatario está conectado, entrega el mensaje directamnete a la terminal.

La figura 5.6 muestra el establecimiento de conexión para el protocolo SMTP.

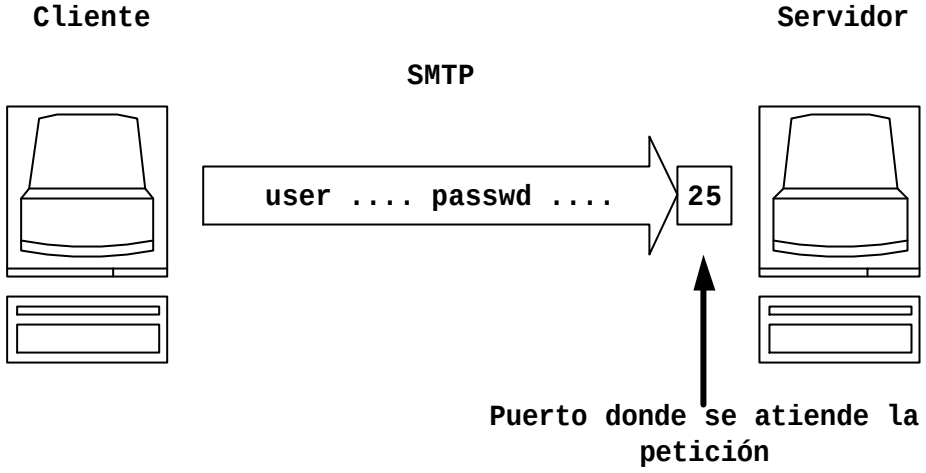


Figura 5.6 Establecimiento de conexión SMTP

5.5 Web

La tarea principal del navegador Web es seguir instrucciones del usuario, el cual introduce la dirección de la página Web que quiere cargar o bien si pulsa uno de los enlaces de la página que se presenta para traer a pantalla la página a la que hace referencia el enlace.
Los servidores Web son el lugar en donde acuden los usuarios que necesitan cierta información que esta almacenada en el servidor correspondiente.

La mayoría de los contenidos de la World Wide Web se encuentran en forma de páginas HTML es decir, como simples archivos de texto formulados en base estándar HTML. El navegador del cliente debe de ser capaz de interpretar el texto HTML, esto no se aplica al servidor, ya que para éste, el texto no es más que un recurso que ha de enviar al navegador, en la mayoría de los casos.

5.5.1 Como funciona la Web

El sistema de Web está basado en tres conceptos principales:

- HTTP , el Hipertext Tranfer Protocol, es el encargado de entender entre el programa cliente, quién llama las páginas y los elementos de la World Wide Web y el programa servidor, que proporciona a petición del cliente estos elementos.
- HTML , el Hipertext Markup Language, se utiliza como lenguaje de descripción y formato para las páginas de la World Wide Web.
- URL`s , son los Uniform Ressource Locators, que se utiliza como un mecanismo estandarizado para dar nombre a las páginas y elementos de la World Wide Web asígnandole un título y una ruta de acceso único.

Las funciones de cada uno de estos tres elementos es diferente y existe una gran división de trabajo.
HTTP es el responsable de transportar los contenidos a traves de la Web. HTML se encarga de la representación y los URL’s se encargan de formular las direcciones de manera única.

CGI

El interfaz de pasarela común hace de intermediario entre los servidores Web, las bases de datos externas y otras fuentes de información. Los CGI o programas pueden considerarse una extensión de la funcionalidad básica de un servidor Web. Las aplicaciones CGI realizan tareas específicas de procesamiento, extracción y formateo de información, define un método para que el servidor Web utilice programas adicionales y servicios que puedan usarse para acceder a aplicaciones externas desde el contexto de un documento Web activo.

Los scripts pueden diseñarse y contruirse para una gran variedad de própositos. Algunos de los mas comunes incluyen los siguientes:

- Recoger opiniones de los usuarios sobre una línea de productos mediante formularios HTTP.
- Conversión dinámica de documentación del sistema a documentos HTML para fácil acceso a tráves del Web.
- Consulta de bases de datos y mostrar resultados como documentos HTML.

Las especificaiones CGI permiten dos tipos HTTP para manejar la entrada CGI: GET y POST. La única diferencia entre el metodo GET y POST es la forma en que se pasa la información a la aplicación CGI.

Método GET

Con el método GET, se pasa la información en la línea de comando, lo cual significa que necesita ser corta (algunos sistemas restringen la longitud de la línea de comadno, lo que produce que se trunquen líneas de comando largas; en la mayoría de los sistemas UNIX, por ejemplo, esta restricción es de 128 o 256 carácteres).

Este método instruye al servidor sobre la recuperación de datos referenciados en un URL, donde el URL especifica una aplicación CGI.

Método POST

Usando el método POST, que es ahora fuertemente recomendado para todas las aplicaciones excepto en simples consultas, la información de entrada se codifica en un bloque de datos que se pasa a través de la entrada estándar a la aplicación CGI. Este método evita todas las restricciones en la longitud de la línea de comando y ofrede un mecanismo de transmisión de los datos mucho más fiable entre los formularios y aplicaciones CGI.

El método POST proporciona las siguientes funciones a un cliente Web:

- Enviar un mensaje a una tala de anuncios, lista de correos o un grupo de noticias.
- Obtener información sobre recursos existentes en el servidor.
- Proporciona datos de un formulario a una aplicación CGI.

La tabla 5.7 muestra los mensajes mas comunes en Internet:

Mensaje	Descripción
400	Solicitud errónea de fichero
401	No autorizado
403	Acceso denegado
404	Fichero no encontrado
408	Excedido el tiempo de espera
500	Error interno
502	Servicio temporalmente sobrecargado
503	Servicio no disponible
505	Versión de HTTP no soportada
Failed DNS Lookup	Falló la búsqueda DNS
Host Unavailable	Servidor no disponible

Tabla 5.7 Mensajes comunes en Internet

5.5.2 El protocolo HTTP

A lo mismo que los demás protocolos de Internet, HTTP es un protocolo que se encarga de establecer una conexión TCP segura entre cliente y servidor siempre y cuando se haya establecido un intercambio de datos anteriormente. Como punto de conexión esta definido el puerto 80, éste es el puerto que utiliza un servidor HTTP para esperar las posibles conexiones por parte de los clientes.

A diferencia de otros protocolos, el cliente establece una conexión con el servidor HTTP deseado a través del puerto 80 y envía al servidor una petición de un determinado documento. El servidor recibe la petición, la evalúa y posteriormente envía el documento solicitado. Este protocolo se conoce también como protocolo sin estado ya que la conexión no pasa por varias fases como: login del cliente, intercambio de datos y el cierre de conexión por parte del mismo cliente.

5.5.3 URL's

Una condición necesaria para que el acceso a los documentos de la Web sea fructuosa es, que exista una dirección única para cada uno de los documentos, mediante la cuál se puede informar al servidor a que documento se desea accesar, internet utiliza el localizador uniforme de recursos, "Uniform Resource Locators".

Dentro del marco de los URL´s, lo primero que se nombra es el protocolo o el servicio que se quiere utilizar, a continuación el servidor deseado y por último la fuente deseada del servidor incluyendo la ruta de acceso. La estructura de un URL-HTTP es:

http://<host>:<puerto>/<ruta>

El prefijo http:// identifica el URL como referencia HTTP y se distingue de otras como ftp, telnet, etc.
Viene a continuación el nombre del host que presta el servicio agregando el puerto y separado por ":". Por último el nombre del dominio del servidor donde se encuentra el recurso en cuestión.

La figura 5.8 muestra el establecimiento de conexión para el protocolo HTTP.

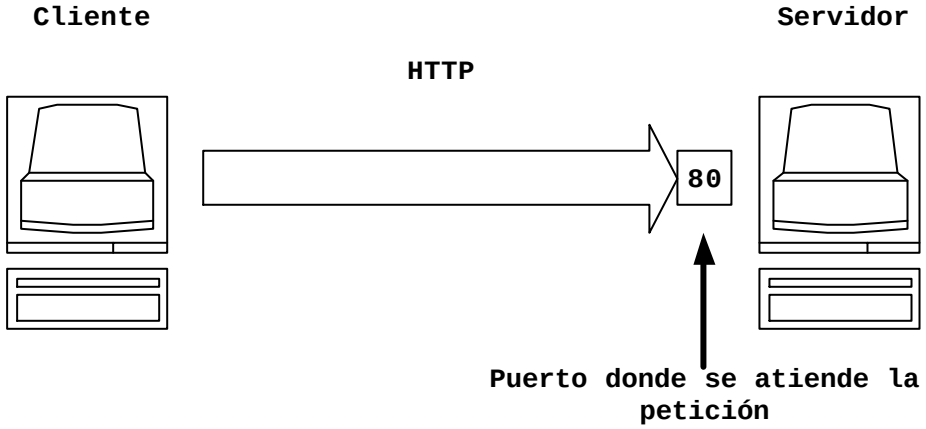


Figura 5.8 Establecimiento de conexión HTTP

5.6 Secure shell

Cuando se realiza una conexión a un servidor remoto utilizando algún servicio como FTP, Telnet, etc; el login (usuario) y passwd (contraseña) se transfieren de forma clara a través de la red, lo cual representa un alto riesgo si existe un programa que capture la información que pasa en la red (comunmente llamado sniffer) con lo que se podría hacer mal uso de los recursos del servidor.

Secure shell permite realizar una comunicación y transferencia de información de forma cifrada proporcionando fuerte autenticación sobre un medio inseguro. Es un programa que permite conexiones entre máquinas a través de una red abierta de forma segura así como, ejecutar programas en una máquina remota y copiar archivos de una máquina a otra.

5.6.1 Algoritmos cifrados

Este programa provee una fuerte autenticación y comunicación por un canal inseguro y se desarrolla con la finalidad de reemplazar a algunos protocolos como FTP, Telnet, Rlogin, etc.
Secure shell admite varios algoritmos de cifrado entre los cuales están:

- Blowfish
- 3DES
- IDEA
- RSA

5.6.2 Protocolo SSH

Existen dos protocolos desarrollados sobre ssh, SSH1 y SSH2. Este segundo provee licencias mas estrictas que SSH1, ya que es de próposito no comercial que al contrario de SSH2 que es para próposito comercial. La última versión de Secure Shell cliente-servidor para Unix con este protocolo es la 3.0.0.

OpenSSH es una versión libre de los protocolos SSH bajo licencia BSD y es totalmente compatible con los protocolos SSH.

La figura 5.9 muestra el establecimiento de conexión para el protocolo SSH.

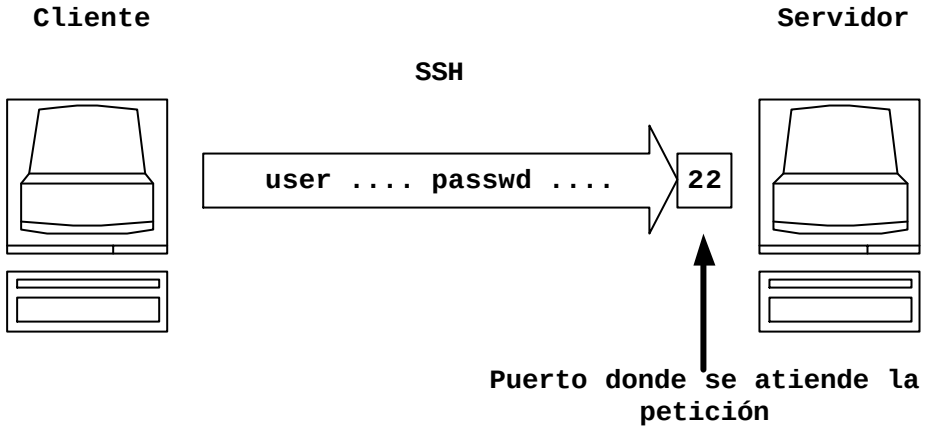


Figura 5.9 Establecimiento de conexión SSH

5.7 Finger

Es un programa de Unix que sirve para poder determinar que usuarios estan conectados a una máquina específica. Si se desea saber quienes estan conectados a un servidor determinado se utiliza este comando seguido del nombre completo del servidor.
Por ejemplo:

```
>> finger @sistemas.dgbiblio.unam.mx
```

No todas las máquinas aceptan el comando finger, esto lo restringen los administradores de red para seguridad de los usuarios.

La figura 5.10 muestra el establecimiento de conexión para el protocolo Finger.

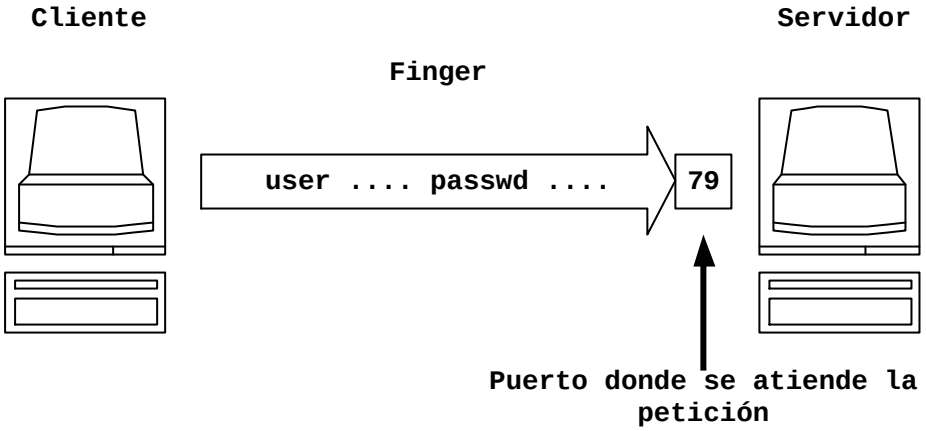


Figura 5.10 Establecimiento de conexión Finger

El siguiente capítulo tiene como finalidad mostrar la investigación realizada en el desarrollo e implementación de una herramienta para el monitoreo de una red, esto con la finalidad de tener continuamente vigilados los servicios utilizados. Dicho sistema se encargara de brindar toda la información generada por las aplicaciones conectadas a un servidor utilizando el protocolo TCP/IP.

Proporciona información respecto al tráfico que fluye a través del servidor de forma cuantitativa, esto se genera a medida que se van capturando los paquetes que entran al servidor.

Existen dos tipos de monitores: los activos y los pasivos. Los monitores activos son aquellos que notifican al administrador sobre los sucesos que se presentan, así el administrador puede tener conocimiento respecto a los problemas que se tienen en el servidor y así dar una respuesta rápida a dichos acontecimientos. El monitor pasivo es aquel que registra un incidente o un acontecimiento en un archivo conocido como bitácora, registra la información del tráfico que fluye a través del servidor, así el administrador puede ingresar a la bitácora en cualquier momento que lo requiera.

El desarrollo de este monitor será de forma pasiva, ya que registrará toda la información generada por las aplicaciones TCP/IP conectadas al servidor. La bitácora tendrá información sobre la dirección IP, numero de puerto por el que se ingreso, servicio solicitado, así como la información transferida. El programa se puede ejecutar desde cualquier máquina dentro de la red por el administrador o se puede ejecutar desde el servidor que se quiere monitorear.

Se eligió el estándar Ethernet debido a que en la actualidad es el protocolo más comercial, por su costo y fácil implementación, pero sobre todo por que soporta las redes TCP/IP. Toda red Ethernet es una red tipo Banda Base que ocupa código Manchester y por ello sólo un canal puede transmitir usando todo el ancho de banda del medio. Cuando dos o mas estaciones de trabajo transmiten al mismo tiempo ocurre una superposición de señales que interfieren todas las comunicaciones causando errores de transmisión hacia todas las estaciones.

Por esta razón existe un protocolo que permite coordinar a todas las estaciones de trabajo para que ocupen el canal en forma eficiente, este protocolo es llamado CSMA/CD (Carrier Sense Multiple Acces / Collition Detection).

