UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

MONITOREO Y UTILERÍAS DE ADMINISTRACIÓN DE ESTACIONES DE TRABAJO UNIX MEDIANTE UNA COMPUTADORA PERSONAL TRABAJANDO EN AMBIENTE WINDOWS.

T E S I S P R O F E S I O N A L

QUE PARA OBTENER EL TÍTULO DE:

INGENIERO EN COMPUTACIÓN

P R E S E N T A N:

J A V I E R S O L Í S I N F A N T E
A D R I A N R O B E L O J I M E N O
S A L V A D O R R U Í Z G O N Z Á L E Z
CLAUDIA PATRICIA RAMÍREZ QUEZADA
RODRIGO ANTONIO MARTÍNEZ OCARIZ

ASESOR DE TESIS: ING. ALEJANDRO RAMÍREZ LOZADA

MÉXICO D.F. **1995.**

Dedico este trabajo...

A mis padres Francisco y Juanita, por todo el apoyo que me han brindado durante toda mi vida, porque gracias a ellos he logrado diferentes metas.

A todos mis hermanos, a quienes siempre agradeceré que compartan sus experiencias y conocimientos conmigo.

A mi novia Adriana, por haberme apoyado en todo momento para el logro de este importante objetivo.

A todos mis amigos de Bancomer, y en especial a los exintegrantes del área de *Tecnología de Información.*

A todos mis amigos y familiares.

Javier.

Dedico esta Tesis a mi Papa y a mi Mama, quienes han sacrificado mucho de ellos mismos para que yo tenga una educación universitaria y termine este trabajo que para ellos y para mi significa la culminación de un sueño que parecía muy lejano. Agradezco a Dios todos los sacrificios y enseñanzas que ellos me dieron, ya que eso es lo que más valoro en la vida.

A mis hermanos Lourdes, Bety y Lalo quienes siempre han tendido su mano cuando lo he necesitado.

A Martha Flores quien siempre me ha brindado un apoyo y amistad invaluables y a la cual quiero mucho. Le doy gracias por todo lo valioso que me ha enseñado.

A Claudia Ramírez, por la gran amistad que llevamos y por estar siempre conmigo en los momentos difíciles.

A la memoria de mis abuelos Eduardo, Maurita y Pepita a quienes les hubiera gustado estar conmigo agradeciendo a Dios en este momento.

Adrián.

Quiero mencionar antes que todo, a mis padres, por que con ellos fue que se empezó a forjar mi carácter y personalidad, gran parte lo que ahora soy y de lo que, para bien o para mal, creo que ya no dejare de ser; por darme siempre el apoyo, la confianza y la seguridad de un hogar, lo cual me permitió dar una mayor dedicación a la finalización de mis estudios y metas personales. A ellos les dedico esta tesis, todo mi cariño y respeto y la seguridad de que, sin importar la distancia o las circunstancias, siempre a su lado para apoyarlos en lo que se necesite.

A Olivia, quien ha estado conmigo desde hace ya algún tiempo, compartiendo conmigo tristezas y alegrías, ofreciendo de si siempre lo mejor de una manera incondicional.

A mis hermanos, hermanas, mi familia, por ser mi familia y por parte del hogar en el que me forme, donde crecimos cada uno de nosotros y donde creo que esta la parte más importante de cada persona.

Hay muchas personas a las que quisiera mencionar y, de alguna forma, agradecer por todo lo que de ellas he recibido, por brindarme su amistad, por su apoyo, sus consejos y por tantas cosas que de ellas he aprendido, amigos, familiares, profesores, compañeros de trabajo. No quisiera mencionar nombres porque creo que me faltaría espacio, además de que no me gustaría olvidar alguno, no obstante, a todos muchas gracias.

Para todos Uds., todo mi cariño, mi amistad y lo mejor de lo mejor.

Salvador.

Con todo cariño a las personas más importantes de mi vida, los cuales me apoyaron en los momentos difíciles de mi carrera y de mi vida general.

Gracias papá por el espíritu de lucha y el caracter que me forjaste desde pequeña para lograr mis objetivos.

Gracias mamá por la fuerza y el amor que siempre me has dado.

Gracias Manolo por ser buen hermano y escucharme.

Gracias Marita por escucharme, comprenderme y apoyarme.

Gracias Raúl por ayudarme y comprenderme.

Gracias Carlitos por tu alegría, optimismo y cariño.

Y a ti Gabriel por el apoyo, comprensión y cariño.

A la memoria de mi abuelito consentido Galdino, porque a pesar de la distancia, tu espiritu y fuerza viven en mi.

Al Ing. Adolfo Navarro por haber creido en mi.

Los quiero mucho.

Claudia.

A mi esposa Laura, ya que sin su constante motivación y cariño, la conclusión de este trabajo no hubiera sido posible.

A mis padres, cuyas enseñanzas y apoyo han sido siempre un aliciente para seguir mejorando día a día.

A mis hermanos Alejandro, Luis y Santiago, que siempre han estado presentes cuando los he necesitado.

A Claudia, Adrian, Javier y Salvador, ya que sin su ayuda y constancia este trabajo nunca hubiera existido.

Rodrígo.

Queremos Agradecer...

Al Ing. Alejandro Ramírez, por su tiempo, dedicación y paciencia para la realización de este trabajo.

A Xerox Mexicana S.A. de C.V., y especialmente a Ana María Lopez por todos los recursos que amablemente nos proporcionaron.

A Sheldon Smith por su orientación y apoyo para el logro de nuestros objetivos.

A la Sra. Beatriz Jimeno por sus innumerables atenciones para con nosotros durante el proyecto.

Al Ing. César Solís por su aportación intelectual.

A SESO S.A. de C.V. por la facilitación de sus instalaciones.

A La Facultad de Ingeniería y a nuestra Universidad por los conocimientos y experiencias que nos proporcionó.

MONITOREO Y UTILERÍAS DE ADMINISTRACIÓN DE ESTACIONES DE TRABAJO UNIX MEDIANTE UNA COMPUTADORA PERSONAL TRABAJANDO EN AMBIENTE WINDOWS.

CONTENIDO.

INTRODUCCIÓN GENERAL.

La necesidad de intercambiar información y compartir recursos entre sistemas de diferentes arquitecturas y ambientes operativos, aún en lugares remotos, ha marcado la pauta para el desarrollo de redes de computadoras y de las Telecomunicaciones.

Dentro del mundo de las comunicaciones, Unix juega un papel importante, siendo un sistema operativo multiusuario robusto, estándar, con servicios de red integrados y disponible prácticamente en cualquier plataforma de hardware existente en el mercado, es decir, desde una computadora personal hasta una supercomputadora como la Cray que actualmente se encuentra en la Dirección de Cómputo Académico de la UNAM.

Microsoft por su parte, líder en el mercado de los programas para computadoras personales desde hace ya varios años, con la gran variedad de aplicaciones Windows y programas orientados a usuarios finales, todavía tiene una gran peso en el mercado de cómputo, apuntalado con lanzamientos como: WindowsNT Server, WindowsNT Workstation, Windows para Trabajo en Grupo y ahora con la liberación de su nuevo Windows 95, el cual ya le da una definición de sistema operativo gráfico, que entre sus cualidades cuenta con servicios de comunicaciones y manejo de dispositivos integrados.

Pese al gran potencial de los sistemas Unix y su amplio crecimiento en el mercado, en lo que a programas orientados a usuarios finales se refiere, Microsoft con sus aplicaciones Windows, ofrece al público una amplia gama de programas de procesamientos de textos, generación de presentaciones, aplicaciones administrativas y utilerías en general muy completas, con interfaces de usuario cada vez más amigables y fáciles de utilizar. Es por esto que hoy en día es común encontrar Instituciones, Compañías y Centros de Cómputo con infraestructura de red tipo LAN (Local Area Network:-Red de Area Local-) con sistemas Unix y computadoras personales interconectados entre sí. En este tipo de sistemas generalmente Unix se utiliza para aplicaciones muy específicas que requieren de un sistema operativo multiusuario, multitareas y con gran capacidad de procesamiento, como pueden ser por ejemplo, aplicaciones de

bases de datos o servidores de comunicaciones, por otro lado, las computadoras personales han sido utilizadas como herramientas para facilitar y simplificar el trabajo cotidiano, utilizando en su mayoría sistemas bajo Ambientes Windows.

Referente al mantenimiento de los sistemas operativos en este tipo de ambientes, aún cuando haya más de un servidor Unix conectado a la red, no se requiere más de una sola persona para realizar esta labor, cuyas funciones principales serán las de administrar y monitorear cada uno de los servidores Unix, tarea que sería muy fácil contando con una herramienta utilizando una PC en ambiente gráfico, que nos permita: accesar las diferentes estaciones de trabajo, convivir con el resto de las aplicaciones de manera transparente y que no consuma demasiados recursos, etc.

Hoy en día existen en el mercado aplicaciones que permiten realizar algunas funciones de monitoreo y administración de sistemas Unix desde una estación de trabajo, entre los cuales encontramos: HP OpenView y SunNet Manager, los cuales requieren de una infraestructura de hardware y software muy costosa. Algunos otros no tan costosos basados en Windows representan ser básicos en cuanto al monitoreo y administración de estaciones Unix se refiere, tales como: VisiNet, SMS, ManageWise, etc.

De acuerdo a lo mencionado anteriormente, presentamos esta Tesis, la cual tiene como **objetivo** ofrecer al mercado de cómputo una solución que permita realizar el monitoreo y administración de estaciones Unix conectadas en red, de manera eficaz, sencilla y amigable, desde una computadora personal corriendo una aplicación bajo ambiente Windows.

Este documento será de importancia para aquellas personas con interés en la administración de redes, que actualmente cuenten con una infraestructura de red con sistema operativo Unix o para quienes su campo de acción se encuentre en el desarrollo de aplicaciones en ambiente Windows, y en general para todos aquellos estudiantes y/o profesionales que se encuentran en el área de Ingeniería de Sistemas.

El documento completo está dividido en 5 capítulos; cuyo contenido se describe a continuación:

- **Capítulo I** ***.Fundamentos de Protocolos.*** En este capítulo se explicará en forma general algunos conceptos que consideramos importantes para el entendimiento del software desarrollado (Sistema de Monitoreo y Utilerías de Administración de Estaciones de Trabajo Unix mediante una Computadora Personal Trabajando en Ambiente WINDOWS), entre los cuales tenemos: Modelo OSI, términos de sistemas administradores de red, y TCP/IP.

- **Capítulo II.** ***Manejo de Sockets.*** Estará dedicado al estudio de los sockets como interface de comunicación entre procesos, dentro de un ambiente Unix y el papel que desempeñan en una intercomunicación con un sistema personal. Se analizarán los principios fundamentales y la programación de los sockets, así como también las principales llamadas al sistema.

- **Capítulo III.** ***UNIX Esquema General.*** En este capítulo se hablará de la historia de Unix, así como de los componentes básicos del sistema operativo, el manejo de archivos y su implementación con NFS. Se describirán cuales son sus procesos básicos y la función que estos realizan. Así mismo, explicaremos algunas de las funciones principales de un administrador de estaciones Unix, describiendo también la importancia de un sistema de monitoreo y administración.

- **Capítulo IV.** ***Interfaces Gráficas de Usuario (GUI's).*** Se hará mención de algunas de las principales herramientas de programación existentes en el mercado para el desarrollo de interfaces gráficas amigables a usuarios finales. Se analizarán las características de la herramienta elegida.

- **Capítulo V.** ***Aplicación.*** Representará la conclusión práctica con base en el marco teórico descrito en los capítulos anteriores, conteniendo el detalle del producto final que será nuestra aportación al mercado de las herramientas de monitoreo y administración de estaciones Unix.

- ***Conclusiones.*** Daremos nuestro punto de vista final sobre el sistema desarrollado así como los comentarios y experiencias adquiridas durante la elaboración de dicho software. Haremos mención de las características generales de este tipo de sistemas y su tendencia en el mercado.

Capítulo I. Fundamentos de Protocolos.

Sabemos que hoy en día las comunicaciones de datos representan una parte fundamental en el ambiente computacional. Observamos también la necesidad de grupos de personas de compartir información en ambientes totalmente heterogéneos (hardware y software de diferentes proveedores), gente que a manera de entretenimiento, actualización y cultura intercambian software mediante una red y bajo ciertos estándares de intercambio de información. En lo que se refiere al mundo científico, el intercambio de información representa una parte esencial para el desarrollo de la ciencia.

I.1 Modelo OSI.

La comunicación e intercambio de datos entre máquinas de diferentes diseños fue una tarea difícil de superar. A principios de los 80´s la Organización Internacional de Estándares (ISO International Organitation for Standarization) reconoció la necesidad de un modelo de red que apoyara a los diferentes proveedores de tecnología. Fue así como en 1984 surge el modelo OSI, el cual rápidamente se puso como el modelo principal para la comunicación entre computadoras.

En la figura 1 mostraremos la implementación de los protocolos, así como su papel en las diferentes capas del modelo OSI, del cual detallaremos más adelante.

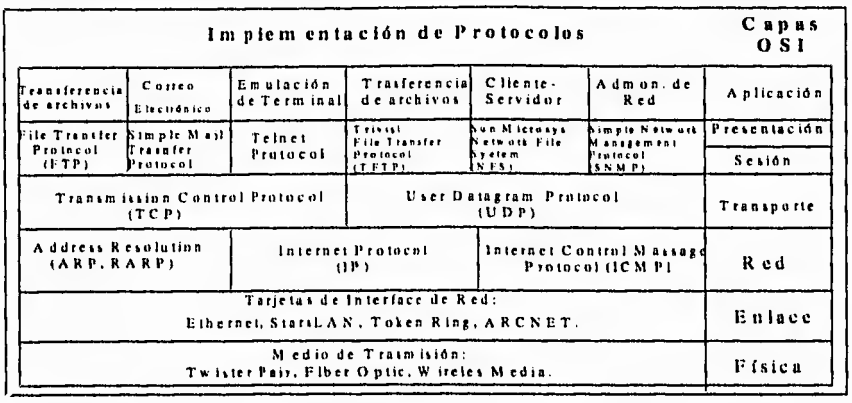


Fig.1 Implementación de protocolos.