La figura 6.1 indica los niveles en los cuales se encuentran los protocolos de comunicación:

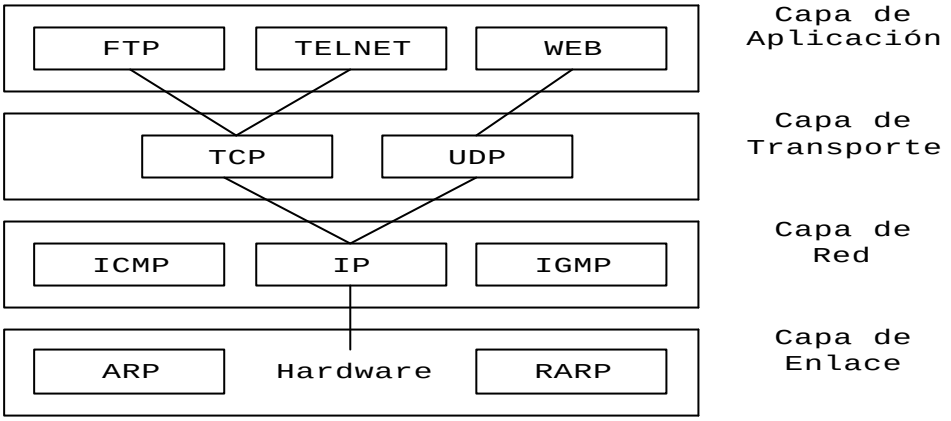


Figura 6.1 Nivel de protocolos

El desarrollo está basado en los paquetes capturados en el nivel de enlace de red, dentro del cual se encuentra el protocolo IP, limitándose al protocolo inferior TCP. Los datos que se encuentran dentro de este paquete serán de utilidad para poder analizar los servicios solicitados por parte de los clientes así como la información .Por esta razón será necesario explicar cada una de las partes que constituyen a los paquetes IP.

La figura 6.2 muestra la estructura del paquete IP para posteriormente analizar cada una de las partes que lo conforman.

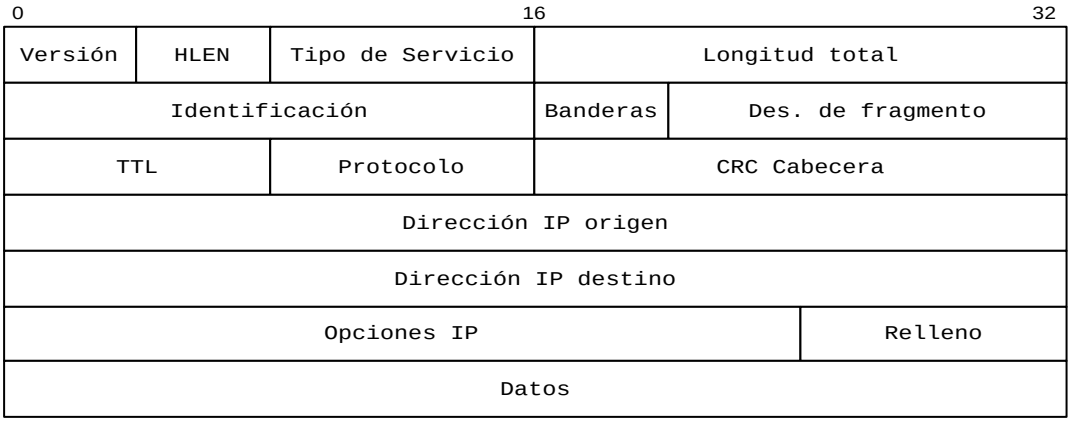


Figura 6.2 Estructura del paquete IP

Versión (4 bits) : Indica la versión del protocolo IP que se utilizo para crear el datagrama. Actualmente se tiene la versión IPv4.

HLEN (4 bits) : Longitud de la cabecera en bytes del paquete.

Tipo de servicio (8 bits) : Determina la prioridad de un paquete y las características de la vía de transferencia.

Longitud total (16 bits) : Longitud total en bytes del paquete, cabecera y datos útiles.

Identificación (16 bits) : Valor del emisor que ayuda a identificar los fragmentos de un paquete IP.

Banderas (3 bits) : Hacen referencia a la fragmentación de un paquete IP.
Desplazamiento de fragmento (13 bits) : Indica el lugar donde se insertará el fragmento actual dentro del datagrama completo.

TTL (8 bits) : Contador de la vida de un paquete que decrece cada vez que pasa por un router.

Protocolo (8 bits) : Clase de paquete contenido en ese paquete IP, este sirve para desglosar y cruzar protocolos superiores (TCP, UDP, etc.)

CRC Cabecera (16 bits) : Contiene la suma de comprobación de errores solo para la cabecera del datagrama.

Direccion IP origen (32 bits) : Contiene la dirección IP del origen.

Direccion IP destino (32 bits) : Contiene la dirección IP del destino.

Opciones IP : Opciones IP en la medida que se necesitan.

La figura 6.3 muestra la estructura de un paquete ARP, este se encuentra en la capa de enlace de datos, un nivel superior al protocolo IP.

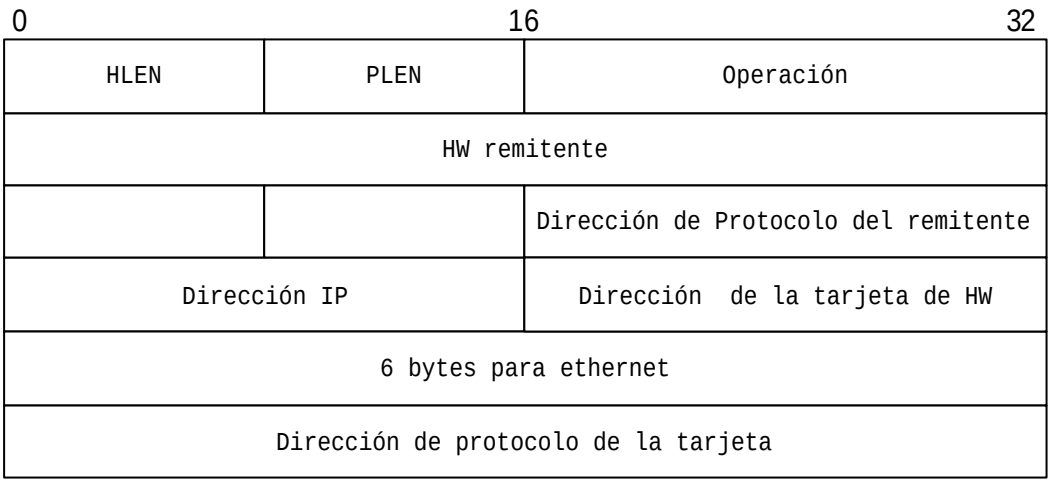


Figura 6.3 Estructura del paquete ARP

HLEN : Longitud de la dirección hardware.

PLEN : Longitud de la dirección de protocolo.

Operación : Indica si es mensaje de consulta o respuesta.

HW Emisor : Dirección física del emisor.

Dirección IP : Dirección IP del emisor.

HW destino : Dirección física del destino.

IP destino : Dirección IP destino.

6.1 Descripción del sistema

El sistema funciona por medio de un programa, el cual se tiene que ejecutar en la máquina que se desea monitorear, es conveniente que su ejecución sea desde consola, esto por seguridad del administrador del equipo aunque se puede correr desde cualquier otra máquina con acceso a internet y que tenga permisos al servidor.

Para correr el programa es necesario ingresar al sistema como usuario root, esto es por que en la programación se utiliza una rutina para verificar quién está corriendo el programa, en caso de que el usuario no halla ingresado como administrador el programa mandará un mensaje notificando para que el programa pueda ser ejecutado es necesario ingresar al sistema como superusuario.

Una vez que el programa este en ejecución, capturará todos los paquetes con conexión TCP que se transfieran al servidor, analizará el número de paquetes en un determinado lapso de tiempo y llevará una estadística de transferencia de paquetes. Esta información se almacenará en un archivo de datos en el mismo directorio en donde se ejecute el programa. En la contabilización del número de paquetes que llegan al servidor, no importa el servicio que se esté solicitando por parte del cliente, aquellos servicios que estén contemplados en la programación serán considerados por el sistema.

En la captura de los paquetes se analiza toda la información que se está transfiriendo hacia el servidor, donde lo más importante es el servicio que esta solicitando (telnet, ftp, secure shell, etc.) y la dirección IP fuente.
La identificación de los servicios solicitados por los clientes, se realiza a través de una parte que forma el paquete este es, el puerto destino.

El puerto destino está formado por 4 bytes que al decodificarlo se puede realizar la comparación con los puertos programados y de esa forma llegar a la traducción de ese puerto.

En la figura 6.4 muestra el diagrama de flujo para analizar el servicio solicitado por parte del cliente.

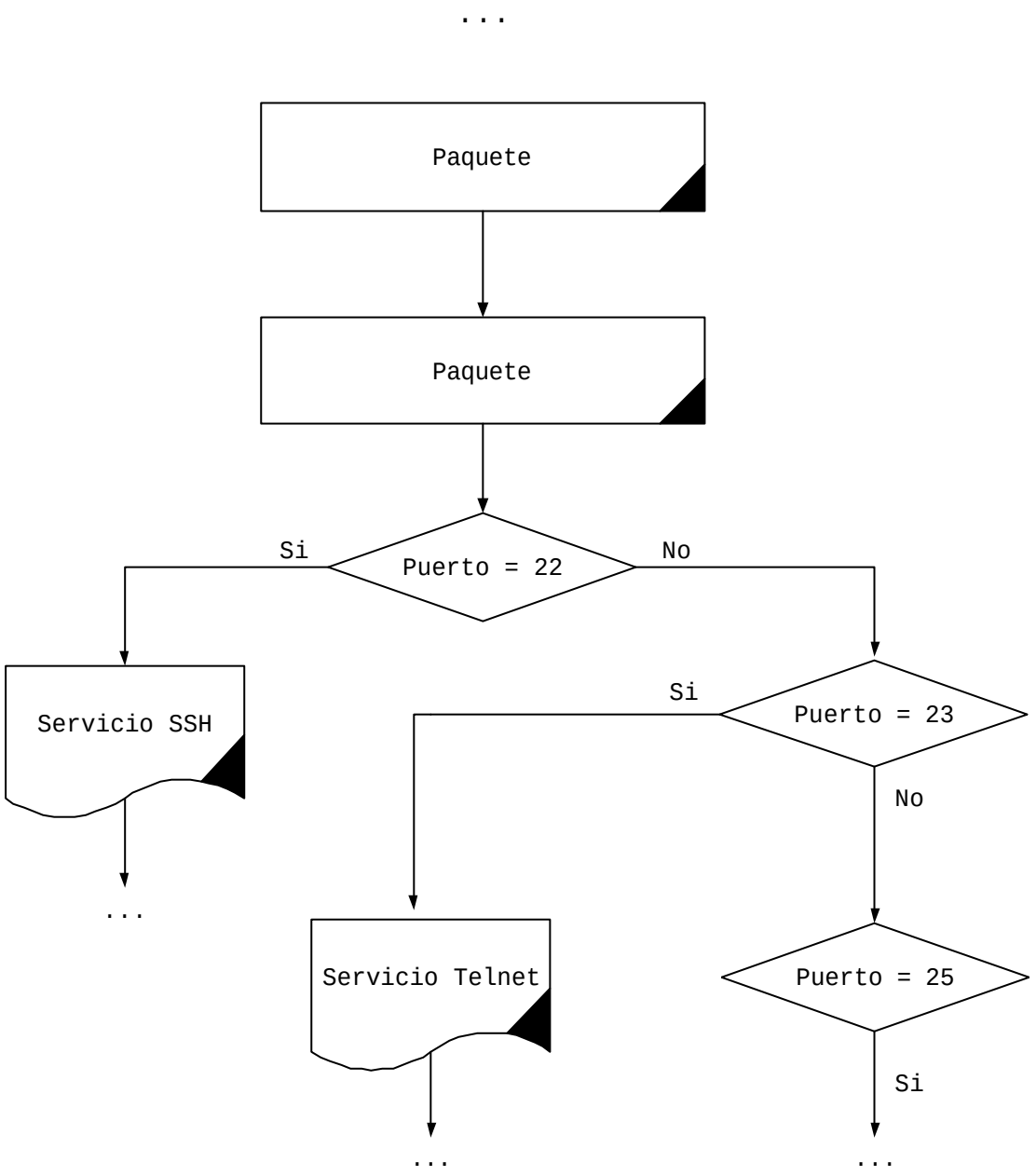


Figura 6.4 Diagrama de flujo para analizar los puertos.

Cuando se analizan los paquetes, es necesario conocer la información contenida dentro de ellos como son :

- Dirección IP origen del paquete.
- Dirección IP destino del paquete.
- Número de puerto fuente.
- Número de puerto destino.
- Información contenida en el paquete.

Por otra parte, la visualización de la información contenida en los paquetes se reflejará en el archivo "tcp.log", el cual contendrá el sistema operativo del cliente que está solicitando algún servicio, la dirección IP fuente, servicio solicitado y la información transferida. La información se visualizará hasta que la información contenga un " ↵ ", con esto se asegura que el comando requerido por parte del cliente ha finalizado y espera la respuesta del servidor.
Se pretende conocer esta información de forma siguiente:

```
Dirección IP Cliente  : 192.168.10.14
Sistema Operativo     : Windows
Servicio              : Telnet
Información           : rm *
```

Cuando existe una petición por primera vez, es necesario registrar la hora en que inicia una nueva sesión, así como la hora de finalización de la misma para posteriormente determinar el tiempo que este cliente estuvo conectado al servidor.

En caso que ya exista la conexión establecida, solo es necesario analizar y almacenar la información transferida en la bitácora para después mostrarla en pantalla.

6.2 Análisis gráfico

La figura 6.5 muestra el análisis gráfico de los clientes con los diferentes sistemas operativos realizando solicitud de peticiones al servidor.

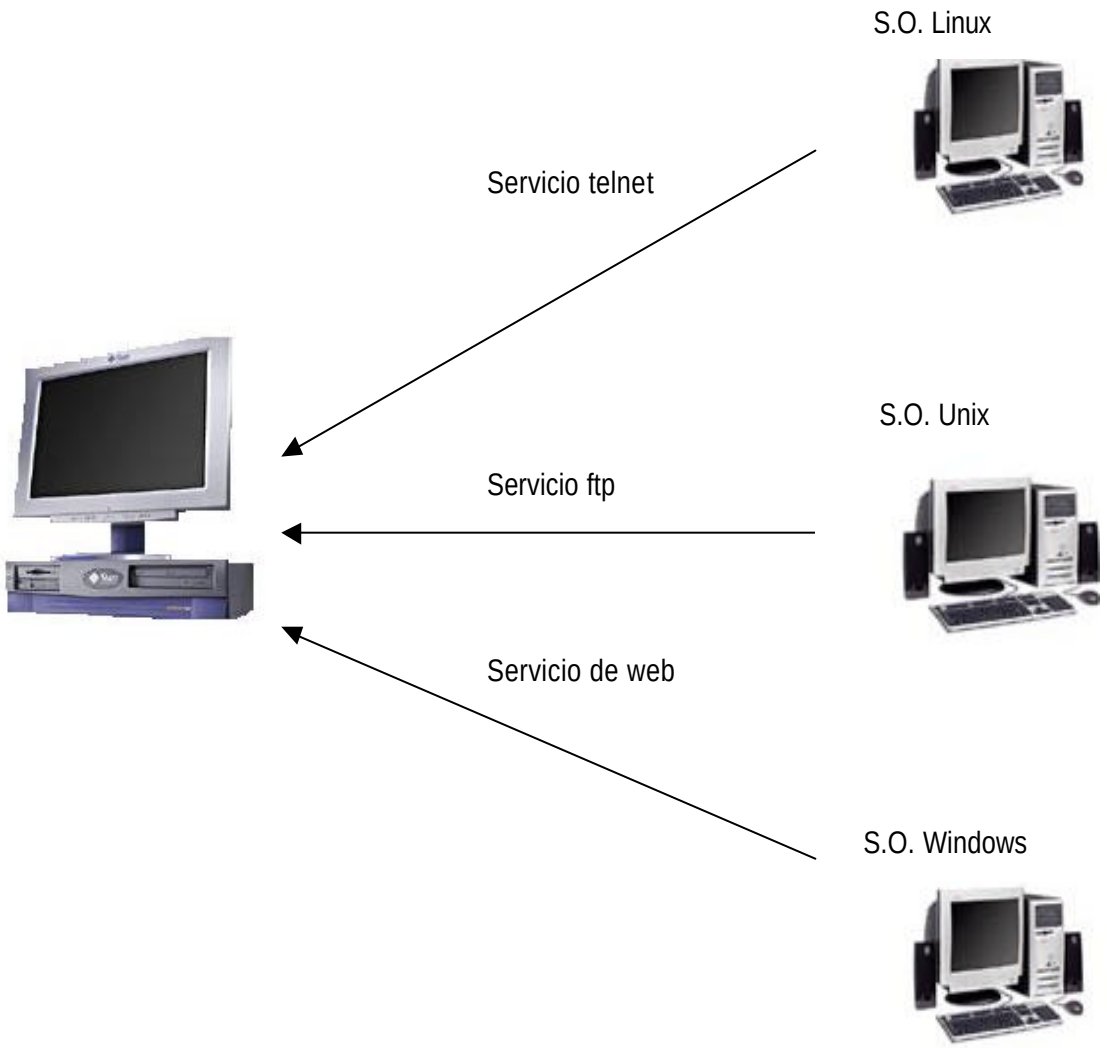


Figura 6.5 Análisis gráfico

6.3 Análisis técnico

La figura 6.6 muestra el análisis técnico y conceptual que se tiene para poder determinar el monitoreo de los paquetes.

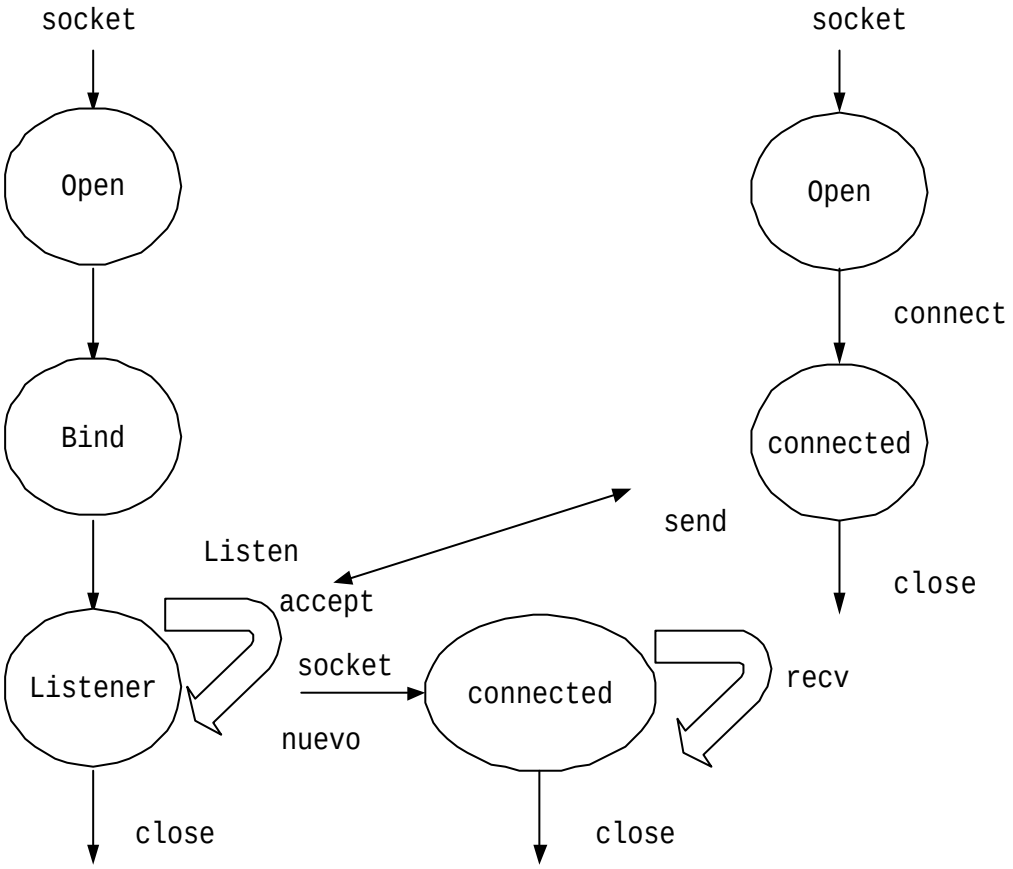


Figura 6.6 Análisis técnico

En la Figura 6.6, se tienen las conexiones TCP, en donde el cliente primero solicita una petición de servicio al servidor, el servidor acepta o rechaza la conexión, en caso de que la conexión sea aceptada, entonces la transmisión de datos puede empezar.

El servidor es el encargado de escuchar las peticiones que se le hagan por parte de los clientes para así interpretar la información recibida y actuar de acuerdo a ella, la forma de trabajar departe del servidor dentro de la programación se describe a continuación.

6.4 Desarrollo

Dentro de la etapa del desarrollo, el sistema se constituye de un programa escrito en lenguaje C para poder capturar los paquetes en la capa de red (protocolo IP) que entran al servidor para un análisis posterior.

La figura 6.7 muestra el diagrama funcional para capturar paquetes en la capa de red.

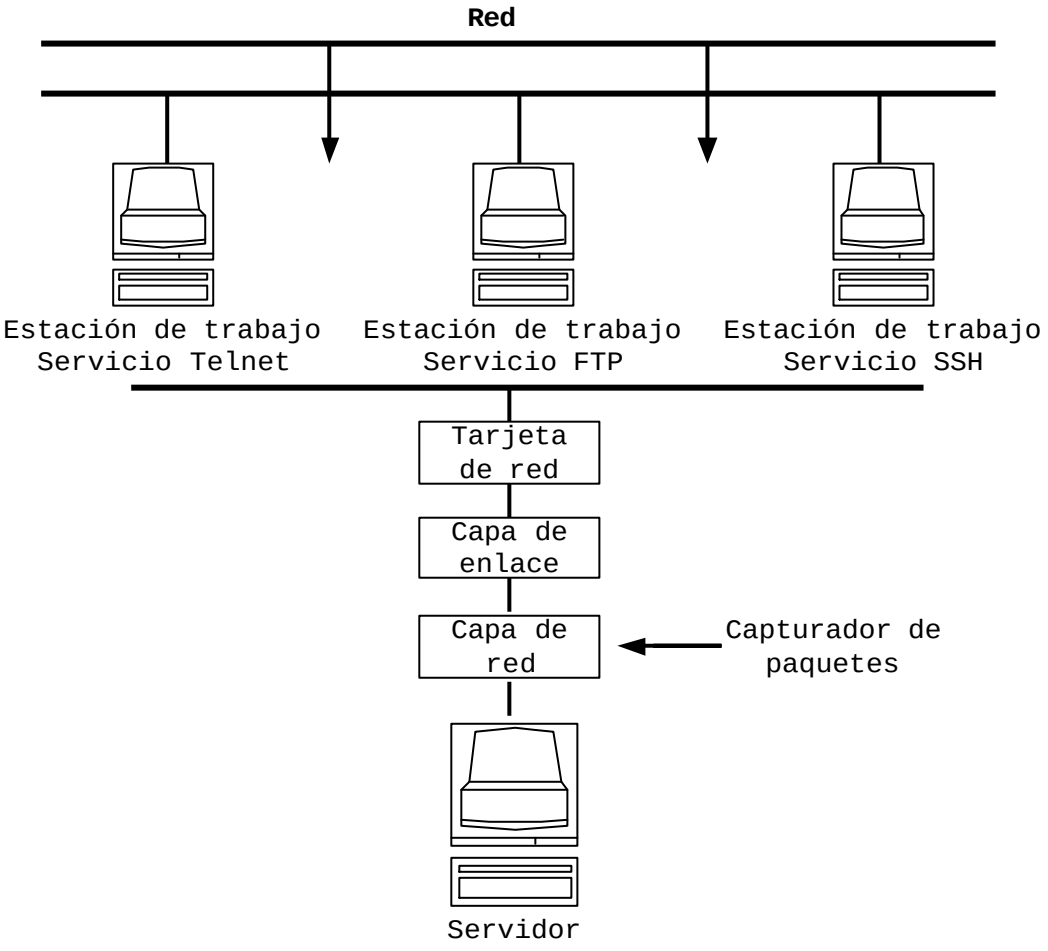


Figura 6.7 Diagrama funcional

El programa esta constituido por un sólo programa, el cual contiene funciones, estructuras, variables, etc. A continuación se explica la funcionalidad de cada una de las partes que constituyen a dicho programa.

Como primer punto, se crean las variables que vamos ha necesitar durante la programación, estas son las mas importantes.

```
#define BUFSIZE 1024
unsigned char packet[BUFSIZE];
struct sockaddr_in bindPort, sin;
int num3,num_pac,min,seg,hor;
char cad[9000],comand[9000];
```

La variable packet, es en donde se van a guardar los datos recibidos, num_pac es donde se almacenará el número de paquetes en un determinado tiempo, la variable cad es donde se almacenará la información recibida de los paquetes.

```
struct ippkt
{
int o_num1,o_num2,o_num3,o_num4;
int d_num1,d_num2,d_num3,d_num4;
int protocolo,datos,puerto,secuencia,tipo;
int tmp,tmp1,tmp2,tmp3,win_uni;
char *fuente;
}paq;
```

Posteriormente se crea una estructura con las variables que se utilizaran para almacenar la información de los paquetes recibidos. Las variables o_num1, o_num2, o_num3 y o_num4 son variables que se utilizaran para formar la dirección IP, esto con la finalidad de conocer el origen de los datos. En la variable protocolo se almacena la versión del protocolo utilizado, en la variable puerto y secuencia se almacenarán tanto el número de puerto de la conexión como el número de secuencia respectivamente. Por último en la variable datos, se almacenará la información que envía el cliente.

Se verifica el número de argumentos que se tiene en la línea de comandos, el único parámetro escrito es la dirección IP donde se va a ejecutar el programa, con la finalidad de no monitorear los paquetes que se transmiten a través de esta terminal. La dirección IP se tendrá que poner con puntos. En caso de poner más de dos argumentos marcará un ERROR con un mensaje y saldrá del programa.

```
   if(argc != 2)
      {
      printf("\n\n\t Debes de Poner la IP del cliente donde se
esta ejecutando \n\n");
      exit(0);
      }
```

La figura 6.8 indica el número de argumentos transferidos hacia el programa, esto con la finalidad de verificar la dirección IP de donde se está ejecutando dicho programa.

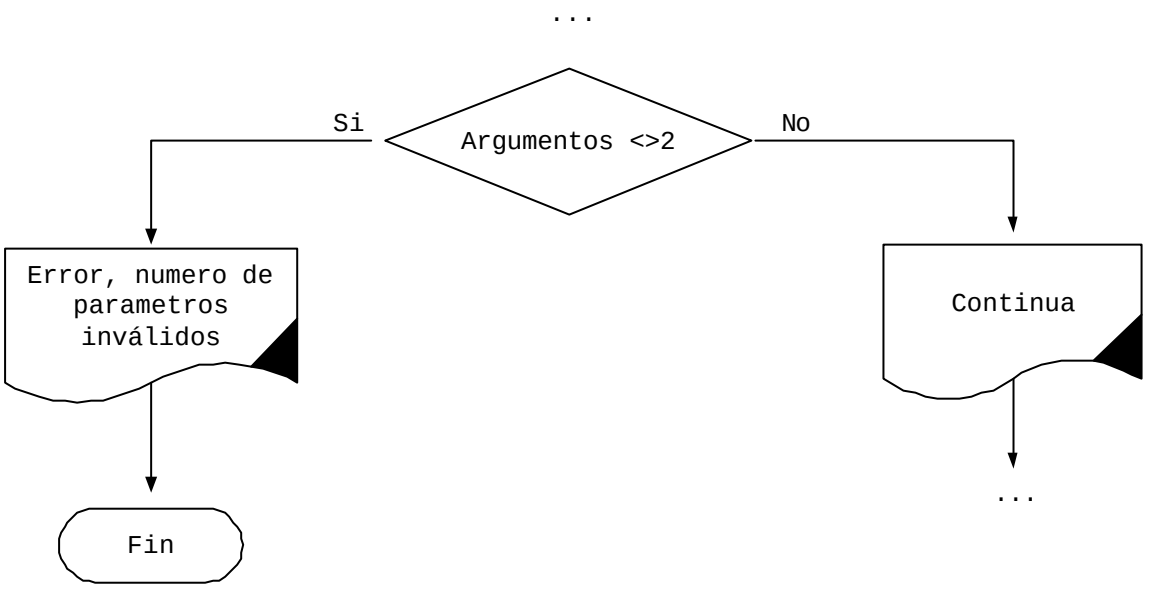


Figura 6.8 Verificación de número de parámetros

El programa contempla la creación de un archivo para poder tener una bitácora para su posterior análisis. Este archivo se llamará tcp.log, las características de este archivo serán las siguientes:

- En caso de que no exista dicho archivo, la función creará uno nuevo y será de escritura.
- En caso de que el archivo exista, se añadirá la información que se le envié, es decir, mantiene la información que tenía anteriormente.
- En caso de que tuviera problemas para crear ese archivo marcará un ERROR y saldrá del programa.

```
if((net=fopen("tcp.log","a+"))==NULL)
   {
   printf("no se puede leer el archivo");
   exit(0);
   }
```

En la Figura 6.9 se tiene la verificación de la apertura del archivo de salida en donde se almacenará la información capturada.

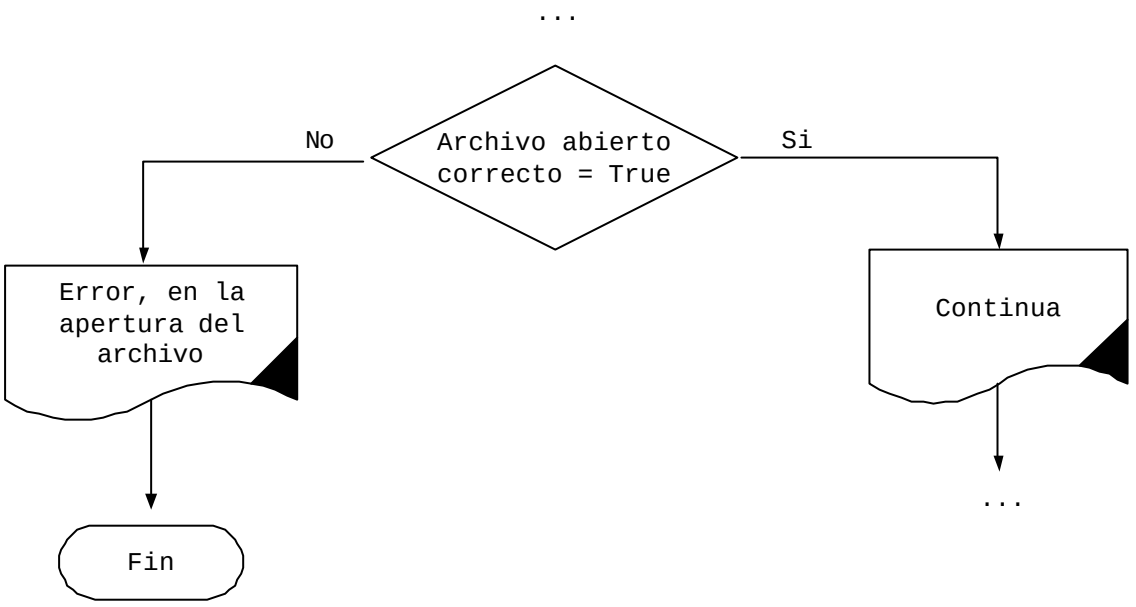


Figura 6.9 Apertura del archivo de salida

El programa crea un socket de tipo TCP, el cual estará en escucha para cualquier tipo de petición orientada a conexión que se le solicite. Si la creación del socket no fuera exitosa regresara un valor de -1, marcara un ERROR y saldrá de la ejecución. Para esta consideración se emplea el tipo de socket SOCK_RAW, ya que es utilizado para interfaces directas con el protocolo IP en la capa de red.

```
   s=socket(AF_INET, SOCK_RAW, IPPROTO_TCP);
   if (s==-1)
      {
      printf("ERROR\n");
      exit(0);
      }
```

En la Figura 6.10, se muestra el diagrama de flujo para la apertura del socket para capturar los paquetes TCP.