I.1.1 Comunicaciones jerárquicas.

El modelo de referencia OSI divide el problema de intercambio de información entre computadoras sobre la red en siete pequeños y manejables problemas. Cada uno de estos siete problemas fueron divididos para hacer más sencillo su análisis, y son resueltos mediante las "capas del modelo". Actualmente la mayoría de los dispositivos de red implementan las siete capas. Las dos últimas capas de OSI son implementadas mediante hardware y software; las primeras 5 son generalmente implementadas mediante software. El modelo OSI describe como se realiza el viaje de información desde los programas de aplicación de una computadora mediante la red hasta otro programa de aplicación en otra computadora.

Explicando el proceso de comunicación entre las diferentes capas tenemos como ejemplo la figura 2, donde se asume que el Sistema X tiene información que enviar al Sistema Y. El programa de aplicación en el Sistema X se comunica con la capa 7 (capa superior), la cual se comunica con la capa 6 del Sistema X, y la cual también se comunica con la capa 5 del mismo sistema, y así sucesivamente hasta comunicarse con la capa 1. La capa 1 está encargada de entregar y recoger información del medio físico de la red. Después de que la información pasa por la red, el Sistema Y la absorbe y se realiza el proceso anterior, pero en orden inverso, es decir, primeramente obtiene información la

capa 1, la cual se comunica con la capa 2, y así sucesivamente hasta que llega al programa de aplicación en el sistema Y.

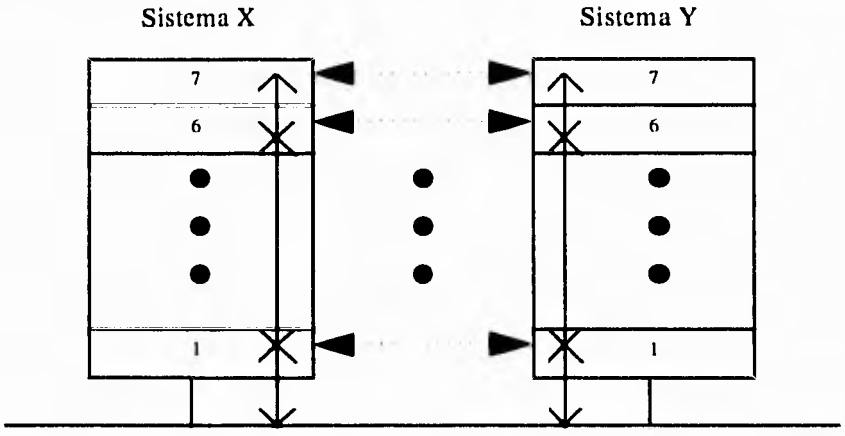


Fig. 2 Comunicación entre dos sistemas de computadoras

Además de que cada una de las capas del sistema X se comunica con su adyacente en el mismo sistema, su principal objetivo es comunicarse con el mismo nivel de capa en el sistema Y. Esto es, el principal objetivo de la capa 1 en el Sistema X es comunicarse con la capa 1 en el Sistema Y, la capa 2 del Sistema X con la capa 2 del sistema Y, y así sucesivamente.

La relación de comunicación entre las capas adyacentes de un mismo sistema se muestra en la figura 3.

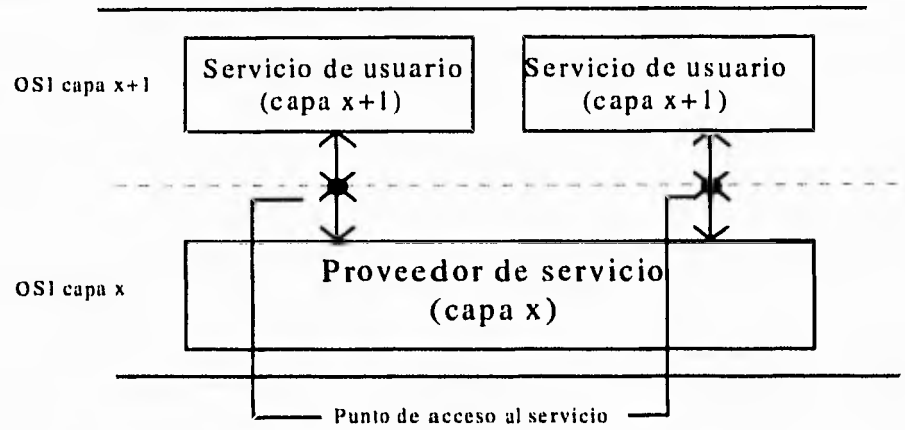


Fig.3 Relación entre las capas adyacentes de un sistema.

I.1.2 Formatos de Información.

Para saber lo que desea la capa "n" del sistema X en la misma capa del sistema Y, se especifican peticiones que son almacenadas como control de información, la cual es pasada entre las diferentes capas mediante un encabezado con el nombre de "header".

I.1.3 Capas.

Una vez que se ha entendido el proceso de comunicación entre las capas y el intercambio de información entre dos aplicaciones, es necesario entender el papel que juegan cada una de ellas como se observó en la figura 1.

Aplicación. La capa de aplicación en el modelo OSI, es la más cercana al usuario, se puede diferenciar de las capas restantes en que no provee servicios a otras capas, pero sin embargo representa la aplicación en sí, como por ejemplo procesadores de texto, hojas de cálculo, etc.

Presentación. Esta se asegura de que la información enviada por la capa de aplicación de un sistema, será leída por la misma de otro sistema. Si es necesario intercambia múltiples datos mediante un formato de representación común.

Sesión. Como su nombre lo indica, esta capa es encargada de establecer, administrar, y terminar la sesión entre las aplicaciones. El término sesión consiste en un diálogo entre dos o más presentaciones.

Transporte. Lo que refiere a las capas de aplicación, presentación, y sesión, tienen que ver directamente con la aplicación; por otro lado las capas restantes tienen que ver directamente con los problemas de transporte de datos. La capa de transporte como su nombre lo indica, provee el servicio de transporte. Provee mecanismos para establecer y mantener *circuitos virtuales*[1].

Red. La capa de red provee la conectividad y la selección de la ruta entre dos sistemas que pueden estar localizados geográficamente en diferentes subredes.

Enlace. La capa de enlace provee una transmisión confiable de los datos por medio de la capa física. Interactua con la primera capa en cuestiones como: la topología, notificación de errores, envío de frames, control de flujo, etc.

Física. En este nivel se definen puntos como: términos eléctricos, niveles de voltaje, conexiones físicas, y cuestiones similares que mantienen la actividad en una red.

I.2 Conceptos y términos importantes.

En términos de redes, como en otras áreas, se cuenta con una terminología y base de conocimientos que es propietaria. Es por este motivo que presentaremos algunos conceptos que son de importancia para su mejor entendimiento.

I.2.1 Direccionamiento.

La localización de los nodos en una red es un componente esencial en cualquier sistema de redes. Existen diversos sistemas de direccionamiento, los cuales

[1] Circuitos creados y que existen lógicamente, no son circuitos físicos.

dependen del protocolo usado. Como ejemplo, el direccionamiento en Apple Talk es diferente del direccionamiento en TCP/IP.

Los dos tipos de direccionamiento son: de enlace y de red. La dirección en la capa de enlace es única para cada conexión de red. Para la mayoría de las LAN's, esta dirección residente en el circuito virtual y su asignación esta determinada por los estándares establecidos (I.E.E.E.). La dirección en la capa de red es una dirección lógica y son este tipo de direcciones las que permiten construir una red. (Dependiendo del protocolo utilizado).

I.2.2 Frames, paquetes y mensajes.

Una vez que se tiene localizado el sistema, la información puede ser intercambiada entre dos o más sistemas.

El término ***frame*** denota una unidad de información la cual su origen y destino es establecido en la capa de enlace. El término ***paquetes*** denota una unidad de información cuyo origen y destino está dado en la capa de red. Finalmente el término ***mensaje*** es usado para referenciar una información particular con un propósito bien definido.

I.3 Conceptos básicos de sistemas administradores de red.

El término administración de red tiene diferentes significados. A principios de los 80's se notaba una gran expansión en el desarrollo de las redes. Las compañías realizaban análisis costo-beneficio sobre las diferentes tecnologías surgidas, fue así como empezaron a añadir diferentes productos para beneficio de las redes.

Entendemos por sistemas administradores de red, a aquellos que son capaces de tomar información de las estaciones de trabajo con el fín de analizar sus datos, determinar eventos críticos, y actualizar parámetros específicos en las mismas .

El principal problema asociado con la expansión de las redes, es la administración diaria y la planeación del crecimiento. Específicamente, cada nueva tecnología de red requiere de expertos propios para la operación.

I.3.1 Arquitectura de administración.

La mayoría de las arquitecturas de administración de redes, usan una estructura básica y un conjunto de relaciones. Nodos (dispositivos administrados) tales como computadoras y otros dispositivos de red corren un software que permite enviar alertas cuando existen problemas.

En la figura 4 se observa la arquitectura de administración de red.

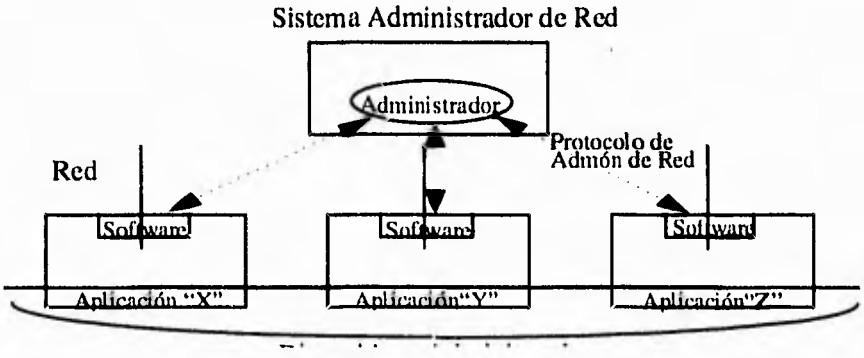


Fig.4 Arquitectura típica de administración de Red.

En este esquema el administrador está monitoreando los dispositivos mediante un software residente en cada máquina. Los sistemas más conocidos son "Simple Network Management Protocol (SNMP)" y "Common Management Information Protocol (CMIP)".

I.3.2 Modelo de administración.

ISO ha contribuido fuertemente en una estandarización de red. Su modelo de administración es el primero que se enfoca a las principales funciones de una red. Este modelo consta de 5 partes que son:

- Administración del Desempeño.
- Administración de la Configuración.
- Administración de Cuentas.
- Administración de Fallas.
- Administración de Seguridad.

Desempeño. El objetivo de la administración del desempeño de una red, es medir y hacer disponible varios aspectos de la misma que garanticen un alto nivel.

Esta administración involucra distintos pasos como:

1. Recopilación de datos de desempeño que son de interés a administradores de red.
2. Análisis de datos que determinan los niveles de desempeño.
3. Determinar el punto máximo de las variables de interés tales que puedan determinar los puntos críticos en una red.

En este tipo de administración continuamente se monitorean las variables de desempeño y en el momento que se excede un valor, se envía una alerta al sistema administrador.

Configuración. El objetivo de la administración de configuración es monitorear la red y contar con la información de software y hardware, debido a que estos elementos son totalmente dinámicos y pueden repercutir en la operación de la red.

Cuentas. El objetivo de la administración de cuentas es medir el acceso a la red. Con este tipo de regulación, se minimizan los problemas y se controlan los accesos de usuarios. Después de contar con un buen desempeño de red, el siguiente punto es contar con una buena administración de los recursos de la misma.

Fallas. La administración de fallas, es detectar, llevar un registro, notificar a los usuarios, así como lograr una reparación automática de los problemas de red.

Dicha administración involucra diferentes pasos como:

1. Determinar los síntomas del problema.
2. Aislar el problema.
3. Reparar el problema.
4. Realizar pruebas de la reparación, y de todos los subsistemas.
5. Registro de la detección y resolución de problemas.

Seguridad. Es lograr un control sobre el acceso a los recursos de la red, de forma tal que se pueda evitar un sabotaje en la red, o bien, se evite la lectura de documentos privados.

Dentro del marco de la seguridad, los subsistemas desempeñan funciones como:

- Identificar los recursos de la red.
- Determinar los medios entre los recursos de la red y el usuario final.
- Monitorear los puntos de acceso a los recursos de la red.
- Contar con un registro de accesos inapropiados a los recursos.

I.3.3 Administradores de Red (SNMP).

SNMP (Simple Network Management Protocol).
Protocolo simple de manejo de redes. El protocolo de manejo de redes Internet, ofrece medios para seguir y determinar la configuración de la red y los parámetros al tiempo de ejecución.

Arquitectura. SNMP surgió del esfuerzo realizado con SGMP (Simple Gateway Monitoring Protocol). Fue desarrollado para ser más eficiente en la administración de red sobre el transporte UDP.

La arquitectura de SNMP cuenta con diferentes elementos como se observa en la figura 5. SNMP cuenta con el software del punto que será administrado.