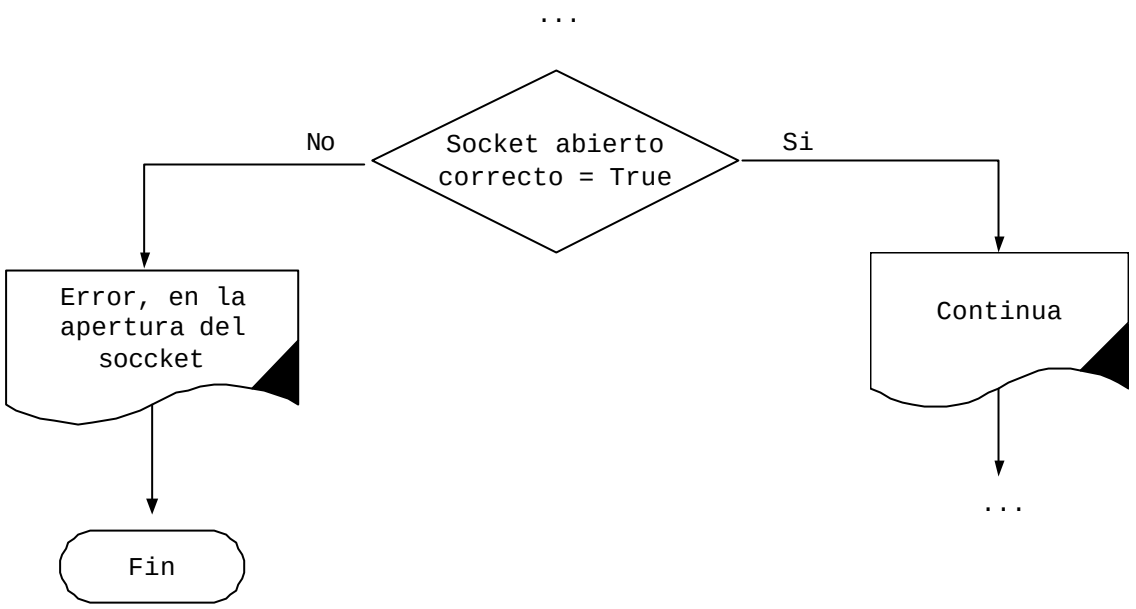


Figura 6.10 Apertura del Socket

Asignamos valores a la variable bindPort, esto para definir los valores predeterminados que utilizará. Se utiliza AF_INET, ya que estaremos trabajando con los conjuntos de protocolos TCP/IP.

```
   bindPort.sin_family=AF_INET;
   bindPort.sin_addr.s_addr=0;
```

Ya creado el socket con los parámetros dados, es necesario asignarle una dirección de forma siguiente.

```
ret=bind(s, &bindPort, sizeof(struct sockaddr_in));
   if (ret != 0)
      {
      printf("ERROR \n" );
      exit(0);
      }
```

Esta función regresa un valor de 0 si tuvo éxito ó un valor de –1 en caso de que exista un ERROR, por lo que indicará un ERROR y saldrá de la ejecución.

Cabe mencionar que antes de almacenar los datos en variables o en una cadena, es necesario limpiar las cadenas, en caso contrario la información será errónea. Las variables de almacenamiento son las siguientes:

```
cad[0]='\0';
comand[0]='\0';
```

Es necesario considerar un intervalo de tiempo para monitorear la cantidad de paquetes que se transfieren en dicho intervalo. Para esto es necesario llamara a la función time() y pasarle como argumento la dirección de una variable tipo entera donde se almacenará el tiempo, esto se da en millonésimas de segundos. Posteriormente se vuelve a hacer la llamada a la función time(), y se le pasa otra variable distinta para posteriormente restar ese tiempo y tomar ese resultado como un entero y saber de esa forma el número de paquetes en el tiempo transcurrido. El monitoreo se hará cada 10 minutos.

```
(void) time(&t1);

      (void) time(&t2);
      seg = (int)t2-t1;
      if(seg>=600){
         printf("\n\t\t   Monitoreo en %i Minutos\n",seg/60);
         fprintf(net,"\n\t\t            Monitoreo    en    %i
Minutos\n",seg/60);
         printf("\n\t\t%i      Paquetes            en      %i
Minutos\n",num_pac,seg/60);
         fprintf(net,"\n\t\t%i     Paquetes          en     %i
Minutos\n",num_pac,seg/60);
         num_pac = 0;
         (void) time(&t1);
         seg = 0;         }
         }
```

En la Figura 6.11, se muestra el ciclo para determinar el tiempo que a su vez permitirá determinar la cantidad de paquetes enviados por parte de los clientes.

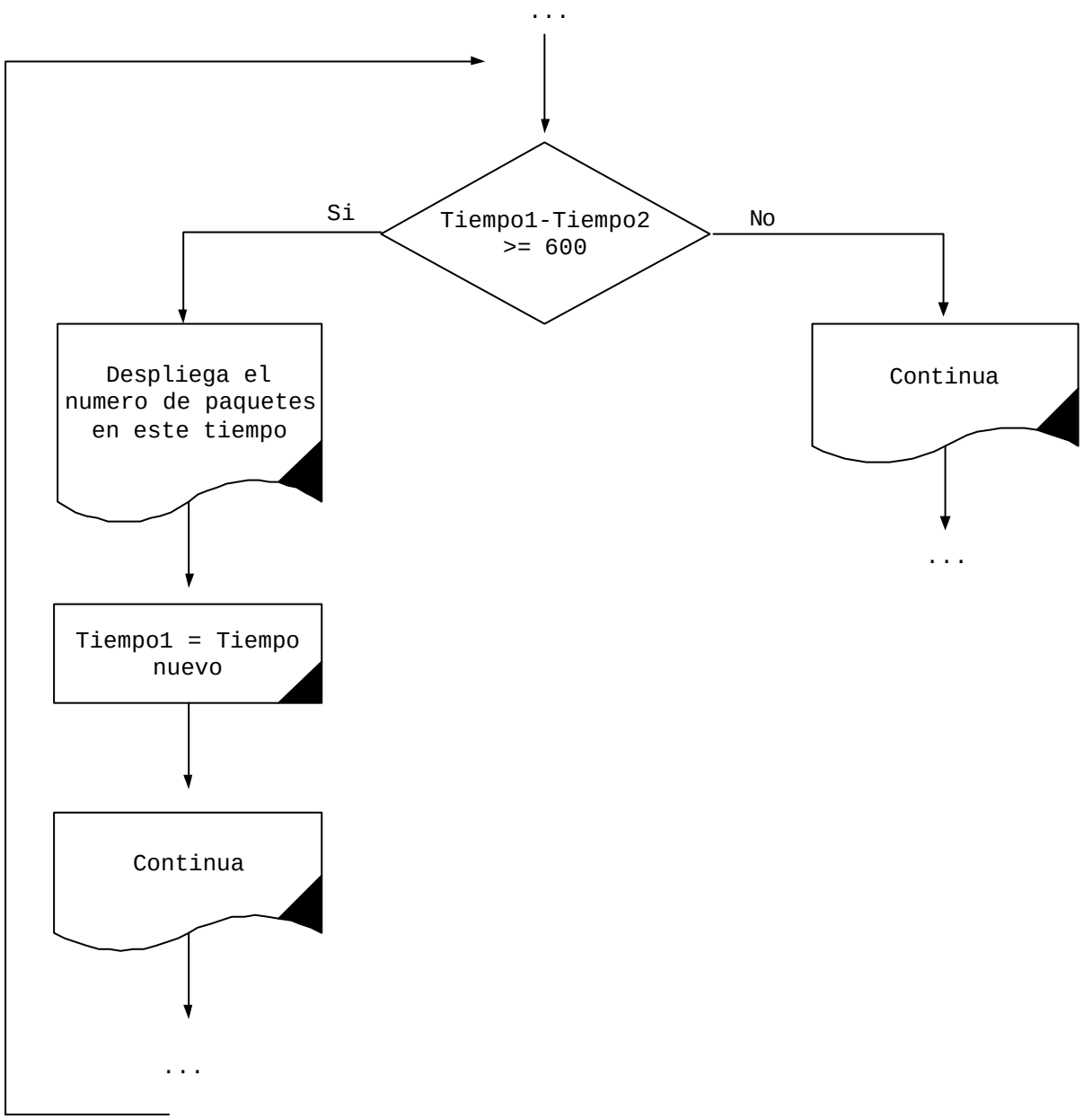


Figura 6.11 Ciclo para determinar el número de paquetes en un determinado tiempo.

Una vez creadas las variables a ocupar y configurando algunas de ellas, es necesario crear un bucle infinito, esto para recibir siempre los paquetes que lleguen a nuestro servidor para posteriormente analizarlo.

Reciben los paquetes que llegan a través de la función recvfrom.

```
recvfrom(s, packet, BUFSIZE, 0, &sin, &sinlen);
```

Se puede observar, que dentro de esta función colocamos como primer parámetro el descriptor del socket que se ha recibido anteriormente, packet es la variable para almacenar la información.

Se forza a la variable packet que sea de tipo iphdr para asignarla a hdr, esto sólo tiene la finalidad de especificar dos maneras distintas de acceder a la información. Posteriormente se tiene que pasar la información a una variable del mismo tipo, en caso contrario marcará un ERROR. La variable fuente es un apuntador a char que posteriormente se utilizará en la comparación con el argumento escrito en la línea de comandos.

A continuación se muestran las posiciones más importantes y utilizadas en la programación. Los paquetes Windows y Solaris son muy semejantes excepto en unas posiciones, que es donde se puede distinguir entre estas.

- En la posición número 24 de este paquete se encuentra el puerto a donde se dirige la información, esto nos servirá para saber que servicio se está utilizando.
- En la posición 2, se encuentra el tipo de máquina que se está conectando, es decir, Windows, Linux o Solaris.
- En la posición 35, se tiene la diferencia entre Sistemas Operativos Windows y Solaris.

Ya que se tiene la dirección IP, es necesario hacer la comparación entre el origen del paquete y la máquina donde se está ejecutando el programa, la función strcmp(), arroja un 0 si las cadenas comparadas son iguales y otro valor en caso contrario. En caso que la función strcmp() devuelva un valor diferente de cero se analiza ese paquete.

```
hdr=(struct iphdr*)packet;
addr.s_addr=hdr->saddr;
fuente = inet_ntoa(sin.sin_addr);
```

```
paq.puerto = (int)*(packet + 23);
paq.tipo = (int)*(packet + 1);
paq.win_uni = (int)*(packet + 34);
re=strcmp(fuente,servidor);
```

La Figura 6.12, muestra el análisis de la cabecera del paquete para determinar posteriormente el servicio que esta solicitando y así poder desplegar la información y enviarla a la bitácora.

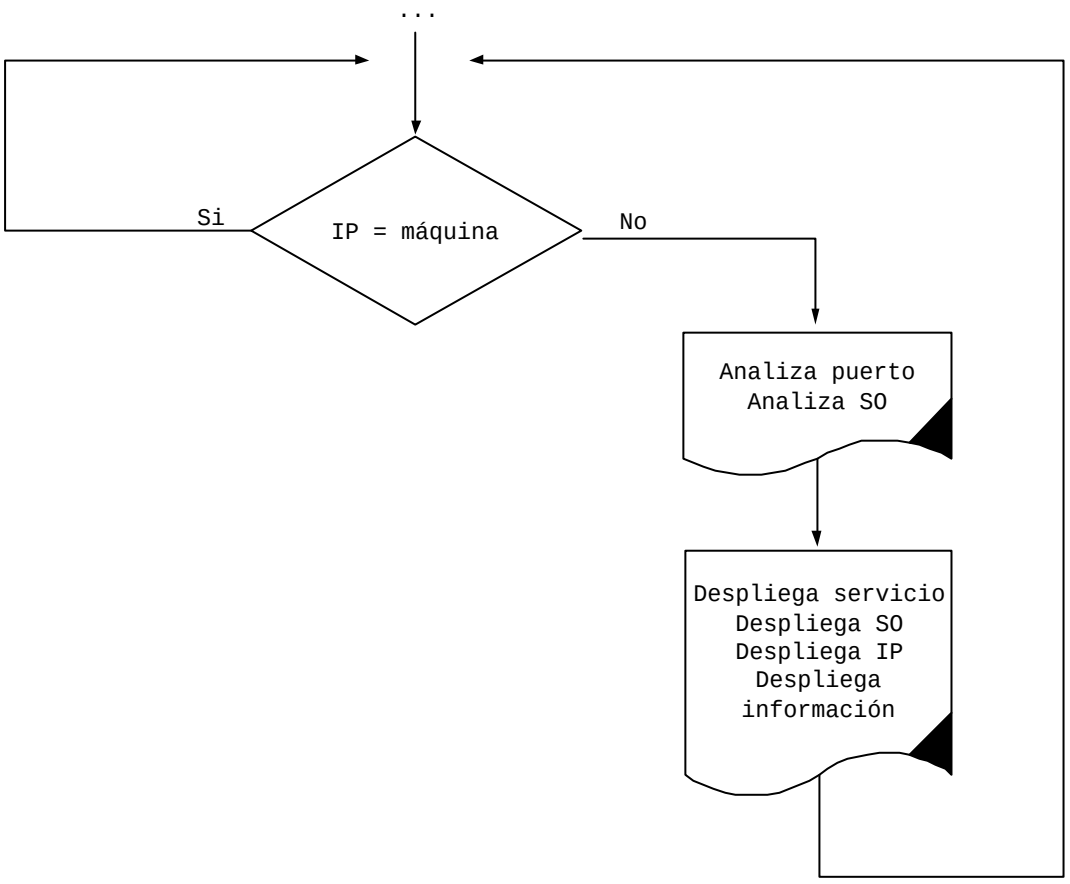


Figura 6.12 Diagrama para analizar del paquete IP

Ya que se tiene identificado cada una de las partes que forman el paquete, ahora es necesario plantear cada uno de los servicios de forma gráfica para poder entender las posiciones de la información. A continuación se presentan cada uno de los servicios mas comunes existentes.

6.4.1 Paquete Telnet

La figura 6.13, muestra el paquete que envía el cliente desde una máquina con Sistema Operativo Solaris. El contenido de la dirección IP Origen, IP destino, el protocolo y los datos.

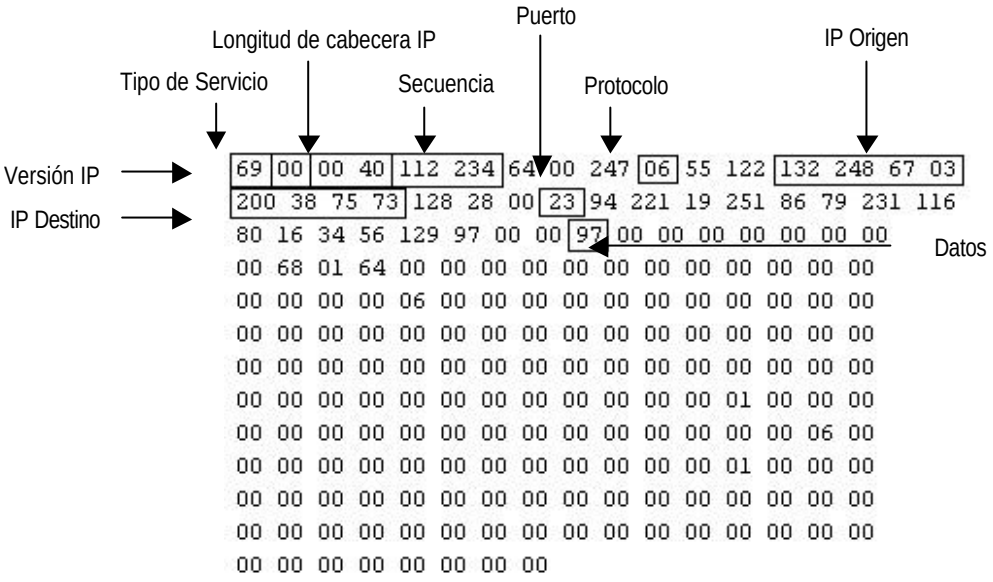


Figura 6.13 Paquete Telnet con Sistema Operativo Solaris

Ahora se hace la comparación de la variable puerto con diferentes servicios, como el TELNET para el caso de que el puerto este solicitando la petición por el puerto 23.

```
if (paq.puerto == 23)
```

En caso afirmativo se tiene que realizar la distinción entre el tipo de máquina cliente (Windows, Solaris o Linux). Para el tipo de máquina Windows y Solaris el paquete tiene como contenido un “00”

```
if (paq.tipo == 00)
```

En caso de ser afirmativa esta comparación se procederá a la distinción de los equipos Windows y Solaris. Para esta distinción es necesario compararlo con los valores “96” ó “34”. En caso afirmativo entonces se está hablando de un equipo Solaris, por lo tanto se analizará la información que se envía.

```
if (paq.win_uni == 96 || paq.win_uni == 34)
```

- En la posición 41, se tiene el dato transmitido por parte del cliente y se almacena en una variable del mismo tipo, en este caso dentro de la variable paq.dato.

```
paq.datos = (int)*(packet + 40);
paq.tipo = (int)*(packet + 3);
if (paq.datos == 13 && paq.tipo == 42)
```

Ahora se tiene la traducción de ese dato, es decir la sustitución de ese número decimal en un código legible. Primero se pasa el número a una variable del mismo tipo (int), para posteriormente ser codificado y almacenado en una cadena para su futura impresión. Con las opciones printf() y fprintf() podemos mandar resultados a la pantalla y a un archivo respectivamente, esto para llevar una bitácora instantánea (pantalla) y la otra para llevar una bitácora (archivo). Aquí se manda a escribir el tipo de sistema operativo (Solaris), el número del puerto que esta dando el servicio (23) y la dirección IP origen de los datos.
Existen dos formas de traducir la información:

- ✓ La primera, es la función toascci(), en la cual sólo pasas como parámetro el entero de 0 a 255 (decimal) y devuelve su correspondiente carácter.
- ✓ La segunda es, realizar una función de conversión, donde se le pasa como parámetro el número decimal, lo convierte y lo almacena en una cadena temporal.

La elección entre estas dos formas de conversión fue personal, por lo que la primera opción fue tomada.

```
datos = paq.datos;
printf("Sol.              Tel.        (%04d)      %15s   ..
",paq.puerto,paq.fuente);
fprintf(net,"Sol.            Tel.       (%04d)     %15s   ..
",paq.puerto,paq.fuente);
traduce(datos);
```

Si la comparación no es exitosa entonces se está hablando de un equipo Windows, por lo tanto se hará la codificación para los equipos Windows

La figura 6.14, muestra un paquete son sistema operativo Windows.

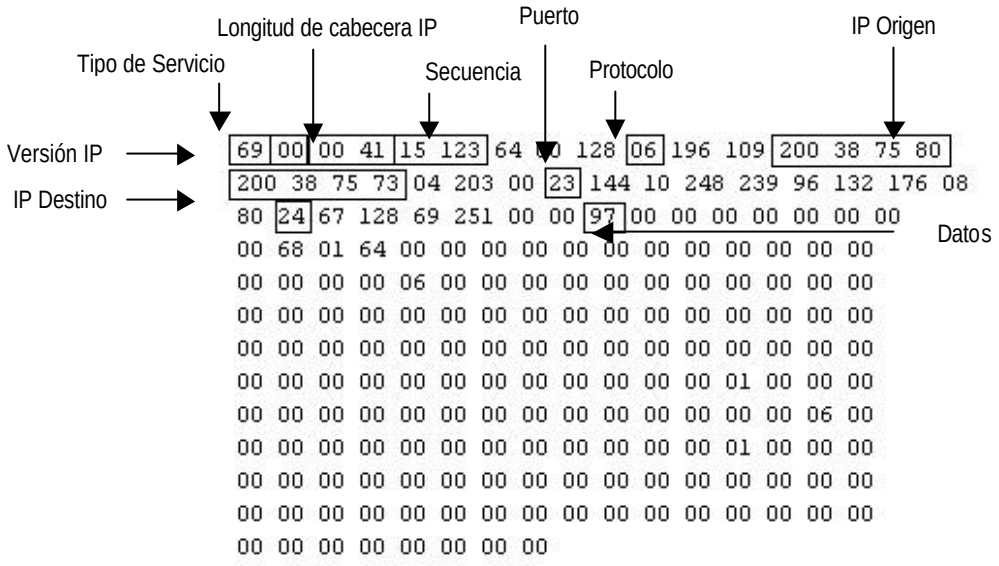


Figura 6.14 Paquete Telnet con Sistema Operativo Windows

```
if (paq.win_uni == 96 || paq.win_uni == 34)
```

Para el proceso de codificación de los datos de máquinas Windows se hace el mismo proceso que el sistema operativo Solaris.

```
printf("Win.              Tel.        (%04d)      %15s   ..
",paq.puerto,paq.fuente);
fprintf(net,"Win.            Tel.       (%04d)     %15s   ..
",paq.puerto,paq.fuente);
traduce(datos);
```

Ahora se tiene el sistema operativo Linux, el cuál tiene características similares a las otras dos pero con diferencia de posiciones. En caso de que el servicio fuera por el puerto 23, pero el valor de la posición 2 fuera “16”, se trata de un equipo Linux.

La figura 6.15, muestra un paquete con sistema operativo Linux.

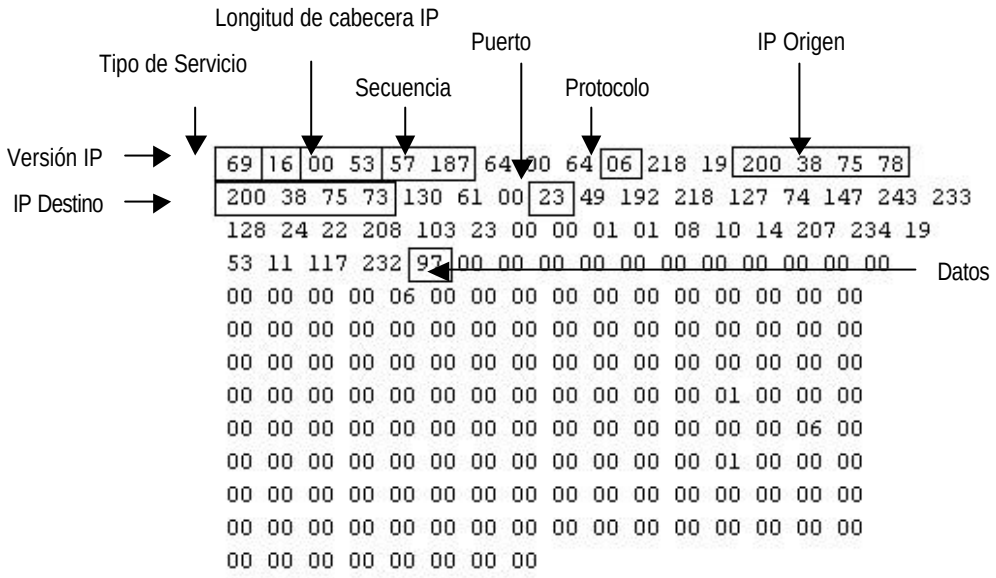


Figura 6.15 Paquete Telnet con Sistema Operativo Linux

```
If (paq.tipo == 16)
```

La bandera para poder imprimir la información que se envía por parte del cliente es el carácter decimal “13”, esto debido a que el usuario al terminar un comando, debe teclear dicho carácter.

6.4.2 Paquete FTP

Ahora se tiene el servicio FTP, el cual trabaja distinto al servicio Telnet como se ha visto en los anteriores capítulos.

Para distinguir este servicio es necesario comparar la variable que trae el puerto (paq.puerto) con el decimal “21”, ya que el número de puerto que ofrece este servicio es el 21.

En caso afirmativo, se tiene que estudiar este tipo de paquete como hasta este momento lo hemos visto, para identificar la información de cada paquete.

Como se puede observar, los datos empiezan a partir de la posición 41, a diferencia del Telnet, el FTP manda el aquete cuando termina su comando, es decir, después de teclear un comando será necesario darle una instrucción que determine la petición de ese comando, esa instrucción es el decimal “13”. En ese momento se envía ese paquete y el servidor traduce ese comando.

Es necesario desplegar en pantalla y en la bitácora el resultado de esta petición, en este caso solo enviamos el servicio (FTP), puerto (21) y la dirección IP Origen.

La figura 6.16, muestra el paquete del servicio FTP, el contenido del paquete es el comando que se desea ralizar.

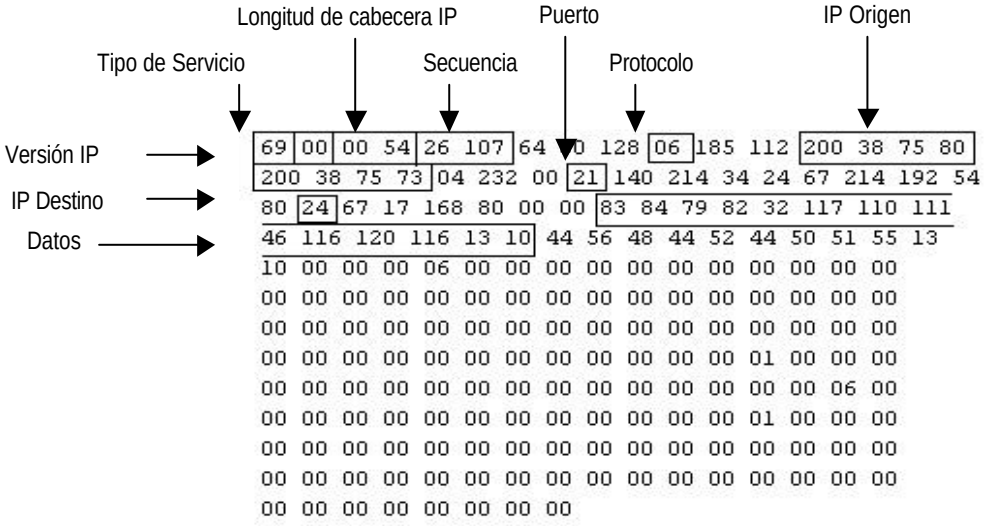


Figura 6.16 Paquete FTP, información de comandos

La figura 6.17, muestra el paquete del servicio FTP, el contenido del paquete es la información del archivo.

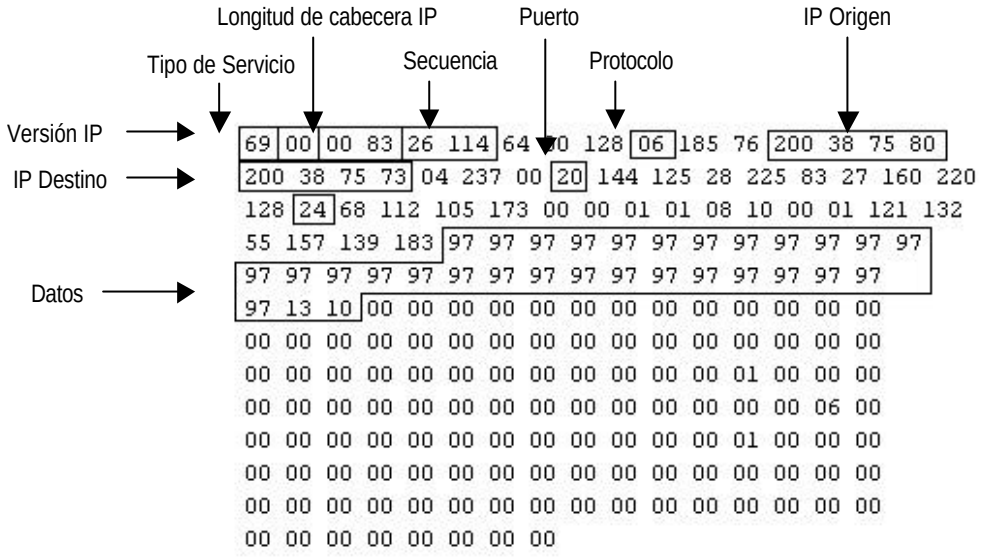


Figura 6.17 Paquete FTP, información del archivo

Para poder decodificar la información enviada en el paquete se utilizará la misma función traduce(), la cual desplegará la información almacenada en las variables.

```
if (paq.puerto == 21 )
{
paq.tipo = (int)*(packet + 33);
if (paq.tipo == 24)
{
printf("----               Ftp.         (%04d)      %15s   ..
",paq.puerto,paq.fuente);
fprintf(net,"----             Ftp.        (%04d)     %15s   ..
",paq.puerto,paq.fuente);
for(i=40;i<=100;i++)
   {
   paq.datos = (int)*(packet + i);   //Datos
   datos = paq.datos;
   traduce(datos);
   if (datos == 13)
      break;
    }
}
```

6.4.3 Paquete Secure Shell

Ahora se tiene el paquete de Secure Shell, en esta parte la información llega de forma encriptada, por lo tanto no se codificará la información que tiene ese paquete.

La figura 6.18, muestra el servicio de SSH.

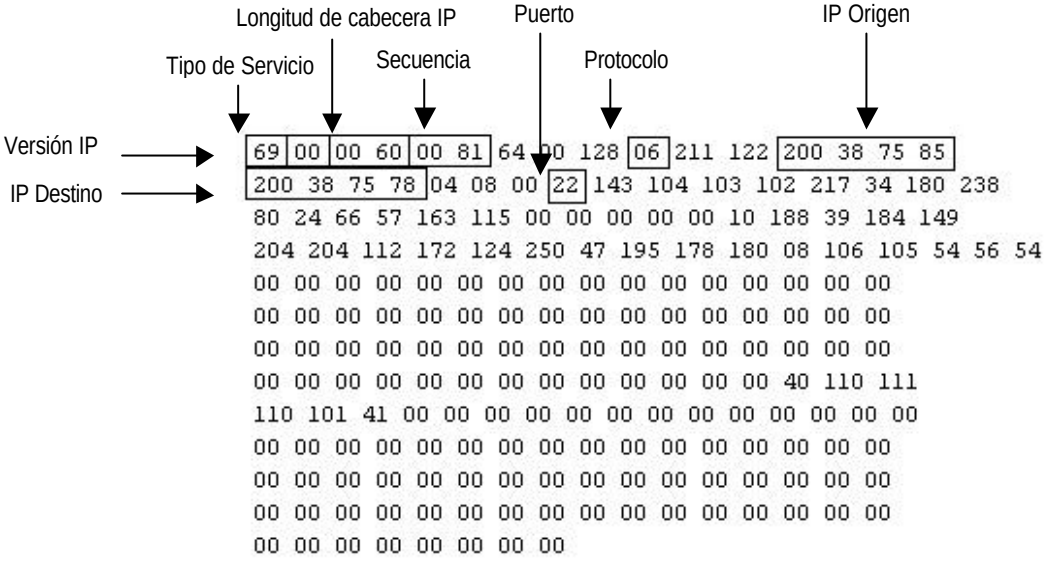


Figura 6.18 Paquete SSH

Para el servicio de Secure Shell, es necesario que la variable puerto tenga un contenido “22”, ya que este es el número de puerto que ofrece este servicio. En esta parte no podemos codificar la información, ya que este tipo de servicio tiene por convicción la encriptación de la información, esto se hace por seguridad de los equipos. Lo único que se puede recuperar de este paquete es la dirección IP fuente y destino, el protocolo de comunicación, la secuencia de paquetes y el puerto de comunicación. Para este monitoreo solo se tiene que presentar la conexión presente dentro del servidor así como en la bitácora.

```
if (paq.puerto == 22 )
        {
        printf("----          SSH   .    (%04d)    %15s  ..
",paq.puerto,paq.fuente);
        fprintf(net,"----        SSH  .   (%04d)   %15s  ..
",paq.puerto,paq.fuente);
        }
```

6.4.4 Paquete Finger

La figura 6.19, muestra el paquete del servicio finger.