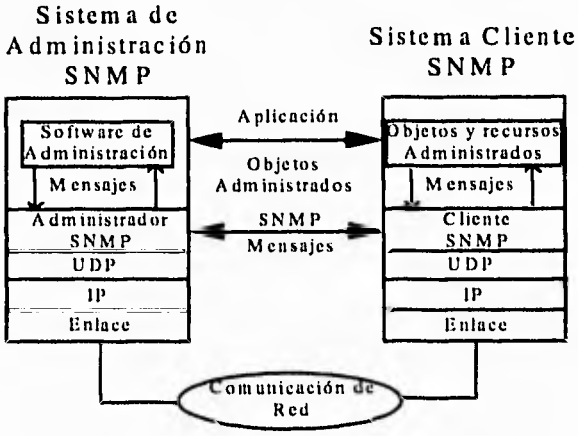


Fig. 5 Arquitectura SNMP.

Mensajes. SNMP representa un protocolo de *poleo*, en el cual el administrador realiza una pregunta y el cliente responde. UDP es el transporte que transmite todos los mensajes de SNMP.
El mensaje de SNMP es seguido del encabezado UDP dentro del frame de transmisión, esto es:

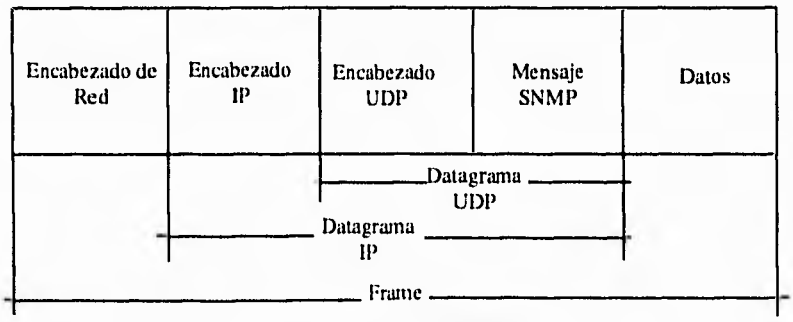


Fig.6 Frame de SNMP.

En lo que refiere al aspecto de ambiente Windows, SNMP representa un protocolo que especifica el formato para la colección de datos necesarios para la administración. Con SNMP en ambiente Windows, es posible utilizar el administrador en nodos Unix o estaciones con TCP/IP.

I.4 TCP / IP

I.4.1 Introducción

El uso de TCP/IP (Transmission Control Protocol / Internet Protocol) se ha convertido en un estándar para la mayoría de las compañías, su desarrollo fue en Estados Unidos en el año de 1969 surgiendo durante un proyecto llamado ARPANET.

El propósito de contar con este protocolo fue proveer las funciones necesarias para enviar los paquetes de Bits desde un origen hasta un destino, sobre sistemas interconectados en Red.

Así entonces en el transcurso del tiempo el proyecto "ARPANET" fué creciendo en el mundo de las Redes, hasta llegar a lo que hoy en día conocemos como "INTERNET".

I.4.2 Estándares en TCP/IP

Los estándares para TCP/IP son publicados en documentos llamados RFC's (Request for Comments). Estos tomos describen el funcionamiento interno, los diferentes servicios, su implementación y políticas en Internet.

I.4.3. TCP (Transmission Control Protocol)

TCP representa uno de los protocolos primarios dentro de TCP/IP. Provee un servicio confiable de transmisión de datos entre dos puntos. TCP juega su papel dentro del modelo OSI en el nivel cuatro (Transporte), observando a los datos como un "STREAM" de bytes, es decir, los datos son transmitidos en segmentos.

Servicio confiable de transmisión significa que por cada segmento de datos que se envía, la máquina que recibe esta información debe regresar una señal de aceptación de los datos, a la cual llamaremos "Acknowledgment (ACK)", así entonces, la seguridad estará establecida por:

1) Si la señal de regreso (ACK) no es recibida, los datos serán retransmitidos.
2) Si el segmento enviado no llega completo, la máquina destino no enviará la señal de aceptación (ACK), provocando la retransmisión del mismo.
3) Formalmente se asigna un número secuencial a cada byte transmitido y se realiza un conjunto por cada segmento.
4) De acuerdo a la secuencia asignada y enviada, se asegurará la cantidad de bytes enviados y recibidos.
5) La señal final dirá a la máquina destino la cantidad de bytes recibidos.

I.4.4 UDP (User Datagram Protocol)

Provee un servicio de datagramas no orientado a conexión, la cual ofrece una entrega poco confiable de los datos enviados al dispositivo destino. Esto significa que la secuencia de datos recibidos puede no ser la especificada por el dispositivo fuente.

UDP es uitilizada en aplicaciones que no requieren del reconocimiento de la llegada de datos.

Como ejemplos de aplicaciones que utilizan UDP tenemos DNR (Domain Name Resolver), NBT (Net Bios sobre TCP/IP), SNMP (Simple Network Management Protocol), y otras aplicaciones basadas en sockets.

I.4.5 IP (Internet Protocol)

El protocolo Internet (IP) fue desarrollado para "proveer las funciones necesarias para enviar los paquetes de bits (en datagramas de Internet) desde un origen hasta un destino sobre sistemas interconectados en red.

IP asume que la entrega de datos puede ser difícil y siempre hará su mejor esfuerzo en la entrega de los mismos.

Así entonces:

- Provee un ruteo de información de una máquina a otra (siendo el ruteo su función primaria).

- Provee un mecanismo de fragmentación y re-ensamblaje de paquetes, de acuerdo a reglas establecidas que le dicen a los ruteadores como manejar los datos.

- Define como los datos son transmitidos a través de la red.

En el proceso de intercomunicación IP, en la capa de red participa como el datagrama, mientras que el encargado de llevar los datos será la capa de transporte, esto es, TCP (Transport Control Protocol).

El término *datagrama* se refiere a los paquetes de datos transmitidos sobre las conexiones de red establecidas, esto significa que se debe establecer la ruta entre el origen y destino, antes del envío de datos.

I.4.6 Direcciones Internet.

Direccionamiento IP. Cada dirección de IP es de 32 bits, dividida en identificadores de "Host" y de "Red". Se pueden tener 5 clases de red (A,B,C,D,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.

Es asignada para redes con un gran número de estaciones. 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 bit restantes (últimos 3 campos) representan la dirección de la estación. Esto nos permite contar con 126 redes y aproximadamente 17 millones de nodos por red.

Clase B.

El direccionamiento en 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 24 bits (los últimos 2

campos) representan el direccionamiento del nodo. Esot nos permite tener 16,384 redes y aproximadamente 65,000 nodos por red.

Clase C.

Es utilizada para redes más pequeñas(Lan´s). 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 por red.

Clase D.

Es usada para multireparto 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 reciben 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 esta reservada para uso futuro, el bit de mayor orden es puesto en 1-1-1-1.

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

I.4.7 Ruteando IP Datagramas.

La principal función de la capa de red es el ruteo. En el se trasladan paquetes de un punto a otro a través de la red.

Cada dispositivo debe contar con una dirección lógica que se identifique como única en la red, y esta dirección es independiente de la dirección del hardware (dirección de 48-bits de Ethernet o de token ring). Es necesario asociar la dirección lógica en un datagrama y a su vez con la dirección lógica en una LAN

o WAN. Seguido de esto se debe distribuir a dispositivos inteligentes, conocidos como ruteadores. Los ruteadores decidirán el camino apropiado para guiar los paquetes. Las dos formas que utilizan los equipos para comunicarse son RIP (Routing Information Protocol) y OSPF (Open Shortest Path First). Finalmente el ICMP (Internet Control Message Protocol) establece un mecanismo de realimentación en la red.

I.5 Aplicaciones

En este punto hablaremos de algunas aplicaciones principales de la capa de aplicación. En esta capa los usuarios interactuan con el nodo y algunas funciones. Esas funciones pueden incluir transferencia de archivos usando Trivial File Transfer Protocol (TFTP), o más complicados como File Transfer Protocol (FTP).

I.5.1 TFTP,FTP.

El protocolo TFTP, lee y escribe archivos o correos desde un nodo a otro. La composición de TFTP es realmente simple, transfiere 512 octetos de datos. Es implementado con el transporte UDP, y no es muy confiable.

TFTP define 5 tipos de paquetes, los cuales son distinguidos por un código de operación (Opcode)

Código de Operación	Operación
1	**Read Request (RRQ)**
2	**Write Request (WRQ)**
3	**Data (DATA)**
4	**Acknowledgement (AC)**
5	**Error (ERROR)**

Los paquetes en la operación 1 y 2 (RRQ, WRQ) cuentan con la misma estructura. El paquete de datos (operación 3) transfiere información. El paquete ACK (operación 4) es usado como reconocimiento, esto es ocasionado porque es basado sobre UDP y tiene que reconocer cada uno de los paquetes. El

paquete de ERROR (operación 5) puede ser usado como de reconocimiento en cualquiera de los 4 tipos de paquetes anteriores.

Uno de los protocolos más usados en la fase de aplicación es File Transfer Protocol (FTP), y como su nombre lo indica, permite compartir archivos en máquinas locales o remotas usando como transporte TCP.

El servicio de FTP incluye al usuario y al servidor, donde el servicio de usuario incluye una interface de usuario (User Interface UI), protocolo intérprete (User Protocol Interpreter UPI), y un proceso de transferencia de datos (User Data Transfer Process DTP). Por lo que se refiere al lado del servidor incluye solamente UPI y DTP, excluyendo la interface de usuario. El protocolo intérprete inicializa la conexión mediante el protocolo TELNET. El usuario usa internamente un número de puerto asignado, para conectarse a un número de puerto en el servidor asignado por el control de FTP.

paquete de ERROR (operación 5) puede ser usado como de reconocimiento en cualquiera de los 4 tipos de paquetes anteriores.

Uno de los protocolos más usados en la fase de aplicación es File Transfer Protocol (FTP), y como su nombre lo indica, permite compartir archivos en máquinas locales o remotas usando como transporte TCP.

El servicio de FTP incluye al usuario y al servidor, donde el servicio de usuario incluye una interface de usuario (User Interface UI), protocolo intérprete (User Protocol Interpreter UPI), y un proceso de transferencia de datos (User Data Transfer Process DTP). Por lo que se refiere al lado del servidor incluye solamente UPI y DTP, excluyendo la interface de usuario. El protocolo intérprete inicializa la conexión mediante el protocolo TELNET. El usuario usa internamente un número de puerto asignado, para conectarse a un número de puerto en el servidor asignado por el control de FTP.

Capítulo II. Manejo de Sockets.

II.1 Conceptos básicos.

Para lograr un buen entendimiento del manejo de sockets, se deben conocer los aspectos generales de la comunicación entre procesos, así como sus propiedades y la relación de los sockets con el modelo OSI.

II.1.1 Comunicación entre procesos (IPC)

El gran auge que tienen hoy en día en el mundo de las redes de cómputo el desarrollo de aplicaciones cliente - servidor, establece nuevas demandas en las habilidades que deben poseer los desarrolladores de las mismas, entre estas incluimos la de dominar la comunicación entre procesos conocida como IPC (Interprocess Communication)

Dentro de un ambiente de red, entendemos como comunicación entre procesos IPC, a la manera de llevar a cabo la transferencia de datos entre dos o más programas. Estos pueden residir o no dentro de la misma computadora o inclusive en computadoras de ambientes diferentes pero que se encuentran o son parte de la misma red.

La mayor parte de las rutinas que componen un IPC son diseñadas para realizar la transferencia de datos entre diferentes dispositivos y no únicamente entre procesos. Este tipo de diseño tiene la ventaja de que permite el modelado de ambientes en los cuales es independiente la ubicación de los puntos finales de comunicación. La mayor parte de los servicios estándar de una red son construidos bajo estos términos.

Modificaciones para IPC´s y aspectos de red fueron la principal adición al sistema Unix, introducidas primeramente en la versión 4.2BSD. Estas modificaciones requirieron algunos cambios en la interface del sistema. La idea básica de esta interface es hacer el IPC similar a la Entrada/Salida de archivos. En el sistema Unix un proceso tiene un conjunto de descriptores de Entrada/Salida, desde el cual uno lee y otro escribe. Los descriptores pueden referirse a archivos normales, a dispositivos (incluyendo terminales). El uso de un descriptor tiene tres fases: A)creación, B)leer y escribir, y C)destrucción. Usando descriptores para escribir archivos, en lugar de simplemente nombrar al archivo en cuestión en la instrucción write, uno gana una sorprendente cantidad de flexibilidad. A menudo, el programa que crea un descriptor será diferente del programa que usa el descriptor. Por ejemplo un shell puede crear un descriptor para la salida del comando **ls** que causará que el listado aparezca en un archivo en lugar de en la pantalla de la terminal.

II.1.2 Propiedades de una Comunicación.

Cualquier aplicación desarrollada y que tenga como base un IPC deberá considerar entre otras cosas los requisitos para una buena comunicación, ya que esto garantiza que la aplicación pueda funcionar adecuadamente. Las características que se deben considerar en una comunicación son las siguientes:

a).-Fiabilidad en la transmisión: Esta característica se refiere a la fiabilidad de que no se volatice la información que se transmite de un sistema a otro.

b).-Conservación del orden de los datos: Seguridad de que los datos llegarán en el orden en que fueron enviados.

c).-No duplicación de datos: Al destino sólo llega un ejemplar de cada dato enviado.

d).-Comunicación en modo conectado: Antes de realizarse la comunicación se establece una conexión lógica entre los dos puntos implicados, una vez establecido dicho canal, el envío de información de un punto está implícitamente destinado al otro punto conectado.

e).- Conservación de los límites de los mensajes: Los límites de los mensajes emitidos se pueden encontrar en el destino de éstos.

f).- Envío de mensajes urgentes: Esta característica define la posibilidad de enviar datos fuera del flujo normal y, por consecuencia, hacerlos accesibles inmediatamente.