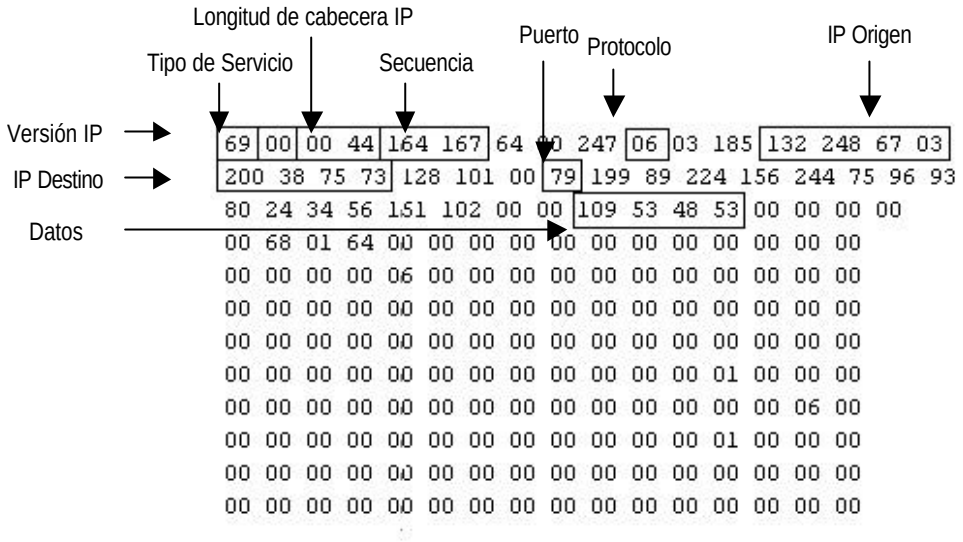


Figura 6.19 Paquete Finger con información

```
if ((paq.tipo == 24 && paq.tmp == 44) || (paq.tipo == 16 &&
paq.tmp == 44))
   {
   for(i=40;i<=100;i++)
      {
      paq.datos = (int)*(packet + i);
      datos = paq.datos;
      traduce(datos);
      paq.tmp1 = (int)*(packet + i + 1 );
      paq.tmp2 = (int)*(packet + i + 2 );
      if (paq.tmp1 == 0 && paq.tmp2 == 0)
          {
          datos =  13;
          printf("----          Fing.    (%04d)    %15s  ..
",paq.puerto,paq.fuente);
          fprintf(net,"----        Fing.   (%04d)   %15s  ..
",paq.puerto,paq.fuente);
          traduce(datos);
          break;
          }
       }
   }
```

6.4.5 Paquete Cliente Aleph

La figura 6.20, muestra el paquete del servicio de Aleph.

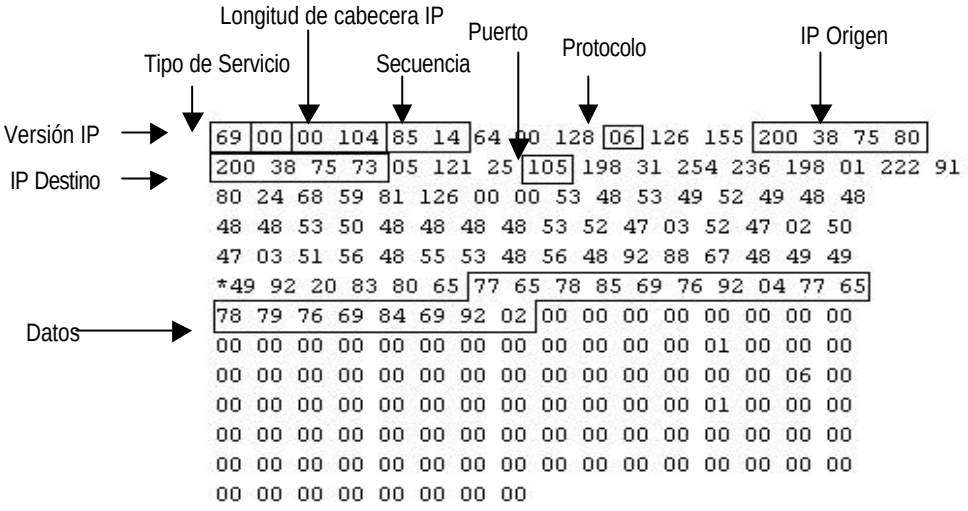


Figura 6.20 Paquete Aleph

Para el servicio de cliente GUI que utiliza la Dirección General de Bibliotecas, tenemos un paquete muy parecido al Finger a comparación de la posición donde se encuentra la información. El puerto utilizado en este paquete es el 6505. Para este tipo de paquete la información se empieza a codificar a partir de la posición 81, justamente aquí empiezan los datos. Al igual que los demás servicios vamos a utilizar la función toascii() para poder codificar la información. Con las funciones que se conocen (printf y fprintf) se envía la información a pantalla y a la bitácora para posteriormente analizarlo. En esta información se manda a imprimir el servicio Aleph, el número de puerto (6505), la información del paquete y la dirección IP Origen.

```
if (paq.puerto == 6505)
   {
   paq.tipo = (int)*(packet + 33);
   paq.tmp = (int)*(packet + 1);
   if (paq.tipo == 24 && paq.tmp == 0)
      {
      for(i=80;i<=150;i++)
         {
         paq.datos = (int)*(packet + i);
         datos = paq.datos;
         paq.tmp1 = (int)*(packet + i + 1 );
```

```
          paq.tmp2 = (int)*(packet + i + 2 );
          }
        }
                  printf("Win.       Aleph   (%04d)   %15s ..
",paq.puerto,paq.fuente);
                  fprintf(net,"Win.     Aleph  (%04d)  %15s ..
",paq.puerto,paq.fuente);
```

Para terminar la ejecución del programa, es neesario que el servidor envíe una señal de finalización del programa, ese tipo de señal será un carácter no muy común dentro del lenguaje, ese carácter es el decimal “255”. La secuencia de dicha terminación se muestra a continuación.

En caso de que el paquete que se recibe fuera de la misma dirección IP en donde se esta ejecutando el programa, se tiene que analizar ese paquete para la terminación del monitoreo, si la información enviada por parte del administrador hacia el servidor fuera el decimal “255”, en ese momento se termina el monitoreo y se cierra el archivo tcp.log para su análisis.

Si el programa de monitoreo se termina de cualquier otra forma, la bitácora se puede destruir, esto debido a que el archivo de bitácora no fue cerrado de forma correcta, lo que provoca que la información no se haya guardado de forma correcta.

El break rompe el while infinito que anteriormente se había definido, con esto se alcanza la función fclose() así como el cierre correcto del archivo de bitácora. La última sentencia (system()) regresa al sistema operativo para realizar el comando que pasa como parámetro, esto es con la finalidad de mostrar de forma inmediata la bitácora resultado del monitoreo.

```
datos = (int)*(packet + 40);
if(datos == 122)
   {
    printf("\nTerminado el Monitoreo\n");
    ban = 0;
    break;
    }
 if (ban = 0)
    break;
} // Cierre del While

fclose(net);
system(“more tcp.log”);
```

La fase de pruebas y resultados se llevó a cabo en tres etapas , con la finalidad de entender los conceptos manipulados a lo largo del desarrollo. Esto se hace más fácil si se realizan pruebas detalladas y se explica el procedimiento.

En el anexo 1, se tienen los listado de los programas realizados durante la investigación de este proyecto.

Programa 1, monitorea_todo_TCP.c: Este programa captura todos los paquetes TCP que entran al servidor y no ningún formato, solo muestra el contenido del paquete y lo envía a pantalla.
Programa 2, monitoreo_red.c: Este programa toma como base monitorea_todo_TCP.c y analiza los paquetes capturados retomando datos importantes como: IP origen, IP destino, puerto solicitado, información dentro del paquete, etc. Esto con la finalidad de llevar una bitácora histórica.

7.1 Lenguaje de Programación

A continuación se describen algunas características importantes de programación de las pruebas realizadas con el lenguaje Java así como los conceptos importantes de comunicación de redes.

7.1.1 Creación de Sockets con JAVA

Para poder crear un Socket TCP/IP, se realiza directamente con el paquete java.net, la forma de realizar la conexión entre los sockets es la siguiente:

El servidor establece un puerto y a la vez espera durante un determinado tiempo (timeout) una petición o solicitud de parte de los clientes. Para poder establecer una comunicación, en el momento que exista una petición de parte de los clientes para el servidor, este último abrirá una conexión socket con la función accept(), posteriormente el cliente establece comunicación con el servidor a través del puerto que se haya designado y por último empieza la transferencia de información a través de manejadores InputStream y OutputStream.

Ahora que se tiene el esquema general de la comunicación entre los Sockets, se explica la creación de los Sockets tanto para el cliente como para el servidor.
En la programación para el cliente, se tiene la siguiente función:

```
Socket Cliente;
Cliente = new Socket( “host”,”puerto” )
```

Se observa que el host es el nombre de la máquina a la que se desea establecer comunicación o abrir la conexión y puerto es el número del puerto del servidor que esta corriendo sobre el cual se quiere conectar.
Cabe mencionar que el rango de puertos libres para poder utilizar comunicaciones debe ser diferente de 0 – 1023, ya que estos están reservados para el uso del sistema.

Ahora para poder interceptar a las excepciones sería de la siguiente manera:

```
Socket Servicio;
try
     {
     Servicio = new ServerSocket( numeroPuerto );
     } catch( IOException e )
     {
     System.out.println( e );
     }
```

En la programación para el servidor, se tiene que crear también un objeto para socket de tipo ServerSocket, esto es para que este alerta a las peticiones de los clientes y así poder aceptar las conexiones.

```
Socket socketServicio = null;
try
     {
                socketServicio = miServicio.accept();
     }
catch( IOException e )
     {
                System.out.println( e );
          }
```

Dentro de la programación para los clientes, se puede utilizar la clase DataInputStream, para poder recibir las respuestas que le haga el servidor, la creación de estos elementos es la siguiente:

```
DataInputStream entrada;
try
     {
          entrada = new DataInputStream(
Cliente.getInputStream() );
     }
catch( IOException e )
     {
System.out.println( e );
     }
```

La función DataInputStream, es una clase que permite la lectura de línea de texto y tipos de datos primitivos, la cual dispone algunos métodos para poder leer todos ellos como lo son: read(), readChar(),readInt(),readLine().

Para el lado del servidor también se usará la clase DataInputStream() de forma siguiente:

```
 DataInputStream entrada;
 Try
     {
                 entrada = new DataInputStream(
socketServicio.getInputStream() );
     }
 catch( IOException e )
     {
          System.out.println( e );
     }
```

Ahora para poder crear las cadenas de salida para el cliente se tiene el siguiente código:

```
PrintStream salida;
try
     {
          salida = new PrintStream(
miCliente.getOutputStream() );
     }
catch( IOException e )
     {
        System.out.println( e );
          }
```

La clase PrintStream tiene métodos para la representación textual de todos los datos primitivos de Java. Sus métodos write y println() tienen una especial importancia en este aspecto. No obstante, para el envío de información al servidor también podemos utilizar DataOutputStream:

```
DataOutputStream salida;
try
     {
          salida = new
DataOutputStream(miCliente.getOutputStream());
     }
catch( IOException e )
     {
          System.out.println( e );
          }
```

La clase DataOutputStream permite escribir cualquiera de los tipos primitivos de Java, muchos de sus métodos escriben un tipo de dato primitivo en el stream de salida. De todos esos métodos, el más útil quizás sea writeBytes().

En el lado del servidor, podemos utilizar la clase PrintStream para enviar información al cliente:

```
PrintStream salida;
Try
     {
          salida = new
PrintStream(socketServicio.getOutputStream() );
     }
catch( IOException e ) {System.out.println( e ); }
```

Ya que se ha establecido la comunicación entre el cliente y el servidor, intercambiado información y terminado la comunicación, ahora será necesario cerrar los puerto de comunicación así como cerrar la conexión tanto del cliente como para el servidor.
Ahora se tiene la forma adecuada para poder cerrar los sockets tanto para el cliente como para el servidor.

Las siguientes instrucciones son para el código del cliente:

```
try
     {
          salida.close();
          entrada.close();
          Cliente.close();
     }
catch( IOException e )
     {
          System.out.println( e );
          }
```

Por parte del servidor se tiene el siguiente código.

```
try
{
        salida.close();
        entrada.close();
        socketServicio.close();
        miServicio.close();
}
catch( IOException e )
{
         System.out.println( e );
}
```

Posteriormente se realizaron las pruebas en Lenguaje C para así poder determinar el lenguaje adecuado para este tipo de necesidades. Tomando en consideración factores importantes para el desarrollo, como la experiencia en programación en ambos lenguajes, la flexibilidad del lenguaje, portabilidad, etc. lo primero que se realizó fue investigar la forma de capturar los paquetes que pasan a través de la capa de red dentro de un servidor, esto para su posterior análisis.

El primer programa fue hecho en C, el cuál nos sirve para poder programar y trabajar en la Capa de Red. El objetivo es analizar la estructura de la información mediante el protocolo TCP en una arquitectura cliente-servidor.
La ejecución del programa se realiza colocando el nombre del programa por ejecutar seguido de la dirección IP en donde se está ejecutando, esto desplegará el resultado a pantalla para poder analizar el resultado de todas las conexiones que se realizan dentro del servidor.

La figura 7.1 muestra un paquete capturado.

```
shitzu emmanuel 34>./monitorea_todo 200.38.75.80
Fuente    : 200.38.75.73
Destino   : 200.38.75.78
Protocol  : 6
ID              : 59573
TELNET
69 16 00 53 181 232 64 00 64 06 93 230 200 38 75 78
200 38 75 73 128 126 00 23 234 202 15 151 06 151 74 159
128 24 22 208 183 43 00 00 01 01 08 10 05 138 143 31
70 141 121 111 97 00 00 00 114 03 00 00 252 03 00 00
169 05 00 00 00 00 00 00 24 06 00 00 185 06 00 00
214 01 00 00 206 02 00 00 18 01 00 00 124 04 00 00
227 06 00 00 72 02 00 00 255 02 00 00 51 05 00 00
171 06 00 00 74 06 00 00 33 01 00 00 00 00 00 00
97 05 00 00 131 01 00 00 100 01 00 00 250 02 00 00
95 06 00 00 00 00 00 00 00 00 00 00 229 05 00 00
99 05 00 00 00 00 00 00 238 00 00 00 00 00 00 00
170 06 00 00 196 06 00 00 199 05 00 00 21 04 00 00
77 05 00 00 123 05 00 00
Fuente    : 200.38.75.73
Destino   : 200.38.75.78
Protocol  : 6
ID              : 59829
TELNET
69 16 00 52 181 233 64 00 64 06 93 230 200 38 75 78
200 38 75 73 128 126 00 23 234 202 15 152 06 151 74 160
128 16 22 208 207 178 00 00 01 01 08 10 05 138 143 31
70 141 193 239 97 00 00 00 114 03 00 00 252 03 00 00
169 05 00 00 00 00 00 00 24 06 00 00 185 06 00 00
214 01 00 00 206 02 00 00 18 01 00 00 124 04 00 00
227 06 00 00 72 02 00 00 255 02 00 00 51 05 00 00
171 06 00 00 74 06 00 00 33 01 00 00 00 00 00 00
97 05 00 00 131 01 00 00 100 01 00 00 250 02 00 00
```

Figura 7.1 Recepción de paquetes

La figura 7.1 muestra el paquete IP enviado desde un cliente, mostrando la información contenida dentro de éste. Ya que se tiene el paquete completo, se empieza a analizar cada uno de los campos que constituyen al paquete IP.
La importancia del conocimiento de la información del paquete IP, radica principalmente en el número de bytes que forman cada uno de los campos dentro del paquete.

Al establecer conexión entre cliente/servidor, el cliente deberá pedir solicitud de comandos al servidor, en el momento en que el primero teclee una letra, este envía información al servidor, el programa captura ese paquete y lo muestra en pantalla.

7.2 Análisis de paquetes

En la segunda etapa, la tarea fue analizar el paquete recibido, con base en la investigación realizada, se pudo descifrar cada uno de los bytes contenidos en el paquete.

El análisis se realizó a través del programa descrito anteriormente en donde se estudia la posición de los datos, el número de puerto a donde se dirige el paquete y la información que contiene. Después de analizar los paquetes se codifican los datos y se almacenan en una cadena para posteriormente enviarla a una bitácora así como a la salida estándar.

El monitoreo se hizo para la comunicación orientada a conexión para los servicios mas concurridos de parte de los clientes, estos servicios son los siguientes:

FTP
Telnet
Secure Shell
Finger
Cliente Aleph GUI
WEB

7.3 Resultados finales

En la última etapa, se tienen que mostrar los datos de forma ordenada, identificada y codificada de cada uno de los paquetes que llegan al servidor de forma que sean claramente leídas por el administrador de la red.

Una forma de monitorear el tráfico es mostrar en pantalla lo que está haciendo cada uno de los clientes conectados al servidor, esto puede ser por cuestión de seguridad o simplemente saber que es lo que esta haciendo un determinado usuario.

La figura 7.2 indica la forma que se muestran los datos en pantalla:

Figura 7.2 Monitoreo de información

En el primer campo se tiene el tipo de equipo que se está conectando al servidor, en segundo campo tenemos el servicio que está solicitando seguido del puerto, el tercer campo se refiere a la dirección IP Fuente y el último es el comando que está ejecutando.

La bitácora tendrá exactamente la misma información que los datos enviados a pantalla pero solo con un encabezado para facilitar su análisis.

Este archivo se muestra en cuanto termina el programa en ejecución.

La figura 7.3, muestra la bitácora almacenada durante el proceso de monitoreo:

Equipo	Serv.	Puerto	IP Origen	Datos
Win.	Tel.	(0023)	200.38.75.86 ..	ls -la
----	Ftp.	(0021)	200.38.75.86 ..	PORT 200,38,75,86,17,205
----	Ftp.	(0021)	200.38.75.86 ..	LIST
----	Ftp.	(0021)	200.38.75.86 ..	PORT 200,38,75,86,18,49
----	Ftp.	(0021)	200.38.75.86 ..	STOR uno
----	Ftp.	(0021)	200.38.75.86 ..	PORT 200,38,75,86,18,192
----	Ftp.	(0021)	200.38.75.86 ..	NLST -la
Win.	Tel.	(0023)	200.38.75.86 ..	more uno
Lin.	Tel.	(0023)	200.38.75.78 ..	1-al
Lin.	Tel.	(0023)	200.38.75.78 ..	h
Lin.	Tel.	(0023)	200.38.75.78 ..	ls s-l
Lin.	Tel.	(0023)	200.38.75.78 ..	ls -la
		Monitoreo en 1 Minutos		
		200 Paquetes en 1 Minutos		
Lin.	Tel.	(0023)	200.38.75.78 ..	vi test.csh
Lin.	Tel.	(0023)	200.38.75.78 ..	dG:x!
Lin.	Tel.	(0023)	200.38.75.78 ..	
		Monitoreo en 1 Minutos		
		35 Paquetes en 1 Minutos		
Sol.	Tel.	(0023)	132.248.67.3 ..	m505
Sol.	Tel.	(0023)	132.248.67.3 ..	Sistem
		Monitoreo en 1 Minutos		
		56 Paquetes en 1 Minutos		
Sol.	Tel.	(0023)	132.248.67.3 ..	m50
Sol.	Tel.	(0023)	132.248.67.3 ..	
Sol.	Tel.	(0023)	132.248.67.3 ..	ls -la
		Monitoreo en 2 Minutos		
		105 Paquetes en 2 Minutos		
Sol.	Tel.	(0023)	132.248.67.3 ..	e
		Monitoreo en 1 Minutos		

Figura 7.3 Bitácora de monitoreo

Para la realización de todo proyecto, es necesario tener un objetivo claro de lo que se necesita o se desea. Es ineludible tener un plan de trabajo muy bien estructurado con el que vayamos avanzando de forma correcta con el desarrollo del proyecto, así que existirá una retroalimentación en cada una de las etapas, lo que ayudará a lograr el objetivo esperado de una forma eficiente.

Para el desarrollo de este proyecto se investigo a fondo el protocolo TCP/IP así como estudiar los servicios y aplicaciones que utilizan este protocolo. La investigación respecto a capas superiores fue un factor importante también, ya que en base a ésta, se pudo definir en que capa de red se realizó la programación.

La elaboración y culminación de este desarrollo fue muy interesante, ya que ver físicamente los paquetes TCP transferidos a un servidor y poderlos manipular fue lo mas importante así como verificar los servicios y aplicaciones utilizadas dentro de la red. Conocer mas a fondo el protocolo TCP/IP y saber como funciona, fue otra parte elemental dentro del proyecto, esto debido a la importancia de este protocolo.

La idea principal nace a partir del interés de contar con una herramienta propia y de uso fácil para poder mostrar los aspectos de la red que se esta monitoreando, esto con la finalidad de detectar congestión en la red, mostrar los servicios mas utilizados por parte de los clientes y llevar una bitácora para una comparación cuantitativa de los servicios de una red.

Este monitor visualiza el tráfico y la información que pasa a través de un servidor permitiendo detectar algunas inconsistencias o anomalías por parte de los clientes. La información que puede presentar es:

- Cantidad de paquetes que pasan a través del servidor.
- Mal uso del servidor.
- Detección de comandos no permitidos.
- Detectar la cantidad de máquinas conectadas al sistema.
- Tiempo de conexión de máquinas conectadas al servidor.
- Bitácora histórica de los acontecimientos presentados en un determinado día.

Una de las ventajas de este monitor es que es flexible, es decir, permite monitorear los servicios más concurridos por los usuarios en donde si se desea analizar otros servicios, será sencillo adecuarlo a las nuevas necesidades.

Es importante ir generando nuevas adecuaciones a este monitor para que la herramienta sea de gran utilidad para las necesidades posteriores en el ámbito de redes y sobre todo para aportar a las próximas generaciones un poco de este software.

Unix Programación avanzada
Márquez, Fco. Manuel
Alfaomega 2001.

Unix Sistema Versión 4
Rosen, Kenneth H.
McGraw-Hill 1998

Internet interno. Marcombo
Tischer, Michael
Marcombo S.A 1997.

Curso de programación en C/C++
Ceballos Sierra, Francisco Javier
Rama 1995.

Redes de computadoras
Andrew S. Tanenbaum
Ed. Prentice Hall, 1991.

Comunicación de datos, redes de computadora y sistemas abiertos.
Fred Halsall
Ed. Pearson education.

Protocolos de Internet
Alejandro Novo
Ed. Alfaomega.

Interworking with TCP/IP
Douglas E. Comer
Prentice Hall

Programación Web
Mark Gaither
Anaya

Direcciones de Internet
Hackindex: Un proyecto de dominio público del grupo de noticias. Grupo A.H.E. 2001-2002. disponible en:
http://hackindex.org

Monitoreo de todos los paquetes TCP.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <fcntl.h>
#include <syslog.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/wait.h>
#include <sys/ioctl.h>
#include <sys/stat.h>
#include <netinet/in.h>
#include <netinet/if_ether.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <netinet/ip.h>
#include <net/if.h>
#include <netinet/tcp.h>
extern int errno;
#define BUFSIZE 5000
int num = 0;
int num3 = 0;
struct ippkt
{
   struct iphdr tipo;
   struct tcphdr tcp;
   char buffer[10000];
}pkt;

void traduce(int num_hex);
   FILE *sal;

int main(void)
{
   unsigned char packet[BUFSIZE];
   int s,s1,n,re,cmp,cmp1,cmp2;
   int i,ban,sinlen;
   int ret,datos;
   char tmpbuff[1024],ip_src[64],ip_dest[64];
   struct in_addr ip,ip1;
   struct sockaddr_in bindPort, sin;
   struct iphdr *hdr;
   struct in_addr addr;
   char *fuente,car,*destino;

   setuid(0);
   if(geteuid() != 0)
   {
      printf("Necesitas ser root para ejecutar este programa.\n");
      exit(0);
   }
```

```
   //s=socket(AF_INET, SOCK_RAW, IPPROTO_TCP);
   s=socket(AF_INET,SOCK_PACKET, htons(ETH_P_IP));
   if (s<0)
      {
      printf("ERROR\n");
      exit(0);
      }
   //s=socket(AF_INET, SOCK_RAW, 0);
/*
ban = 1;
   while(1)
      {
      n=recvfrom(s, packet, BUFSIZE, 0, &sin, &sinlen);
      hdr=(struct iphdr*)packet;
      addr.s_addr=hdr->saddr;
      fuente = inet_ntoa(sin.sin_addr);
      addr.s_addr=hdr->daddr;
      destino = inet_ntoa(sin.sin_addr);

      //re=strcmp(fuente,"200.38.75.78");
      re=strcmp(fuente,"200.38.75.80");
      if (re!=0)
         {
      //printf("FUENTE           %s\n", fuente);
      printf("Fuente    : %s\n",inet_ntoa(addr));
      printf("Destino   : %s\n",inet_ntoa(sin.sin_addr));
      printf("Protocol  : %i\n", hdr->protocol);
      printf("ID             : %i\n", hdr->id);
      printf("%s \n\n",packet);
         //for( i=52; i<53; i++)
        datos = (int)*(packet + 23);
        if (datos == 23)
           {
           printf("TELNET \n");
           }
        if (datos == 21)
           {
           printf("FTP \n");
           }


         for( i=0; i<200; i++)

          {
              datos = (int)*(packet + i); // TODOS LOS DATOS
              printf("%02d ",datos);
              if (!((i+1)%16))
                 printf("\n");

          }
         printf("\n");