II.1.3 El rol de los sockets en el modelo OSI.

Para que se pueda efectuar una **IPC** es necesario un punto o interface donde se realiza el intercambio de información, dicha interface es representada por los **sockets** que trabajan en el **nivel 5 de sesión** del modelo **OSI** y que nos permitirán tener un alto grado de confiabilidad en los mensajes originados en el nivel de aplicación (nivel 7) de la misma forma en que el nivel 2 de enlace asegura la comunicación entre nodos adyacentes. La razón por la que se ubican en este nivel es por que la transmisión de paquetes de información se lleva a cabo desde el punto de origen al punto destino no necesariamente entre nodos adyacentes como sería en el nivel 2.

Lo anterior nos lleva a una primera definición de un socket: Es un objeto abstracto a través del cual un proceso puede enviar o recibir información en la red o dentro del mismo sistema.

II.2 Definición

Los siguientes puntos mostrarán la definición de un socket, así como sus funciones y características principales, realizando una comparación con los WinSockets.

II.2.1 Sockets y WinSockets.

El bloque básico para la construcción de una comunicación es el Socket. Un ***socket*** es un punto final de la comunicación. Cada socket en uso tiene un tipo y un proceso asociado.

Los sockets son interfaces que nos permiten crear una comunicación entre procesos o una conexión entre ellos. Fueron desarrollados por la Berkeley Software Distribution (BSD) de la Universidad de Berkeley en California, para la comunicación entre procesos en sistemas Unix. Esta interface de comunicación fue desarrollada en lenguaje de programación C.

Basado en lo anterior, se crearon los WinSockets, que funcionan para Microsoft Windows. Estos definen una interface de programación en red para ambiente Windows la cual cuenta con diferentes extensiones que permiten al desarrollador tomar ventaja del manejo de mensajes en Windows.

II.2.2 Modelo Cliente-Servidor

La tendencia en el desarrollo de aplicaciones nos dicta construir aplicaciones bajo el modelo cliente-servidor. En este esquema, las aplicaciones cliente realizan peticiones a las aplicaciones de servidor. Esto implica establecer una comunicación asimétrica entre el cliente y el servidor.

Tanto el cliente como el servidor requieren conocer las convenciones antes de que el servicio pueda ser proporcionado. Dependiendo de la situación, el protocolo puede ser simétrico o asimétrico. En el protocolo simétrico cualquiera de los dos puede ser servidor o cliente. En un protocolo asimétrico, se conoce claramente quien es el cliente y quien el servidor.

II.2.3 Funciones.

El diseño de la interface de sockets en Unix se hizo teniendo en cuenta la naturaleza evolutiva de la tecnología de redes, por lo que la portabilidad fue uno

de los objetivos del diseño principal. En forma interna, un proceso Unix maneja los sockets de la misma forma que a los identificadores de archivos: a través de un descriptor de archivo. La característica anterior es fundamental ya que permite la redirección de los archivos de entrada/salida estándar y con esto, la utilización de aplicaciones estándares sobre la red.

Los sockets normalmente intercambian datos con sockets que se encuentran en su mismo dominio, aunque se puede realizar con otros dominios, siempre y cuando se ejecute un proceso de translación. Las facilidades de los WinSockets soportan un dominio de comunicación simple el cual es utilizado por procesos los cuales se comunican utilizando el protocolo de Internet.

II.2.4 Tipos de Sockets.

De todos los tipos de sockets los más utilizados son los **DGRAM** y **STREAM,** pero presentamos una definición de los más importantes.

SOCK_DGRAM: Soportan un flujo bidireccional de datos en los cuales no es necesario realizar verificaciones sofisticadas en el envío y recepción de datos, es decir, que podemos tener duplicidad de la información. Esto es, que un proceso que recibe mensajes en un socket de este tipo, puede encontrar mensajes duplicados y posiblemente en un orden diferente al que fueron enviados. Las comunicaciones establecidas con esta clase de sockets cumplen con la propiedad de comunicación de conservación de los límites de los mensajes. En el dominio Internet, el protocolo subyacente utilizado es el protocolo UDP, el cual fue diseñado para aplicaciones en las que no es necesario poner secuencias de datagramas juntos y no hay que realizar verificaciones sofisticadas en el envío y recepción de los mismos.

SOCK_STREAM: Los sockets de este tipo permiten comunicaciones confiables en modo conectado, cumpliendo con las propiedades de fiabilidad en la transmisión, conservación del orden de los datos, no duplicación de los mismos y comunicación en modo conectado. Eventualmente permiten según el protocolo utilizado, los mensajes fuera de flujo normal, cumpliéndose así la propiedad de mensajes urgentes. El protocolo subyacente utilizado en el dominio Internet es el TCP, el cual es un protocolo de transporte confiable para volúmenes grandes de información.

SOCK_RAW: Este tipo de sockets permite el acceso a los protocolos de más bajo nivel por ejemplo el protocolo IP en el dominio Internet. Su utilización está reservada al superusuario y además permiten la implantación de nuevos protocolos.

SOCK_SEQPACKET: La comunicación utilizada en este tipo de sockets se encuentra en el dominio de XNS (Xerox Network System) y cumple con todas las propiedades de una comunicación excepto la de envío de mensajes urgentes.

II.3 Dominios

El dominio de un socket especifica el formato de direcciones que se podrán asignar al mismo, de igual modo, define los protocolos soportados por las comunicaciones que se realizarán a través de los sockets pertenecientes a cierto dominio. Un socket existe dentro de un dominio y sólo puede intercambiar datos con otro socket que pertenezca al mismo dominio. Los dominios que describiremos son: **Unix e Internet**. Los primeros son utilizados para la comunicación entre procesos dentro de un mismo sistema **Unix**, y los otros soportan la comunicación entre procesos en sistemas separados. Los Winsockets tienen su dominio en Internet, pero como se menciona anteriormente, pueden comunicarse con los BSD si un proceso de traslación corre entre ellos.

La estructura utilizada para la definición del dominio de un socket es:

```
struct sockaddr
    u_short     sa_family;     /* familia de dirección  */
     char       sa_data    ;   /* 14 octetos de dirección (máximo) */
```

II.3.1 Dominio UNIX

En este dominio los sockets son llamados BSD, que son utilizados para la comunicación de tipo local entre procesos locales, por lo que las direcciones

son también locales en el sistema en el que han sido definidos. La estructura de una dirección en este dominio está predefinida en la librería **sys/un.h**

```
struct sockaddr_un
        short sun_family;       /*dominio UNIX:AF_UNIX */  (AF=Address

Format)
        char  sun_data;         /* referencia de dirección */
        ;
```

II.3.2 Dominio INTERNET.

Este dominio se utiliza para el uso de las direcciones **INTERNET**, y la estructura que lo define es la **sockaddr_in**, que se utiliza para designar un servicio de red sobre cierta máquina en particular.

```
struct in_addr
        u_long              s_addr ;
struct sockaddr_in
        short       sin _family;      /*la familia de la dirección:AF_INET
*/
        u_short     sin_port          /* el número de puerto */

struct in_addr      sin_addr;         /* la dirección INTERNET */
        char        sin_zero  8 ;           /* un campo de ocho ceros */
```

Para la comunicación utilizando los protocolos Internet, en el primer campo tendrá el valor **AF_INET**; el segundo campo será un número de puerto, siendo recomendable utilizar la representación estándar. En el caso de los servicios estándares, se podrá utilizar un nombre simbólico para designar el puerto, es decir, puertos lógicos que ya están asignados por omisión.

El último campo sirve para hacer coincidir el tamaño de esta estructura con el de la de dirección genérica.

En este dominio podemos mencionar, por ejemplo **NetIPC** que lo utilizan las máquinas **HP**.

II.4 Creación y supresión.

II.4.1 Creación.

La creación de un socket se realiza utilizando la primitiva socket, cuyo valor será un descriptor sobre el cual se podrán realizar operaciones de lectura y escritura.

Al utilizar esta primitiva, se lleva a cabo la inicialización de entradas a las diferentes tablas del sistema de gestión de archivos: la tabla de descriptores de procesos, la tabla de archivos y estructuras de datos conteniendo las característica de los sockets de igual manera que los nodos para los archivos.

Entre las características de un socket se encuentran:

1) El tipo, el dominio y el protocolo.
2) El estado de socket (conectado o no, enlazado, en estado de recibir o enviar)
3) La dirección del socket conectado, en caso de que tenga alguna.
4) Apuntadores a los datos, tanto en emisión como en recepción.
5) Un grupo de procesos para la gestión de mecanismos asíncronos.

La forma general de la primitiva que permite crear un socket y obtener un descriptor para utilizarlo es la siguiente:

```
int socket(dominio,tipo,protocolo)
int dominio;                    /* AF_INET AF_UNIX ... */
int tipo;                       /*SOCK_DGRAM   SOCK_STREAM
*/
int protocolo;                  /* 0: protocolo por omisión */
```

El primer parámetro especifica el dominio al que pertenecerá el socket; el segundo es el tipo de socket, para elegir el servicio de transporte, el cual dependerá del dominio en que se cree el mismo, y el tercero que representa el

protocolo por lo general será igual a 0, con el cual el sistema elige un protocolo por omisión, asociado al tipo y dominio del socket creado.

II.4.2 Supresión.

La supresión se lleva a cabo realizando al menos una llamada a la primitiva **close**, la cual se realiza cuando no hay ningún proceso que este haciendo referencia a dicho socket.

El comando es: **int close(int sock)**

donde el parámetro sock es el descriptor asociado al socket obtenido con la primitiva socket.

II.5 Conexión, envío y recepción.

II.5.1 Conexión

Algunos protocolos tales como el TCP requieren que antes de usar un socket, éste se conecte a un punto final remoto; por ejemplo, un cliente del servicio de impresión debe conectarse al puerto 515 del nodo que presta dicho servicio. La primitiva utilizada para inicializar dicha conexión es la **connect**, y la sintaxis es:

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

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

Para protocolos como el UDP que prestan servicios sin conexión, no es necesario realizar la llamada **connect**; sin embargo, si se hace la llamada, sólo se almacenará localmente la dirección para poder utilizarla en otras funciones. Esta función, regresa un entero positivo si no hubo ningún error y -1 si lo hay.

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

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

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

Un servicio puede recibir varias peticiones de conexión simultáneamente, sin embargo, en caso de suceder esto, la función **accept** sólo puede procesarlas secuencialmente, por lo que deben ser almacenadas en algún lugar antes de ser atendidas.

int listen(int sock,int peticiones);

La creación de este tipo de conexión es hecha en tres pasos que son: **inicialización**, **requerimiento** y **establecimiento**, los cuales se describen a continuación:

Inicialización: El cliente crea un socket con la función **socket()**. El servidor también crea un socket con la función **socket()**. El servidor une una dirección (número de puerto) al socket utilizando la función **bind()**. El servidor crea una cola que ingresa requerimientos de conexión utilizando la función **listen()**.

Requerimiento. El cliente envía un requerimiento de conexión utilizando la función **connect()**. En el server el requerimiento esta en las llamadas a la cola.
Establecimiento. El servidor acepta el requerimiento de conexión utilizando la función **accept**. Un nuevo descriptor de socket es asignado para establecer la conexión y el intercambio de datos. Al inicializar el descriptor del socket es posible detener un nuevo requerimiento.

II.5.2 Envío

Una vez que un proceso ha creado un socket, puede hacer uso de varias llamadas al sistema para enviar datos a través de él. Existen diferentes formas para llevar a cabo lo anterior, que son:
la primitiva write.

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

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

Por otro lado,

la primitiva send.

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

realiza la misma función que write, pero recibe un parámetro extra llamado bandera que permite modificar la forma de transmitir los datos. El valor de este último parámetro puede ser **MSG_OOB** (para enviar datos fuera de la banda), **MSG_DONTROUTE** (para no utilizar las facilidades de ruteo).

La primitiva sendto.

int sendto (sock,msg,lo,opcion,p_dest,lgdest)

```
int sock;                  /* descriptor del socket de envío */
char *msg;                 /* dirección del mensaje a enviar */
int lg;                    /* longitud del mensaje */
int opción;                /* =0 para el tipo SOCK_DGRAM */
struct sockaddr *p_dest    /* puntero a la dirección del socket
                           destino */
int lgdets                 /* longitud de la dirección del socket
                           destino */
```

permite, además de especificar la dirección del destino del mensaje, el uso en sockets no conectados (aquellos que no han utilizado la función **connect**).

En **la primitiva sendto**, el valor de retorno es en caso de éxito, el número de caracteres enviados y -1 en caso de falla. Cabe recordar que los únicos errores que se detectarán serán los locales. Esto significa en particular que, si un mensaje se envía con destino a una máquina existente sobre un puerto no asociado a un socket, el mensaje se perderá y el emisor no será notificado de este error.

La primitiva sendmsg.

Cuando se requiere realizar una serie de envíos de mensajes sucesivos, es recomendable utilizar *la primitiva sendmsg*. Una de las ventajas importantes al utilizar dicha primitiva es que realiza el envío de varios mensajes con una sola llamada al sistema, lo cual mejora el rendimiento.

```
int sendmsg (sock,msg,opción)
        int sock;                   /* descriptor del socket d1e emisión */
        struct msghdr msg;          /* tabla de los envíos a efectuar */
        int opción;
```

La estructura **msghdr** está predefinida en la librería <sys/socket.h>

```
struct msghdr
    caddr_t     msg_name;       /* dirección opcional */
    int         msg_namelen;    /* tamaño de la dirección */
    structiovec*msg_iov;        /* tabla de mensajes */
    int         msg_iovlen;     /* num. de elementos en msg_iov*/
    caddr_t     msg_accrights;  /* no utilizada para los sockets */
    int         msg_accrightslen;       /* no utilizada para los sockets */
    ;
```

II.5.3 Recepción.

La interface sockets provee también algunas llamadas al sistema para recibir los datos enviados con las llamadas descritas anteriormente.

Los datos enviados con la primitiva **write** pueden ser recibidos con **read**.

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

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

La primitiva recv.

int recv (int sock,char*buffer,int n,int bandera);

recibe los datos transmitidos por la llamada **send**. El parámetro bandera puede tener los mismos valores que la llamada send.

La primitiva recvfrom.

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

que lee los datos escritos por **sendto**.

II.5.4 Enlace a una dirección.

Cuando un socket es creado, no tiene nombre. Hasta que un nombre es ligado al socket, los procesos no tienen forma de referenciarlo, y consecuentemente, ningún mensaje puede ser recibido en el. La comunicación de procesos es ligada por una asociación. En el dominio Internet, una asociación es compuesta de direcciones locales y remotas, y puertos locales y remotos, mientras en el dominio Unix, una asociación es 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). En la mayoría de los dominios, las asociaciones deben ser únicas. En el dominio Internet nunca deben ser duplicadas.

La llamada al sistema **bind()** permite a un proceso especificar la mitad de una asociación, mientras que las primitivas **connect()** y **accept()** son usadas para completar la asociación del socket.

int bind (sock,name,lg)

```
int  sock;                    /* descriptor de sockets */
struct sockaddr *name         /*nombre a ligar con el socket */
int lg;                            /* longitud de la dirección */
```

La primitiva **bind()** retorna un valor de 0 cuando se logra con éxito la asignación y cuando eso sucede, cualquier proceso puede hacer referencia al socket a través de la dirección que se la ha asociado. Por lo anterior, la primitiva **bind()**, es imposible si el proceso no pertenece al socket del sistema.
Para ilustrar como funciona los sockets al momento de recibir y al momento de enviar datos en forma secuencial se tiene el siguiente diagrama. en este se muestra un socket que esta continuamente recibiendo información del nodo conectado y otro enviando secuencialmente información. (Fig 7).

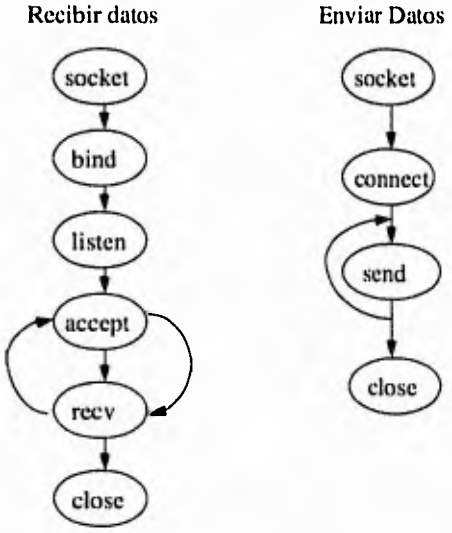


Fig.7 Sockets para envío y recepción.

II.6 WinSockets y BSD Sockets.

Winsockets cuenta actualmente con una serie de extensiones de los sockets de BSD estas se realizaron con el propósito de permitir a los desarrolladores en ambiente Windows, contar con la naturaleza del manejo de mensajes. De acuerdo a este punto anexamos un apéndice, en donde se observan las principales adecuaciones de los Winsockets.

Capítulo III. UNIX Esquema general

III.1 Introducción.

El Sistema Unix ha tenido una historia y una evolución fascinante. Iniciado como proyecto de investigación con un puñado de personas, se ha convertido en un producto importante utilizado ampliamente en el mundo de los negocios, el académico y el gubernamental. Unix sistema V versión 4 es un paso importante hacia la estandarización del mismo..

Este capítulo explicará el rápido crecimiento de Unix, así como su historia y las principales derivaciones del sistema.

III.1.1 La creciente importancia del sistema UNIX.

Durante los últimos 20 años el sistema operativo Unix se ha convertido en un sistema operativo potente, flexible y versátil. Sirve como sistema operativo para todo tipo de computadoras incluyendo las computadoras personales de monousuario, las estaciones de trabajo de ingeniería, microcomputadoras multiusuario, minicomputadoras, mainframes y supercomputadoras. El número de computadoras que corren bajo el sistema Unix ha crecido exponencialmente. El éxito se debe a muchos factores, entre ellos está la portabilidad a un gran abanico de máquinas, su adaptabilidad y simplicidad, el amplio rango de tareas que puede realizar, su naturaleza multiusuario y multitarea y su adecuación a las redes. A continuación mencionamos brevemente algunas características que han hecho popular al Sistema Unix.

- El código fuente del sistema Unix, y no el código ejecutable, ha estado disponible a usuarios y programadores. A causa de esto, mucha gente ha sido capaz de adaptarlo de diferentes formas. Este carácter abierto ha conducido a la introducción de un amplio rango de características nuevas y de versiones

personalizadas que se ajustan a necesidades especiales. A las personas que han desarrollado adaptaciones al sistema, les ha resultado fácil porque el código correspondiente es sencillo, modular y compacto. Esto ha fomentado la evolución del sistema, haciendo posible la fusión de las capacidades desarrolladas por diferentes variantes necesarias para soportar los entornos de computación de hoy en un sistema operativo único: el Unix sistema V versión 4.

- Proporciona a los usuarios diferentes herramientas y utilidades que se pueden utilizar para realizar una gran variedad de trabajos. Algunas de estas herramientas son órdenes que se pueden utilizar para llevar a cabo tareas específicas. Otras herramientas y utilidades son realmente pequeños lenguajes de programación que se pueden utilizar para construir guiones que resuelven sus propios problemas. Lo más importante es que las herramientas están diseñadas para funcionar juntas, como partes de una máquina o bloques de construcción.

- Puede ser utilizado por computadoras con muchos usuarios o con un único usuario. También es un sistema multitarea, ya que un usuario puede llevar a cabo más de una tarea al mismo tiempo.

- Proporciona un entorno excelente para redes. Ofrece programas y facilidades que proporcionan los servicios necesarios para construir aplicaciones basadas en red, utilizando el concepto de computación distribuida, en donde las computadoras comparten información y procesamiento con diferentes computadoras de la misma red. De este concepto continua creciendo la popularidad del sistema Unix.

- Es mucho más fácil de portar a nuevas máquinas que otros sistemas operativos. Esto es, se necesita menos trabajo para adaptarlo y correrlo sobre una máquina nueva. La portabilidad del sistema Unix es consecuencia directa de estar escrito casi completamente en un lenguaje "C". La portabilidad a un amplio rango de computadoras hace posible mover las aplicaciones de un sistema a otro.

Las descripciones anteriores nos muestra algunos de los atributos importantes del Sistema Unix que han motivado su crecimiento explosivo. Hoy lo ejecutan más de **100.000** máquinas, y el número de usuarios es del orden de millones. Cada vez más personas comienzan a utilizar el sistema Unix al darse cuenta que proporciona un entorno de computación que soporta sus necesidades.

III.1.2 El nacimiento de UNIX.

La historia del Sistema Unix data de los años 60's, cuando los laboratorios Bell de AT&T y el fabricante de computadoras GE (General Electric) trabajaron sobre un sistema operativo experimental denominado MULTICS, Información Multiplexada y Sistema de Computación (Multiplexed Information and Computing System). Fue diseñado como sistema operativo interactivo para la computadora GE 645, permitiendo la compartición de información al tiempo que proporcionaba seguridad. El desarrollo sufrió muchos retrasos, y las versiones de producción resultaron lentas y con grandes necesidades de memoria. Por una serie de razones, los laboratorios Bell abandonaron el proyecto. Sin embargo, el sistema MULTICS implementó muchas características innovadoras y produjo un entorno de computación excelente.

En 1969, Ken Thompson, uno de los investigadores de los laboratorios Bell involucrado en el proyecto MULTICS, escribió un juego para la computadora GE denominado Space Travel. Este juego simulaba el sistema solar y una nave espacial. Thompson vio que el juego se ejecutaba a tirones sobre la máquina GE y resultaba muy costoso -aproximadamente 75 Dlls., por ejecución -. Con la ayuda de Denis Ritchie, Thompson volvió a escribir el juego para ejecutarse sobre un DEC PDP-7. Esta experiencia inicial le dio la oportunidad de escribir un nuevo sistema operativo sobre el PDP-7, utilizando la estructura de un sistema de archivos que habían diseñado Thompson, Ritchie y Rudd Canaday. Thompson, Ritchie y sus colegas crearon un sistema operativo multitarea, incluyendo un sistema de archivos, un intérprete de órdenes y algunas utilidades para el PDP-7. Más tarde, una vez que el nuevo sistema operativo se estaba ejecutando, se revisó el Space Travel para ejecutarlo sobre él. Muchas cosas en el sistema Unix proceden de este simple sistema operativo.

Puesto que el nuevo sistema operativo multitarea para el PDP-7 podía soportar dos usuarios simultáneamente, se le llamó humorísticamente UNICS de Uniplexed Information and Computing System y el uso de este nombre se atribuye a Brian Kernighan. El nombre se cambió ligeramente a Unix en 1970, y ha permanecido así desde entonces. El grupo de investigación en informática (Computer Science Research Group) quería seguir utilizando el sistema Unix, pero sobre una máquina mayor. Ken Thompson y Denis Ritchie gestionaron la

obtención de un DEC PDP-11/20 en contrapartida a la promesa de añadir capacidades de procesamiento de texto al sistema Unix. Esto condujo a un grado modesto de soporte financiero de los laboratorios Bell para el desarrollo del proyecto del sistema Unix. El sistema operativo Unix, con el programa de formateado de texto runoff, ambos escritos en lenguaje ensamblador, fueron portados al PDP-11/20 en 1970. Este sistema de procesamiento de texto inicial consiste en el sistema operativo Unix, un editor y runoff, los cuales fueron adoptados por el departamento de patentes de los laboratorios Bell como procesador de texto; runoff evolucionó a troff, el primer programa de edición electrónica con capacidad de composición tipográfica.

En 1972, la segunda edición del Manual del programador Unix mencionaba que había exactamente diez computadoras utilizando el sistema Unix, pero indicaba que se estaban esperando más. En 1973, Ritchie y Thompson volvieron a escribir el núcleo en el lenguaje de programación "C", un lenguaje que a diferencia de la mayor parte de los sistemas escritos para máquinas pequeñas que utilizaban generalmente un lenguaje ensamblador. La escritura del sistema operativo Unix en "C" hacía mucho más fácil su mantenimiento y portabilidad a otras máquinas. La popularidad del sistema Unix creció debido a sus innovaciones y debido a que estaba escrito compactamente en lenguaje que podía modificarse en código, de acuerdo a las preferencias individuales. AT&T no ofreció comercialmente el sistema Unix porque en ese tiempo no estaba en el negocio de las computadoras. Sin embargo AT&T permitió la disponibilidad del sistema Unix a universidades, firmas comerciales y al gobierno por un costo simbólico.

Hacia 1974 comenzó a utilizarse ampliamente en los laboratorios Bell la cuarta edición del Sistema Unix. (las versiones del sistema Unix producidas por los grupos de investigación de los laboratorios Bell se han conocido tradicionalmente como ediciones.)

En 1977 salió la quinta y sexta edición: éstas contenían bastantes herramientas y utilerías. El número de máquinas que podrían ejecutar el sistema, (fundamentalmente en los laboratorios Bell y en las Universidades), se incrementó en más de 600 en 1978. La séptima edición, el ascendente directo del sistema operativo Unix disponible hoy día, salió en 1979.

El sistema III de Unix, basado sobre le edición séptima, se convirtió en 1982 en la primera versión comercial de AT&T, sin embargo, una vez lanzado el

sistema III, AT&T a través de su subsidiaria Western Electric continuó vendiendo versiones. El sistema III de Unix, las diferentes ediciones de investigación y las versiones experimentales se distribuyeron a los colegios universitarios y otros laboratorios de investigación. Con frecuencia resultaba imposible para los informáticos saber si una característica particular formaba parte de la estructura principal del sistema Unix o era sólo parte de una de sus variantes.

III.1.3 El sistema V de UNIX

Para eliminar esta confusión sobre las variedades del sistema Unix, AT&T introdujo el Unix sistema V versión 1 en 1983, y posteriormente la versión 2 en 1985.

En 1987, AT&T introdujo el Unix sistema V versión 3.0; ésta incluía un enfoque de redes simple y consistente. Estas capacidades incluyen: a) STREAMS, utilizados para construir software de redes, b) el sistema de archivos remoto, utilizado para compartir archivos a través de redes, y c) la interface a nivel de transporte (TLI), utilizada para construir aplicaciones que utilizan redes. La versión 3.1 hizo al sistema V de Unix adaptable internacionalmente, soportando conjuntos de caracteres y formatos de hora y fecha más amplios. También proporciona los diferentes aumentos de rendimiento en el uso de la memoria y en las copias de seguridad y la recuperación de archivos. La versión 3.2 proporcionó seguridad ampliada del sistema, incluyendo la visualización de la hora de presentación última del usuario, el registro de los intentos de inicialización sin éxito y un archivo oculto de contraseñas que impide a los usuarios la lectura de contraseñas cifradas. La versión 3.2 también introdujo el FACE (Framed Acces Command Environment) que proporciona una interface de usuario orientada a menú.

La versión 4 unifica varias versiones del sistema Unix que han sido desarrolladas dentro y fuera de AT&T. Antes de descubrir el contenido de esta nueva versión, presentaremos las versiones del sistema Unix que aparecieron en Berkeley, en Sun Microsystems y Xenix, un sistema Unix desarrollado para microcomputadoras.