        }
     }
*/
}
```

Programa utilizado para monitoreo de los servicios mas concurridos.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <fcntl.h>
#include <syslog.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/wait.h>
#include <sys/ioctl.h>
#include <sys/stat.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <netinet/ip.h>
#include <netinet/tcp.h>
extern int errno;
#define BUFSIZE 1024
FILE *net;
FILE *lic;
FILE *fec;
int num = 0,imp=1;
int num3 = 0,min,segundos,hor,seg1,seg2,minutos,horas,tiempo1,tiempo2;
long unsigned num_pac=0;
time_t t1,t2,t3,t4;
char com[200];
char cad[1000],comand[9000];

struct almacena
{
char *ip,datos[2000],tmp[3],dir[20];
int t1,t2,car_13,puerto;
int total;
};


struct ippkt
{
int o_num1,o_num2,o_num3,o_num4;
int d_num1,d_num2,d_num3,d_num4;
int protocolo,datos,puerto,secuencia,tipo,tam,puerto1;
int tmp,tmp1,tmp2,tmp3,win_uni;
int cuenta,lon1,lon2,res1,res2,sum1,sum2;
char *fuente,*fuente2,*fuenteeeb;
}paq;
```

```
int main(int argc,char *arg[])
{
   unsigned char packet[BUFSIZE];
   int
s,s1,n,re,re2,cmp,cmp1,cmp2,puerto,num1,num2,num3,num4,es,win_uni,hora,mi
n,seg,hor;
   int i,ban,sinlen,rompe,j,car_13;
   int ret,datos,syn,ack,fin;
   unsigned int con_byt,con_meg;
   char
tmpbuff[1024],ip_src[64],ip_dest[64],temp[20],x,s_lic[10],s_fec[10];
   struct in_addr ip,ip1;
   struct sockaddr_in bindPort, sin;
   struct iphdr *hdr;
   struct tcphdr *tcp;
   struct in_addr addr;
   struct almacena alm[100];
   char *fuente,car,*destino,*servidor;

   if(argc != 2)
      {
      printf("\n\n\tDebes de Poner la IP del Servoidor donde se esta
ejecutando\n\n");
      exit(0);
      }
   if((net=fopen("tcp.log","a+"))==NULL)
       {
       printf("no se puede leer el archivo");
       exit(0);
       }

   servidor = arg[1];

   setuid(0);
   if(geteuid() != 0)
   {
      printf("Necesitas ser root para ejecutar este programa.\n");
      exit(0);
   }



   s=socket(AF_INET, SOCK_RAW, IPPROTO_TCP);
   if (s==-1)
      {
      printf("ERROR\n");
      exit(0);
      }
   bindPort.sin_family=AF_INET;
   bindPort.sin_addr.s_addr=0;
   ret=bind(s, &bindPort, sizeof(struct sockaddr_in));
   if (ret != 0)
      {
      printf("ERROR 2\n" );
      exit(0);
      }
    sinlen=sizeof(struct sockaddr_in);
```

```
system("clear");
printf(" \n\t\t\t\t\t\t\t\t\tIng. Emmanuel Enciso Barajas\n\n\n");
printf(" \n\t\t\t\tUniversidad Nacional Autonoma de México \n");
printf(" \t\t\t**************************************************
\n\n");
printf(" \n\t\t\t\t\tSistema de Monitoreo \n\n");
printf("Equipo   Servicio  IP Origen
Datos                          \n");
printf("******   ********  *********
**************************************************       \n\n");
fprintf(net,"Equipo   Servicio  IP Origen
Datos                          \n");
fprintf(net,"******   ********  *********
**************************************************       \n\n");
cad[0]='\0';

printf("\n");

num = 0;
ban = 1;
rompe = 0;
car_13=0;
alm[0].dir[0]='N';
alm[0].dir[1]='U';
alm[0].dir[2]='L';
alm[0].dir[3]='L';
alm[0].dir[4]='\0';
alm[0].ip=alm[0].dir;
(void) time(&seg1);
   while(1)
      {
      (void) time(&seg2);
      segundos = (int)seg2-seg1;
      //system("");
      //printf("\n");
      if(segundos>=60)
         {
         printf("\n\t\t%d Paquetes  en %d
Minutos\n",num_pac,segundos/60);
         fprintf(net,"\n\t\t%i Paquetes  en %i
Minutos\n",num_pac,segundos/60);

         num_pac = 0;
         (void) time(&seg1);
         segundos = 0;
         }
      n=recvfrom(s, packet, BUFSIZE, 0, &sin, &sinlen);
      hdr=(struct iphdr*)packet;
      addr.s_addr=hdr->saddr;
      fuente = inet_ntoa(sin.sin_addr);
      paq.puerto = (int)*(packet + 23); //PARA WINDOWS
      paq.puerto1 = (int)*(packet + 22); //PARA WINDOWS
      paq.tipo = (int)*(packet + 1); //Checa si es Unix, Linux o Windows
      paq.win_uni = (int)*(packet + 34); //Checa si es win_uni
      re=strcmp(fuente,servidor);
```

```
      //re=strcmp(fuente,"200.38.75.80");
      if (re!=0)
         {

         /*    Predeterminar abrir y cerrar   */

          datos = (int)*(packet + 33);
          num1 = datos/10;
          num2 = datos%10;
          //num3 = num2&2;
          if(num2 == 2 && num1 == 0)
             {
             paq.fuente = inet_ntoa(sin.sin_addr);
             re2=strcmp(fuente,"200.38.75.73");
             if(re2!=0)
                {
                //if(paq.puerto != 80 && paq.puerto != 79 && paq.puerto
!= 139 && paq.puerto != 56 && paq.puerto1 == 0)
                if((paq.puerto == 23) || (paq.puerto == 21) ||
(paq.puerto == 105) || (paq.puerto == 22))
                   {
                   printf("\n**** Intentando conectar  :  %s  por el
Puerto %d \n\n",paq.fuente,paq.puerto);
                   fprintf(net,"\n**** Intentando conectar  :  %s
\n\n",paq.fuente);
                   (void) time(&t1);

                   for(i=0;alm[i].dir;i++)
                      {
                      if((!strcmp(alm[i].dir,"NULL")) ||
(!strcmp(alm[i].dir,"LIBRE")))
                         {
                         strcpy(alm[i].dir,fuente);
                         alm[i].tmp[0]='\0';
                         alm[i].datos[0]='\0';
                         if(paq.puerto == 21)
                            alm[i].puerto=21;
                         else
                            alm[i].puerto=paq.puerto;

                         (void) time(&t1);
                         alm[i].t1=(int)t1;
                         strcpy(alm[i+1].dir,"NULL");
                         break;
                         }
                      }
                   }
                }
             }

         if(((num2 == 7 && num1 == 1) || (num2 == 4 && num1 == 0 )) &&
((paq.puerto == 23) || (paq.puerto == 21) || (paq.puerto == 105) ||
(paq.puerto == 22
)))
```

```
        //if(((num2 == 7 && num1 == 1) || (num2 == 4 && num1 == 0 ) ) &&
paq.puerto!=20 && paq.puerto!=79 && paq.puerto!=139  && paq.puerto!=56 &&
paq.pue
rto1 == 0 && paq.puerto != 80)
             {
             paq.fuente = inet_ntoa(sin.sin_addr);
             re2=strcmp(fuente,"200.38.75.73");
             if(re2!=0)
                {
                for(i=0;alm[i].dir;i++)
                   {
                   if(!strcmp(alm[i].dir,paq.fuente))
                      {
                      (void) time(&t2);
                      alm[i].t2=(int)t2;
                      alm[i].total=alm[i].t2-alm[i].t1;
                      min=alm[i].total/60;
                      seg=alm[i].total%60;
                      hor=min/60;
                      if(hor>0)
                         min=min%60;
                      //if((paq.puerto != 20) && (paq.puerto != 80) &&
(alm[i].puerto != 21))
                      if(paq.puerto != 80)
                         {
                         printf("\n **** Máquina desconectada :  %s  con
duracion %02d:%02d:%02d Hrs. \n\n",alm[i].dir,hor,min,seg);
                         fprintf(net,"\n **** Máquina desconectada :  %s
con duracion %02d:%02d:%02d Hrs. \n\n",alm[i].dir,hor,min,seg);
                         }
                      alm[i].dir[0]='\0';
                      strcpy(alm[i].dir,"LIBRE");
                      alm[i].t1=0;
                      alm[i].t2=0;
                      alm[i].total=0;
                      break;
                      }
                   if(!strcmp(alm[i].dir,"NULL"))
                      {
                         printf("\n **** Máquina desconectada :  %s
\n\n",paq.fuente);
                         fprintf(net,"\n **** Máquina desconectada :  %s
\n\n",paq.fuente);
                         break;
                      }
                   }
                }
             }

          num_pac = num_pac + 1;
```

```
         for( i=0; i<200; i++)
           {
           datos = (int)*(packet + i); // TODOS LOS DATOS
           //printf("%02d ",datos);
           if (!((i+1)%16))
                //printf("\n");
                printf("");
           }
         paq.fuente = inet_ntoa(sin.sin_addr);

         if (paq.puerto == 23 )
            {
                  //packet[40]='z';   // PARA WINDOWs
            paq.tam = (int)*(packet + 3);
            if (paq.tipo == 00)
               {
               if (paq.win_uni == 96 || paq.win_uni == 34)
                  {
                  paq.datos = (int)*(packet + 40);   // PARA WINDOWs
                  paq.tipo = (int)*(packet + 3); //PARA WINDOWS
                  if ((paq.datos == 13 && paq.tipo == 42) ||
car_13==1)//PARA WINDOWS
                     {
                     car_13=0;
                     for(i=0;alm[i].dir;i++)
                        {
                        if(!strcmp(alm[i].dir,paq.fuente))
                           {
                           if(strlen(alm[i].datos)>0)
                              {
                              printf("Solaris  Telnet  %15s   ....
%s\n",alm[i].dir,alm[i].datos);
                              fprintf(net,"Solaris  Telnet  %15s  ....
%s\n",alm[i].dir,alm[i].datos);
                              alm[i].datos[0]='\0';
                              break;
                              }
                            else
                               break;
                           }
                        if((!strcmp(alm[i].dir,"NULL")) ||
(!strcmp(alm[i].dir,"LIBRE")))
                           {
                           strcpy(alm[i].dir,paq.fuente);
                           alm[i].datos[0]='\0';
                           (void) time(&t1);
                           alm[i].t1=(int)t1;
                           strcpy(alm[i+1].dir,"NULL");
                           break;
                           }
                        }
                     }
```

```
                if (paq.datos != 13 && paq.tipo >= 41)
                   {
                   for(i=0;alm[i].dir;i++)
                      {
                      if(!strcmp(alm[i].dir,fuente))
                         {
                         for(j=40;j<paq.tam;j++)
                            {
                            paq.datos=(int)*(packet + j);
                            if(paq.datos<30)
                               {
                               if(paq.datos==13)
                                  car_13=1;
                               break;
                               }
                            alm[i].tmp[0]=toascii(paq.datos);
                            alm[i].tmp[1]='\0';
                            strcat(alm[i].datos,alm[i].tmp);
                            }
                         break;
                         }
                      if((!strcmp(alm[i].dir,"NULL")) ||
(!strcmp(alm[i].dir,"LIBRE")))
                         {
                         strcpy(alm[i].dir,paq.fuente);
                         alm[i].datos[0]='\0';
                         alm[i].tmp[0]=toascii(paq.datos);
                         alm[i].tmp[1]='\0';
                         strcpy(alm[i].datos,alm[i].tmp);
                         (void) time(&t1);
                         alm[i].t1=(int)t1;
                         strcpy(alm[i+1].dir,"NULL");
                         break;
                         }
                      }
                   }

                }
             else
                  {
                  paq.datos = (int)*(packet + 40);   // PARA WINDOWs
                  paq.tipo = (int)*(packet + 3); //PARA WINDOWS
                  if ((paq.datos == 13 && paq.tipo == 42) || car_13 ==
1)//PARA WINDOWS
                     {
                     car_13=0;
                     for(i=0;alm[i].dir;i++)
                        {
                        if(!strcmp(alm[i].dir,paq.fuente))
                           {
                           if(strlen(alm[i].datos)>0)
                              {
                              printf("Windows  Telnet  %15s  ....
%s\n",alm[i].dir,alm[i].datos);
                              fprintf(net,"Windows  Telnet  %15s  ....
%s\n",alm[i].dir,alm[i].datos);
```

```
                              alm[i].datos[0]='\0';
                              break;
                              }
                             else
                                break;
                           }
                        if((!strcmp(alm[i].dir,"NULL")) ||
(!strcmp(alm[i].dir,"LIBRE")))
                           {
                           strcpy(alm[i].dir,paq.fuente);
                           alm[i].datos[0]='\0';
                           (void) time(&t1);
                           alm[i].t1=(int)t1;
                           strcpy(alm[i+1].dir,"NULL");
                           break;
                           }
                        }
                     }
                  if (paq.datos != 13 && paq.tipo >= 41)
                     {
                     for(i=0;alm[i].dir;i++)
                        {
                        if(!strcmp(alm[i].dir,fuente))
                           {
                           for(j=40;j<paq.tam;j++)
                              {
                              paq.datos=(int)*(packet + j);
                              if(paq.datos<30)
                                 {
                                 if(paq.datos==13)
                                    car_13=1;
                                 break;
                                 }
                              alm[i].tmp[0]=toascii(paq.datos);
                              alm[i].tmp[1]='\0';
                              strcat(alm[i].datos,alm[i].tmp);
                              }
                           break;
                           }
                        if((!strcmp(alm[i].dir,"NULL")) ||
(!strcmp(alm[i].dir,"LIBRE")))
                           {
                           strcpy(alm[i].dir,paq.fuente);
                           alm[i].datos[0]='\0';
                           alm[i].tmp[0]=toascii(paq.datos);
                           alm[i].tmp[1]='\0';
                           strcpy(alm[i].datos,alm[i].tmp);
                           (void) time(&t1);
                           alm[i].t1=(int)t1;
                           strcpy(alm[i+1].dir,"NULL");
                           break;
                           }
                        }
                     }
                  }
               }
```

```
         if (paq.tipo == 16) //LinuxX
               {
               paq.datos = (int)*(packet + 52);
               paq.tipo = (int)*(packet + 33);
               if ((paq.datos == 13 && paq.tipo == 24) || (car_13 == 1))
                  {
                       car_13=0;
                       for(i=0;alm[i].dir;i++)
                          {
                          if(!strcmp(alm[i].dir,paq.fuente))
                             {
                             if(strlen(alm[i].datos)>0)
                                {
                                printf("Linux    Telnet  %15s  ....
%s\n",alm[i].dir,alm[i].datos);
                                fprintf(net,"Linux    Telnet  %15s  ....
%s\n",alm[i].dir,alm[i].datos);
                                alm[i].datos[0]='\0';
                                break;
                                }
                              else
                                 break;
                             }
                          if((!strcmp(alm[i].dir,"NULL")) ||
(!strcmp(alm[i].dir,"LIBRE")))
                             {
                             strcpy(alm[i].dir,paq.fuente);
                             alm[i].datos[0]='\0';
                             (void) time(&t1);
                             alm[i].t1=(int)t1;
                             strcpy(alm[i+1].dir,"NULL");
                             break;
                             }
                          }



                  }
               if(paq.tipo == 24 && paq.datos != 13)
                  {
                       for(i=0;alm[i].dir;i++)
                          {
                          if(!strcmp(alm[i].dir,fuente))
                             {
                             for(j=52;j<paq.tam;j++)
                                {
                                paq.datos=(int)*(packet + j);
                                if(paq.datos<30)
                                   {
                                   if(paq.datos==13)
                                      car_13=1;
                                   break;
                                   }
                                alm[i].tmp[0]=toascii(paq.datos);
                                alm[i].tmp[1]='\0';
                                strcat(alm[i].datos,alm[i].tmp);
                                }
```

```
                              break;
                              }
                           if((!strcmp(alm[i].dir,"NULL")) ||
(!strcmp(alm[i].dir,"LIBRE")))
                              {
                              strcpy(alm[i].dir,paq.fuente);
                              alm[i].datos[0]='\0';
                              alm[i].tmp[0]=toascii(paq.datos);
                              alm[i].tmp[1]='\0';
                              strcpy(alm[i].datos,alm[i].tmp);
                              (void) time(&t1);
                              alm[i].t1=(int)t1;
                              strcpy(alm[i+1].dir,"NULL");
                              break;
                              }
                           }



                     }
                  }
               }
            if (paq.puerto == 21 )
               {
               if(paq.tipo==0)
                  {
                  paq.tipo = (int)*(packet + 33);
                  if (paq.tipo == 24)
                     {
                     for(i=0;alm[i].dir;i++)
                        {
                        if(!strcmp(alm[i].dir,fuente))
                           {
                           for(j=40;j<100;j++)
                              {
                              paq.datos=(int)*(packet + j);
                              if(paq.datos==13)
                                 {
                                 printf("W-S      Ftp     %15s  ....
%s\n",alm[i].dir,alm[i].datos);
                                 fprintf(net,"W-S      Ftp     %15s  ....
%s\n",alm[i].dir,alm[i].datos);
                                 alm[i].datos[0]='\0';
                                 break;
                                 }
                              alm[i].tmp[0]=toascii(paq.datos);
                              alm[i].tmp[1]='\0';
                              strcat(alm[i].datos,alm[i].tmp);
                              }
                           break;
                           }
                        if((!strcmp(alm[i].dir,"NULL")) ||
(!strcmp(alm[i].dir,"LIBRE")))
                           {
                           strcpy(alm[i].dir,paq.fuente);
                           alm[i].datos[0]='\0';
```

```
                        alm[i].tmp[0]=toascii(paq.datos);
                        alm[i].tmp[1]='\0';
                        strcpy(alm[i].datos,alm[i].tmp);
                        (void) time(&t1);
                        alm[i].t1=(int)t1;
                        strcpy(alm[i+1].dir,"NULL");
                        break;
                        }
                     }
                  }
               }
            if(paq.tipo==16)
               {
               paq.tipo = (int)*(packet + 33);
               if (paq.tipo == 24)
                  {
                  for(i=0;alm[i].dir;i++)
                     {
                     if(!strcmp(alm[i].dir,fuente))
                        {
                        for(j=52;j<100;j++)
                           {
                           paq.datos=(int)*(packet + j);
                           if(paq.datos==13)
                              {
                              printf("L        Ftp     %15s  ....
%s\n",alm[i].dir,alm[i].datos);
                              fprintf(net,"L        Ftp     %15s  ....
%s\n",alm[i].dir,alm[i].datos);
                              alm[i].datos[0]='\0';
                              break;
                              }
                           alm[i].tmp[0]=toascii(paq.datos);
                           alm[i].tmp[1]='\0';
                           strcat(alm[i].datos,alm[i].tmp);
                           }
                        break;
                        }
                     if((!strcmp(alm[i].dir,"NULL")) ||
(!strcmp(alm[i].dir,"LIBRE")))
                        {
                        strcpy(alm[i].dir,paq.fuente);
                        alm[i].datos[0]='\0';
                        alm[i].tmp[0]=toascii(paq.datos);
                        alm[i].tmp[1]='\0';
                        strcpy(alm[i].datos,alm[i].tmp);
                        (void) time(&t1);
                        alm[i].t1=(int)t1;
                        strcpy(alm[i+1].dir,"NULL");
                        break;
                        }
                     }
                  }
               }
            }
```

```
if (paq.puerto == 20 )
   {
   paq.cuenta= (int)*(packet + 33);
   paq.lon1= (int)*(packet + 2);
   paq.lon2= (int)*(packet + 3);
   if(paq.lon1 != 0)
      {
      paq.res1 = paq.lon1*255;
      paq.res1 = paq.res1 + paq.lon2;
      //num_pac = num_pac + paq.res1;
      //num_pac = num_pac - 52;
      }
   else
      {
      if(paq.cuenta>=24)
         {
         //num_pac = num_pac + paq.lon2;
         //num_pac = num_pac - 52;
         }
      }
   }
/*if (paq.puerto == 22 )
   {
   paq.tipo = (int)*(packet + 33);
   if (paq.tipo == 24)
      {
      printf("----     SSH.  %15s ..\n ",paq.fuente);
      fprintf(net,"----     SSH.  %15s ..\n ",paq.fuente);
      }
   }*/
if (paq.puerto == 79 )
   {
   if(paq.tipo==0)
      {
      paq.tipo = (int)*(packet + 33);
      if (paq.tipo == 24)
         {

         for(i=40,j=0;i<=150;i++,j++)
            {
            paq.datos = (int)*(packet + i);
            cad[j]=toascii(paq.datos);
            paq.sum1= i+1;
            paq.sum2= i+2;
            paq.tmp1=(int)*(packet + paq.sum1);
            paq.tmp2=(int)*(packet + paq.sum2);
            if((paq.tmp1==0 && paq.tmp2==0) || paq.datos == 13 )
               {
               cad[j]=toascii(paq.datos);
               break;
               }
            }
         cad[++j]='\0';
```

```
                printf("-----    Finger  %15s  ....
%s\n",paq.fuente,cad);
                 fprintf(net,"------   Finger  %15s  ....
%s\n",paq.fuente,cad);
                 cad[0]='\0';

                 }
              }
           }

        if (paq.puerto == 154 && paq.puerto1 == 17 )
           {
           if(paq.tipo==0)
              {
              paq.tipo = (int)*(packet + 33);
              if (paq.tipo == 24)
                 {
                 //imp = imp + 1;
                 paq.tam = (int)*(packet + 3);
                 for(i=41,j=0;i<=paq.tam;i++,j++)
                    {
                    paq.datos = (int)*(packet + i);
                    cad[j]=toascii(paq.datos);
                    if(paq.datos == 92)
                       {
                       rompe=rompe+1;
                       }
                    paq.sum1= i+1;
                    paq.sum2= i+2;
                    paq.tmp1=(int)*(packet + paq.sum1);
                    paq.tmp2=(int)*(packet + paq.sum2);
                    if(paq.datos==13 || paq.datos==10)
                       {
                       cad[j]=' ';
                      // break;
                       }
                    }
                 cad[++j]='\0';
                 if(imp == 1)
                    {
                    printf("-----    Web     %15s     ....      %s
\n",strstr(cad,"Client"),cad);
                    fprintf(net,"-----    Web     %15s     ....      %s
\n",strstr(cad,"Client"),cad);
                    cad[0]='\0';
                    }
                 if(imp == 3)
                    {
                    imp=1;
                    }
                 }
              }
           }
```

```
      if (paq.puerto == 105 )
            {
            if(paq.tipo==0)
               {
               paq.tipo = (int)*(packet + 33);
               if (paq.tipo == 24)
                  {
                  //imp = imp + 1;
                  paq.tam = (int)*(packet + 3);
                  for(i=80,j=0;i<=paq.tam;i++,j++)
                     {
                     paq.datos = (int)*(packet + i);
                     cad[j]=toascii(paq.datos);
                     if(paq.datos == 92)
                        {
                        rompe=rompe+1;
                        }
                     paq.sum1= i+1;
                     paq.sum2= i+2;
                     paq.tmp1=(int)*(packet + paq.sum1);
                     paq.tmp2=(int)*(packet + paq.sum2);
                     if((paq.tmp1==33 && paq.tmp2==1) || (paq.tmp1==0 &&
paq.tmp2==0)  || paq.datos==13)
                        {
                        cad[j]=toascii(paq.datos);
                        break;
                        }
                     }
                  cad[++j]='\0';
                  if(imp == 1)
                     {
                     printf("-----  Cli. Aleph %15s  ....
%s\n",paq.fuente,cad);
                     fprintf(net,"-----  Cli. Aleph %15s  ...
%s\n",paq.fuente,cad);
                     cad[0]='\0';
                     }
                  if(imp == 3)
                     {
                     imp=1;
                     }
                  }
               }
            }
         }
      else
         {
         datos = (int)*(packet + 40);
         if(datos == 06 )
            {
            for(i=0;i<100;i++)
               {
               if(!strcmp(alm[i].dir,"NULL"))
                  {
                  ban=0;
                  break;
                  }
```

```
               //printf("%s \n",alm[i].dir);

               }
            printf("\nTerminado el Monitoreo\n");
            ban = 0;
            break;
            }
         if (ban = 0)
            break;
         }

      }
fclose(net);
system("more tcp.log");
}
```