La Distribución de software de Berkeley (BSD)

Muchas innovaciones importantes del sistema Unix se han hecho en la Universidad de California en Berkeley. Algunas de estas ampliaciones han formado parte del Sistema V de Unix en las primeras versiones, y muchas más se han introducido en la versión 4.

La U.C. Berkeley se involucró en el sistema Unix en 1974, comenzando con la cuarta edición. El desarrollo de la versión de Berkeley del sistema Unix fue promovida por Ken Thompson en 1975 en el departamento de informática. En Berkeley, Ken transportó la sexta edición a un PDP-11/70 haciendo que el sistema Unix estuviese disponible para un gran número de usuarios. Los estudiantes graduados Bill Joy y Chuck Haley hicieron buena parte del trabajo de la versión Berkeley. Ellos diseñaron un editor denominado **ex** y produjeron un compilador de Pascal. Joy diseño un paquete que llamó el "Berkeley Software Distribution". También hizo muchas otras innovaciones valiosas, incluyendo el "C" shell y el editor orientado a pantalla **Vi** -una ampliación de ex-. En 1978 se hizo la segunda distribución de Software de Berkeley; a la que se conoció como 2BSD. En 1979 se distribuyó 3BSD, basada en la 2BSD y la séptima edición, proporcionando memoria virtual para permitir que pudiesen ejecutarse grandes programas. 3BSD se desarrolló para correr en el DEC VAX-11/780.

A finales de los 70´s, el departamento DARPA (Defense´s Advanced Research Projects Agency) de los Estados Unidos decidió basar su entorno de computación universal en el sistema Unix. Decidió que el desarrollo de su versión se llevaría a cabo en Berkeley, consolidando la 4BSD. En 1983 apareció la 4.1BSD, que disponía de mayor rendimiento. La 4.2BSD también apareció en 1983, introduciendo características de red, incluyendo redes TCP/IP, que pueden ser utilizadas para transferir archivos y presentaciones remotas, y un nuevo sistema de archivos que acelera el acceso a los mismos. La versión 4.3BSD vino en 1987, con pequeños cambios sobre la anterior.

Muchos vendedores de computadoras han utilizado el sistema BSD como base para el desarrollo de sus variantes. Uno de los cambios más importantes es el sistema operativo de Sun (SunOS), desarrollado por Sun Microsystems, una compañía fundada por Joy. SunOS añadió muchas características a la 4.2BSD, incluyendo el sistema de archivos de red (NFS). SunOS es uno de los componentes que se ha fundido en el Unix sistema V versión 4.

Muchas innovaciones importantes del sistema Unix se han hecho en la Universidad de California en Berkeley. Algunas de estas ampliaciones han formado parte del Sistema V de Unix en las primeras versiones, y muchas más se han introducido en la versión 4.

La U.C. Berkeley se involucró en el sistema Unix en 1974, comenzando con la cuarta edición. El desarrollo de la versión de Berkeley del sistema Unix fue promovida por Ken Thompson en 1975 en el departamento de informática. En Berkeley, Ken transportó la sexta edición a un PDP-11/70 haciendo que el sistema Unix estuviese disponible para un gran número de usuarios. Los estudiantes graduados Bill Joy y Chuck Haley hicieron buena parte del trabajo de la versión Berkeley. Ellos diseñaron un editor denominado **ex** y produjeron un compilador de Pascal. Joy diseño un paquete que llamó el "Berkeley Software Distribution". También hizo muchas otras innovaciones valiosas, incluyendo el "C" shell y el editor orientado a pantalla **Vi** -una ampliación de ex-. En 1978 se hizo la segunda distribución de Software de Berkeley; a la que se conoció como 2BSD. En 1979 se distribuyó 3BSD, basada en la 2BSD y la séptima edición, proporcionando memoria virtual para permitir que pudiesen ejecutarse grandes programas. 3BSD se desarrolló para correr en el DEC VAX-11/780.

A finales de los 70's, el departamento DARPA (Defense's Advanced Research Projects Agency) de los Estados Unidos decidió basar su entorno de computación universal en el sistema Unix. Decidió que el desarrollo de su versión se llevaría a cabo en Berkeley, consolidando la 4BSD. En 1983 apareció la 4.1BSD, que disponía de mayor rendimiento. La 4.2BSD también apareció en 1983, introduciendo características de red, incluyendo redes TCP/IP, que pueden ser utilizadas para transferir archivos y presentaciones remotas, y un nuevo sistema de archivos que acelera el acceso a los mismos. La versión 4.3BSD vino en 1987, con pequeños cambios sobre la anterior.

Muchos vendedores de computadoras han utilizado el sistema BSD como base para el desarrollo de sus variantes. Uno de los cambios más importantes es el sistema operativo de Sun (SunOS), desarrollado por Sun Microsystems, una compañía fundada por Joy. SunOS añadió muchas características a la 4.2BSD, incluyendo el sistema de archivos de red (NFS). SunOS es uno de los componentes que se ha fundido en el Unix sistema V versión 4.

III.1.4 El sistema XENIX

En 1980, Microsoft introdujo el sistema Xenix, una variante del sistema Unix, diseñado para ejecutarse sobre microcomputadoras. La introducción del sistema Xenix llevó las capacidades del sistema Unix a máquinas de escritorio, capacidades que anteriormente sólo estaban disponibles en grandes computadoras. El sistema Xenix se basó originalmente en la séptima edición, con algunas utilidades prestadas de la 4.1BSD. En la versión 3.0 del sistema Xenix, Microsoft incorporó nuevas características del sistema III del Unix de AT&T y en 1985 el sistema Xenix se desplazó hacia uno basado en el sistema V de Unix.

Xenix ha sido portado a diferentes microprocesadores, entre los que se encuentran la familia Intel 8086 y 80386 y la familia Motorola 68000. En particular en 1987 Xenix fue portado por la Operación Santa Cruz a las máquinas basadas en el 80386, una compañía que ha trabajado con Microsoft sobre el desarrollo de Xenix. En 1987, Microsoft y AT&T comenzaron a desarrollar juntos la fusión de Xenix y el sistema V de Unix. Esto se ha conseguido en el Unix sistema V versión 3.2. Como resultado, Xenix ya no es un sistema separado. Este esfuerzo logró una versión unificada del sistema Unix que puede correr sobre sistemas que van desde las computadoras personales de escritorio hasta las supercomputadoras.

Estándares. El uso de diferentes versiones del sistema Unix ocasionaba problemas a las personas encargadas de desarrollar aplicaciones para un amplio rango de computadoras que soportaban este sistema. Para resolver este problema se han desarrollado varios estándares. Estos estándares definen las características que deberá tener un sistema para que puedan construirse aplicaciones que funcionen sobre cualquier sistema que se adecúa al estándar. Uno de los objetivos de la versión 4 es unificar las variantes importantes del sistema Unix en un único producto estándar.

Para que el sistema V de Unix se convierta en un estándar de la industria, otros vendedores necesitan probar sus versiones para acomodarlas a la funcionalidad del sistema V. En 1983 AT&T publicó la Definición de interface del sistema V (SVID). La SVID especifica como debe comportarse un sistema operativo para cumplir con el estándar. Las personas que desarrollan aplicaciones pueden construir programas con garantía de funcionar sobre cualquier máquina que

opere con una versión del sistema Unix conforme a SVID. Además, la SVID especifica características del sistema Unix de las que existen garantías de no cambiar en futuras versiones, de manera que existe la garantía de que las aplicaciones puedan ejecutarse sobre todas las versiones del sistema V de Unix. Los vendedores pueden comprobar si sus versiones del sistema Unix se ajustan a SVID ejecutando el System V Verification Suite desarrollado por AT&T. El SVID ha evolucionado con las nuevas versiones del sistema V de Unix. Se ha preparado una nueva versión del SVID en conjunción con el Unix system V versión 1. Naturalmente, el Unix system V versión 4 se ajusta a SVID.

Un esfuerzo independiente para definir un estándar de entorno de sistema operativo comenzó en 1981 a iniciativa de /usr/group, una organización constituida por usuarios del sistema Unix que querían asegurar la portabilidad de las aplicaciones (se publicó un estándar en 1984). Debido a la magnitud del trabajo, en 1985 el comité de trabajo sobre estándares se fusionó con el IEEE (P1003). El objetivo de P1003 fue establecer un conjunto de estándares ANSI (American National Standards Institute). Los estándares que los diferentes grupos de trabajo del P1003 están estableciendo se conocen como POSIX (Portable Operating System Interface for Computer Environments). POSIX es una familia de estándares que define cualquier aplicación que interactua con un Sistema Operativo. Entre las áreas cubiertas por los estándares POSIX están las llamadas al sistema, bibliotecas, herramientas, interfaces, verificación y prueba, características en tiempo real y seguridad. El estándar POSIX que ha recibido mayor atención es el P1003.1, que define la interface del sistema.

III.1.5 UNIX Sistema V versión 4.

El Unix sistema V versión 4 fusiona los esfuerzos de diferentes iniciativas en un único sistema operativo potente y flexible que recoge las necesidades descritas por diferentes comités de estándares y organizaciones vendedoras.

Los cambios de la versión 4 involucran muchos aspectos del sistema Unix, incluyendo el núcleo, las órdenes y aplicaciones de archivos y los shells.

El Unix sistema V versión 4 unifica en un único paquete las versiones más populares del sistema Unix, el sistema V versión 3, y los sistemas Xenix, BSD y

SunOS. La versión 4 se ajusta a los estándares importantes definidos para el sistema Unix por diferentes organizaciones industriales y gubernamentales.
Para unificar las variantes del sistema Unix y adecuarlas a los estándares fue necesario redefinir partes de su arquitectura. Por ejemplo, el sistema de archivos tradicional se ha extendido para soportar sistemas de archivos de diferentes clases. Estos cambios se han hecho de forma que se garantice un alto grado de compatibilidad con las versiones anteriores del sistema V de Unix.
La siguiente figura (Fig. 8) muestra las relaciones de las diferentes versiones del sistema Unix con el Sistema V versión 4.

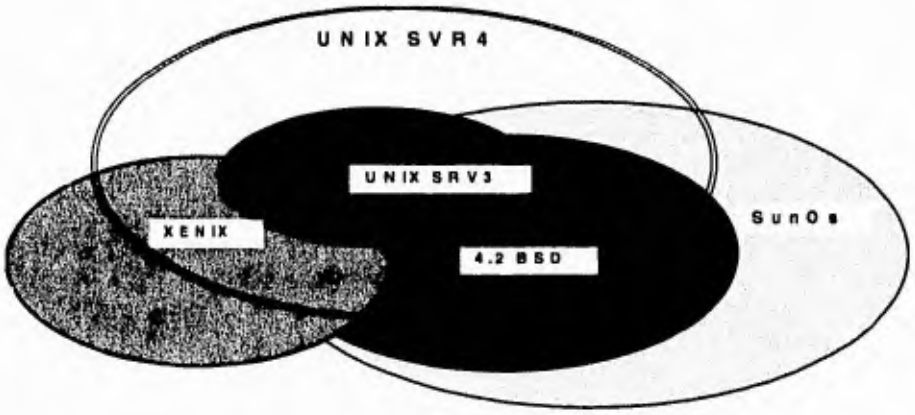


Fig 8. Diferentes versiones UNIX.

III.2 Sistema Operativo UNIX

El sistema operativo Unix puede ser descompuesto en tres componentes básicos: El scheduler, el file system y el shell.

III.2.1 EL Scheduler UNIX.

El scheduler Unix es un programa que permite a más de una gente usar una computadora al mismo tiempo. El scheduler administra los recursos de la computadora a través de esos usuarios, permitiendo a cada uno una pequeña

pieza del procesador de la computadora. Este concepto es conocido como "**time-sharing**".
Por ejemplo, supongamos tres personas que quieren correr diferentes programas **a, b** y **c** (Fig.9). El scheduler copia estos tres programas desde el disco que almacena los programas dentro de la memoria de la computadora. En el sistema Unix, estas copias en memoria son llamadas procesos, en esta forma hacemos distinción entre programas que son guardados como archivos en un disco y un proceso que está en memoria haciendo cosas.

El scheduler permite al proceso **a** correr por unos varias centésimas de segundo y hacer un poco del trabajo para el cual fue diseñado. Después de que esta pequeña pieza de tiempo ha terminado, el proceso **a** es temporalmente parado, o suspendido, y se permite entonces correr al proceso **b**. Después, cuando el tiempo del proceso b termina, el proceso c tiene la oportunidad de correr.

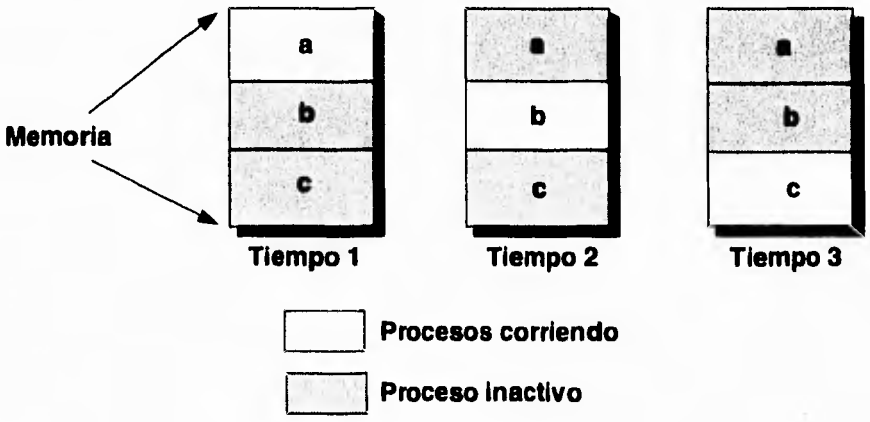


Fig 9. Multiprocesamiento.

Cuando cada proceso ha tenido la oportunidad de correr, el scheduler regresa al proceso **a**. Este no empieza el proceso **a** desde el principio, sino que lo empieza donde lo dejo cuando este proceso fue suspendido. Así, el scheduler permite a cada proceso trabajar de esta forma hasta su terminación.

La mayoría de los sistemas "**time-sharing**" permiten que mucho más de tres procesos puedan correr al mismo tiempo, en efecto el scheduler Unix puede

trabajar con varios cientos de procesos. Este también permite a cada usuario correr en forma efectiva más de un proceso a la vez.

Debido a la gran velocidad de las computadoras, el efecto deseado del time-sharing es dar a los usuarios la impresión de que ellos están siendo servidos simultáneamente, a pesar de que el scheduler actualmente atiende los procesos uno a la vez.

Swapping and Paging

Este simple modelo de scheduler trabaja bien al inicio, pero tarde o temprano, toda la memoria de la computadora se llenará con los procesos corriendo. En este punto, si un usuario quiere correr un nuevo programa, el scheduler debe encontrar una manera de meterlo en la memoria.

Esto trajo el concepto de **swapping.** Cuando la memoria está llena y otro proceso necesita correr, el scheduler toma un proceso en memoria y lo copia a disco. El scheduler entonces pone al nuevo programa en memoria (creando un nuevo proceso) y le permite correr. Después, el proceso en disco se regresa a la memoria y otro proceso en memoria se pasa a disco para seguir procesando.

Por ejemplo, digamos que hay tres procesos **a**,**b** y **c**. Ahora una petición es hecha por un usuario para correr un programa **d**: (Fig. 10)

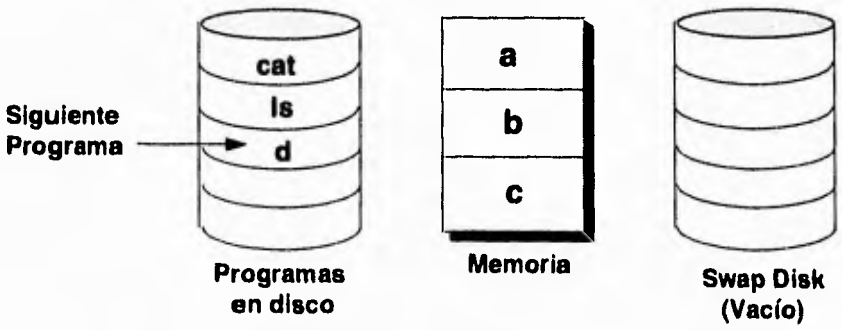


Fig. 10 Inicio de Swapping.

ya que no hay espacio en la memoria para el proceso **d**, el proceso **a** es copiado a disco. (Fig 11)

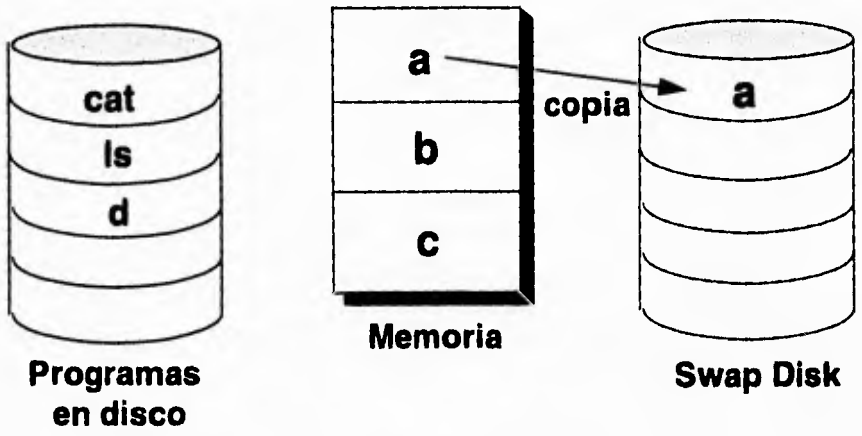
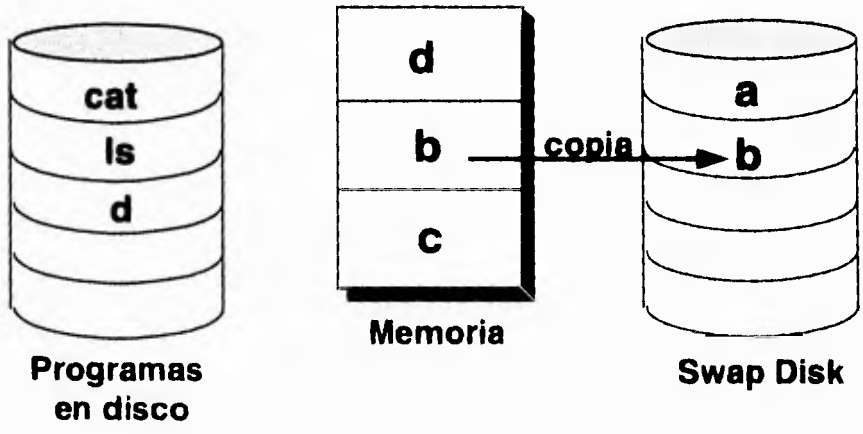


Fig. 11Swapping a Disco.

Después de que **a** es copiado a disco, una copia de **d** es puesta en memoria donde **a** estaba.

Después de unos pequeños lapsos de tiempo, el scheduler regresará el proceso **a** a memoria, usualmente intercambiando lugar con el proceso que ha estado más tiempo en memoria. Asumamos que **a** ha sido cambiado con **b**. Primero, **b** debe ser copiado a disco. (Fig. 12).



Fig, 12 Procesos en Swapping.

Este método de copiado de procesos desde disco y memoria continua mientras que no haya suficiente espacio para todos los procesos en memoria al mismo tiempo. Por supuesto, este ejemplo es muy simple comparado con la actual operación del scheduler Unix. Este debe manejar cientos de procesos de todos los diferentes tamaños, de los cuales solo treinta o cuarenta podrían estar dentro de la memoria al mismo tiempo.

Hoy en día, la mayoría de los sistemas Unix emplean un mecanismo llamado **paging** en lugar de **swapping. Paging** es muy similar al **swapping**, excepto que sólo partes de un programa (páginas) son copiadas a y desde disco. Las páginas son siempre fijas en tamaño, usualmente de 2048 ó 4096 bytes. Cuando la memoria es necesitada, una pagina es copiada a disco, dejando libre una porción de memoria. Después cuando la página es necesitada por el programa en uso, ésta es copiada de regreso a la memoria, causando posiblemente que otra página diferente sea copiada a disco.

A pesar de ser mas compleja que el **swapping**, **paging** es más eficiente, debido a que sólo piezas de un programa necesitan ser movidas cuando la memoria es requerida.

III.2.2 SHELL.

Una gran parte del uso del sistema Unix consiste en emitir órdenes, cuando usted emite una orden está relacionándose con el Shell, la parte del sistema Unix a través de la cual se controlan los recursos del sistema operativo. El **Shell** proporciona muchas de las características que hacen al sistema Unix un entorno potente y flexible. Se trata de un interprete de órdenes, un lenguaje de programación, y más. Como interprete de órdenes el Shell lee las órdenes que se le introducen y dispone lo necesario para que se ejecuten. Además se puede utilizar el lenguaje de órdenes del Shell como un lenguaje de programación del alto nivel para crear programas denominados ***"Guiones"***.

Shell de presentación. Cuando usted se presenta al sistema, se inicia automáticamente un programa de ***shell.*** Este es el shell de presentación. El programa de shell particular que se ejecuta cuando usted se presenta está

determinado por su entrada en el archivo */etc/passwd*. Este archivo contiene la información que el sistema necesita conocer sobre cada usuario, incluyendo el nombre, el ID de presentación, y demás.

Funciones del Shell. Una vez que usted se presenta, muchas de las iteraciones con el sistema Unix tienen la forma de diálogo con el shell. Este diálogo sigue una secuencia simple repetida una y otra vez.

- El shell le solicita una orden cuando está dispuesto para recibir una entrada y espera que usted la introduzca.
- Usted 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 programa el shell le solicita entrada, comenzando de nuevo el ciclo.

La parte de este ciclo que realiza el trabajo real es el tercer paso -cuando el shell lee y procesa la línea de orden y ejecuta las instrucciones que contiene- por ejemplo, sustituye las palabras de la línea de orden que contienen comodines por los nombres de archivos correspondientes. Determina de dónde procede la entrada de la orden y a donde va la salida. Después de realizar estas y similares operaciones, el shell ejecuta el programa indicado en la orden, proporcionándoles los adecuados argumentos -por ejemplo, opciones y nombres de archivos.

Variables del Shell. El shell dispone de un mecanismo para definir variables que se pueden utilizar para contener información utilizada por los programas del sistema o para uso propio. Algunas variables las utiliza el propio shell en otros programas del sistema Unix. Se puede definir otras para uso propio. Las variables del shell se pueden utilizar para personalizar o particularizar la información relativa a nombres de directorios y de archivos que necesitan los programas y para personalizar la forma en la que los programas (incluyendo el propio shell) interactuan con el usuario.
Algunas de las variables del shell incluyendo aquellas que fija automáticamente el sistema, se muestran a continuación:

Variable de Shell	Descripción	Ejemplo
CDPATH	Lista de directorios buscados por la orden **cd**	/home/you /Book
HOME	Nombre de camino de su directorio de presentación.	/home/you
LOGE NAME	Su nombre de presentación.	you
MAIL	Nombre de camino del directorio que contiene su correo.	/home/you /Mail

III.2.3 FILE SYSTEM

Un pilar básico del Sistema Unix es el sistema de archivos jerárquico. El sistema de archivos proporciona una forma potente y flexible de organizar y gestionar nuestra información contenida en la computadora. Aunque muchas de las características del sistema de archivos se inventaron originalmente para el sistema Unix, su estructura ha resultado ser tan útil que se ha adoptado por muchos otros sistemas operativos.

Este capítulo proporciona una introducción al sistema de archivos Unix, en donde se mostrarán las características del sistema de archivos, los tipos de estos, y la estructura jerárquica en general.

Tipos de archivos del sistema UNIX.
El archivo es la unidad básica del sistema Unix. Dentro del sistema operativo Unix hay cuatro tipos diferentes de archivos: a) archivos ordinarios, b)vínculos, c) vínculos simbólicos, d) archivos especiales, y e) directorios. Además los archivos pueden tener más de un nombre, conocidos como vínculos.

a) Archivos ordinarios.

Como usuario, la información con la que trabaja será almacenada como archivo ordinario. Los archivos ordinarios son agregados de caracteres tratados como una unidad por el sistema Unix. Un archivo ordinario puede contener caracteres ASCII normales tales como texto de manuscritos o programas. Los archivos ordinarios pueden crearse, cambiarse, o borrarse cuando se desee.

b) Vínculos.

Un vínculo no es una clase de archivo, sino un segundo nombre para un archivo. Si dos usuarios necesitan compartir la información de un archivo, ellos pueden tener copias separadas de este archivo. Un problema al mantener copias separadas es que las dos copias pueden rápidamente perder la consistencia. Por ejemplo, un usuario puede realizar cambios que el otro podría no conocer. Un vínculo proporciona la solución a este problema. Con un vínculo, dos usuarios pueden compartir un único archivo. Ambos usuarios parecen tener copias del archivo, pero solamente existe un archivo con dos nombres. Los cambios que cualquier usuario realiza tienen lugar sobre la versión común. Este vínculo no solamente ahorra espacio al tener una única copia de un archivo, sino que asegura que la copia que cada uno utiliza es la misma.

c) Vínculos simbólicos.

Los vínculos se pueden utilizar para asignar más de un nombre a un archivo. Pero tienen algunas limitaciones importantes. No se pueden utilizar para asignar a un directorio más de un nombre. Y no se pueden utilizar para vincular nombres de archivos sobre computadoras diferentes. Esto es un fallo importante de los vínculos, ya que la versión 4 proporciona dos sistemas de archivos distribuidos, NFS y RFS, (de los cuales se hablará posteriormente) para hacer posible la compartición de archivos entre computadoras.

Estas limitaciones pueden eliminarse utilizando vínculos simbólicos, introducidos en el sistema Unix sistema V versión 4 procedente de BSD. Un vínculo simbólico es un archivo que solo contiene el nombre de otro archivo. Cuando el sistema operativo opera sobre un vínculo simbólico, éste se dirige al archivo al que apunta el vínculo simbólico. Los vínculos simbólicos no sólo se pueden utilizar para asignar más de un nombre a un archivo, sino que pueden usarse para asignar más de un nombre a un directorio. Los vínculos simbólicos también pueden ser utilizados por vínculos que residen en sistemas de archivos físicos diferentes. Esto hace posible que un árbol de directorio lógico incluya archivos que residen sobre computadoras diferentes que están conectadas a

través de una red. Los vínculos también se denominan vínculos duros (hard link) para distinguirlos de los vínculos simbólicos.

d) Directorios.
Un directorio es un archivo que mantiene otros archivos y contiene información sobre las localizaciones y atributos de éstos. Por ejemplo, un directorio incluye una lista de todos los archivos y subdirectorios que éste contiene, así como sus direcciones, características, tipos de archivos (si son archivos ordinarios, vínculos simbólicos, directorios, o archivos especiales), y otros atributos.

e) Archivos especiales.
Los archivos especiales constituyen una característica no usual del sistema de archivos Unix. Un archivo especial representa un dispositivo físico. Puede ser una terminal, un dispositivo de comunicaciones, o una unidad de almacenamiento como un disco. Desde la perspectiva del usuario, el sistema Unix trata los archivos especiales como archivos ordinarios; esto es, puede leer o escribir los dispositivos exactamente como lee y escribe los archivos ordinarios. Se pueden tomar los caracteres pulsados en el teclado y escribirlos de la misma forma en un archivo ordinario o una pantalla. El sistema Unix toma estas órdenes de lectura y escritura y produce la activación del hardware conectado al dispositivo.
Esta forma de tratar el hardware del sistema tiene una consecuencia importante para los usuarios del sistema Unix. Puesto que el Unix trata casi todo como si fuese un archivo, no se necesita aprender las particularidades del hardware de la computadora. Una vez que se aprende a manejar los archivos del sistema Unix, se sabe cómo manejar todos los objetos del sistema Unix. Se utilizará la misma orden (**Ls**) para ver si puede leer o escribir en un archivo, una terminal, o un disco.

La estructura jerárquica de los archivos.

Debido a que los directorios pueden contener otros directorios, que a su vez pueden contener otros directorios, el sistema de archivos del Unix se denomina **sistema de archivos jerárquicos**. De hecho, dentro del sistema Unix, no existe limitación del número de archivos y directorios que se pueden crear en un directorio propio. Los sistemas de archivos de este tipo se conocen como sistemas de archivos con estructura en árbol, porque cada directorio se permite ir o saltar hacia otros directorios y archivos. Los sistemas de archivos con

estructura de árbol se dibujan normalmente de arriba abajo, con la raíz del árbol en la parte superior del dibujo.

La figura siguiente (Fig. 13) muestra las conexiones entre los archivos y directorios tratados en los ejemplos.

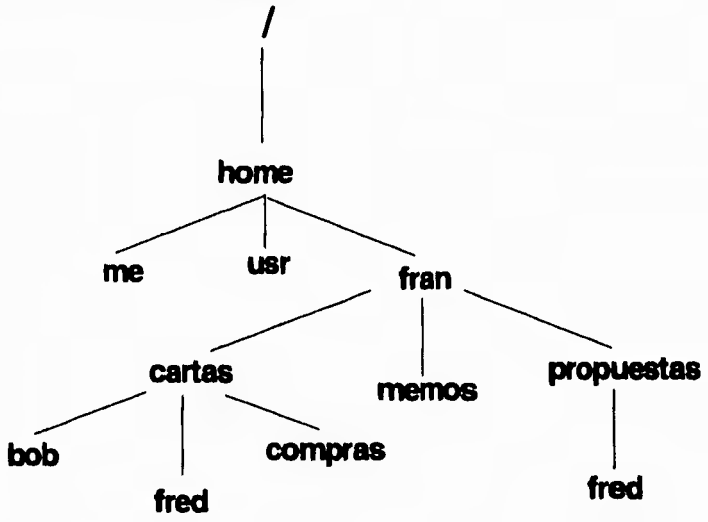


Fig. 13 Estructura de Directorios.

La raíz del directorio completo está en la parte superior del dibujo. Se denomina directorio raíz, o solamente raíz. La raíz se representa con un / (slash).

En la figura la raíz contiene un subdirectorio home. Dentro del directorio home se tiene un nombre de usuario que tiene un subdirectorio asociado (fran); en este directorio hay tres subdirectorios (cartas, memos, propuestas): y en aquellos directorios hay otros subdirectorios o archivos (bob, fred, compras, fred).

El directorio en el que se encuentra usted colocado cuando realiza la presentación se denomina su directorio de trabajo (home). Cada usuario del sistema Unix tiene un único directorio de trabajo. En cada sesión se comienza en su directorio de trabajo y se mueve hacia arriba y hacia abajo en el árbol de directorios.

El árbol de directorios.

Uno de los cambios de la versión 4 es la reorganización del árbol directorio. La disposición se ha cambiado para acomodar la compartición de archivos entre diferentes computadoras por medio de un sistema de archivos distribuido, tal como RFS o NFS.

El sistema Unix le permite crear un número arbitrario de subdirectorios y llamarlos de la forma que quiera. Sin embargo, a menos que se sigan reglas o convenciones, el sistema de archivos se convertirá rápidamente en algo difícil de utilizar. Se ha utilizado un conjunto de reglas informales con las primeras versiones del sistema V de Unix. Estas convenciones describen que directorios deberían contener archivos con tipos particulares de información y cuáles deberían ser los nombres de los archivos. Por ejemplo, en Unix sistema V versión 3, el directorio /usr contenía todos los directorios de presentación de los usuarios (home) y /bin contenía ciertos programas ejecutables importantes, tales como /bin/mail y /bin/sh. La versión 4 altera estas convenciones. Para muchos usuarios del sistema Unix, estos cambios son uno de los aspectos más notables de esta versión.

Una versión parcial de un sistema de archivos típico sobre una computadora con Unix sistema V versión 4 se muestra en la siguiente figura. Este ejemplo incluye las partes del árbol directorio de la versión 4 de interés común. (fig. 14)

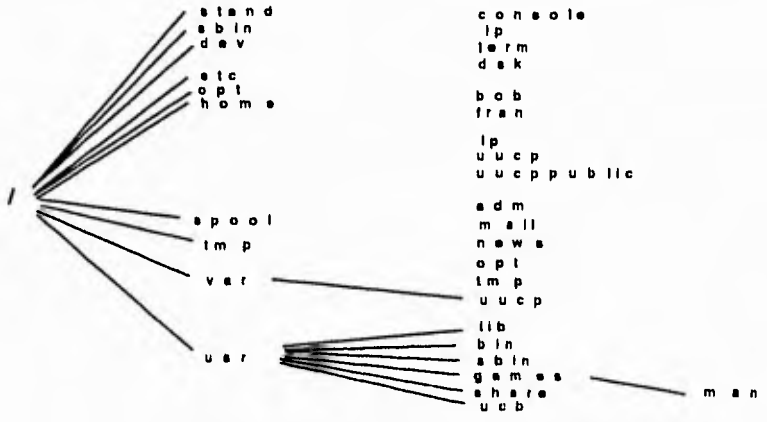


Fig. 14 Arbol de Directorio de Sistema V versión 4.

A continuación se describe el uso o contenido de cada uno de los directorios y subdirectorios arriba mostrados.

/ Este es el directorio raíz del sistema de archivos, el directorio principal del sistema completo de archivos y el directorio de trabajo (home) para el super usuario.

/stand Este contiene los programas estándar y los archivos de datos que se utilizan cuando se instala por primera vez el sistema Unix, o se inicializa.

/sbin Este contiene los programas utilizados en la inicialización del sistema y en la recuperación del mismo.

/dev Este contiene los archivos especiales (dispositivos) que incluyen a las terminales, impresoras y dispositivos al sistema operativo, incluyendo:

/dev/console,	la consola del sistema
/dev/lp,	la impresora de línea
/dev/term/*,	terminales de usuario.
/dev/dsk/*,	discos del sistema.

/etc Este contiene la administración del sistema y las bases de datos de configuración.

/opt Este es la raíz del subárbol que contiene los paquetes de aplicación específicos.

/home Este contiene los directorios y archivos de trabajo de todos los usuarios. Si su nombre de usuario es fran, su directorio de trabajo por defecto es /home/fran.

/spool Este contiene los directorios de los archivos temporales de spooling. El spooling consiste en salvar copias de los archivos para un procesamiento posterior. Los archivos temporales sometidos a spooling se eliminan una vez que se han utilizado. Los archivos en este directorio incluyen:

/spool/lp Es un directorio para efectuar spooling sobre archivos para impresoras.

/spool/uucp Es un directorio para poner en la cola los trabajos del sistema UUCP.

/spool/uuc/ppublic Es un directorio que contiene archivos depositados por el sistema UUCP.

/tmp Este contiene todos los archivos temporales utilizados por el sistema UNIX.

/var Este contiene los directorios de todos los archivos que varían entre sistemas. Este incluye archivos que registran la actividad del sistema, archivos de contabilidad, archivos de correo, paquetes de aplicación, archivos para seguridad de editores, y muchos otros tipos de archivos que varían de sistema a sistema. Los archivos de este directorio incluyen:

/var/adm	contiene archivos de registro y contabilidad.
/var/news	contiene mensajes de interés común.
/var/opt	es la raíz de un subárbol que contiene paquetes de aplicación.
/var/tmp	es un directorio para archivos temporales.
/var/uucp	contiene archivos de registro y estado para el sistema UUCP.

/usr Contiene otros directorios accesibles al usuario tales como /usr/lib y /usr/bin.

Los archivos que varían y que estaban en /usr en la versión 3 se han desplazado a /var.
Los archivos de este directorio incluyen:

/usr/bin	contiene muchos programas ejecutables y utilerías del sistema.
/usr/sbin	contiene programas ejecutables para la administración del sistema.
/usr/games	contiene programas binarios de juegos y datos para juegos.
/usr/share/man	contiene las páginas del manual en línea.
/usr/ucb	contiene paquetes de compatibilidad binarios BSD.

III.3 NETWORK FILE SYSTEM (NFS)

III.3.1 Introducción

El protocolo NFS permite a una estación compartir recursos e información con otras estaciones, minicomputadoras y mainframes. Este protocolo provee acceso transparente entre diferentes arquitecturas y sistemas operativos. Entre las características con las que cuenta están:

- Compartir archivos con otros usuarios de la red sin necesidad de duplicar información, permitiendo así que los requerimientos de almacenamiento en disco sean mucho menores.

- Transferir archivos desde y hacia una estación, esto permite que la red sea controlada desde un punto central, ya que los archivos pueden ser actualizados centralmente.

- Compartir archivos con usuarios de diferentes sistemas operativos tales como UNIX, VMS y DOS.

- Usar los dispositivos conectados a la red tales como impresoras, unidades de disco, etc.

- Intercambiar mensajes con otros usuarios de la red.
- Utilizar comandos de red equivalentes a comandos del sistema operativo UNIX, con los cuales la mayoría de los usuarios de NFS ya están familiarizados.

III.3.2 Vista General.

Es muy usual en ambientes donde se comparte información que se deseen transferir archivos hacia otras estaciones. Esto se hace generalmente mediante diskettes. Si fuera posible desde el mismo lugar enlazarse con otras computadoras, no sólo se podría hacer más fácilmente la transferencia de archivos, sino también se podrían leer archivos y directorios que residieran en

otras estaciones, correr aplicaciones en nuestra estación que residieran en un servidor de la red, utilizar impresoras conectadas a otra computadora de la red, respaldar archivos y directorios a dispositivos compartidos tales como unidades de cinta y discos duros, accesar bases de datos demasiado largas para almacenarse en una estación, etc.

El protocolo NFS incluye un conjunto de aplicaciones que proveen todas estas ventajas en un ambiente de red. NFS utiliza un protocolo denominado Internet Protocol Suite.

Cada computadora de la red debe tener un nombre único de tal forma que el software pueda reconocerlo. Este nombre único es el **Nombre del Dispositivo**. Ya que puede utilizarse cualquier computadora de la red, también debe existir para cada usuario un nombre único, este único nombre es el **nombre del usuario**.

Cualquier computadora en la red que proporcione un servicio a otra computadora se llama **servidor**. Las computadoras que usan cualquiera de estos servicios se llaman **clientes**, las computadoras que proporcionan almacenamiento para grandes cantidades de archivos se llaman **servidores de archivos**. Cada servidor de archivos incluye una gran cantidad de archivos, los cuales deben estar organizados de tal forma que puedan ser accesados rápida y fácilmente por los clientes.

Existen dos formas de acceso a archivos: local y remota. El término local se refiere a la propia computadora o estación, cualquier archivo que esté en el disco de la estación es local. Los archivos existentes en otras computadoras de la red se denominan remotos. De igual forma los dispositivos se denominan locales o remotos dependiendo de su ubicación con respecto a nuestra estación.

Es posible accesar información de una máquina remota de diferentes formasE el método más directo es conectarse a dicha máquina desde nuestra estación, este proceso recibe el nombre de acceso remoto. Una vez que se accesa, se usa la estación como si fuera una terminal de la máquina remota.

Para accesar estos dispositivos remotos tales como impresoras, primero se debe realizar el acceso remoto desde nuestra estación, entonces el dispositivo aparecerá como si fuera un dispositivo local, sin realmente serlo; por ejemplo, cuando se accesa un servidor de archivos remoto desde nuestra estación, dicho

servidor aparece ante nosotros como si fuera simplemente un drive o directorio más de la red; de tal forma bastará con indicar la ruta que se desea utilizar para poder accesarlo.
Para poder accesar un sistema de archivos en una computadora remota, primero debe ser montado el sistema de archivos. También pueden ser montadas impresoras remotas. Una vez que hayan sido montados los sistemas de archivos ya pueden ser accesados como previamente se describió. Posteriormente, este recurso se exporta para que cualquier cliente pueda accesarlo. Frecuentemente, los clientes Unix NFS también fungen como servidores NFS, con lo cual los recursos a los que tienen acceso los usuarios de la red se multiplican cada vez que una nueva estación se agrega a la misma.

Adicionalmente a NFS existen otros sistemas de compartición de archivos como lo son RFS (Remote File Sharing) y AFS (Andrew File System). RFS ya tiene algún tiempo disponible en el mercado, ya que viene incluido en el System V, pero no ha tenido mayor éxito.
AFS apenas está saliendo del medio de la investigación y se está comenzando a utilizar comercialmente. En el mundo existen solamente unos cuantos cientos de usuarios de este sistema, mientras que de NFS existen cientos de miles de usuarios.

Dentro de NFS existen unas funciones llamadas **Demonios.** Estos ***demonios***, sirven para realizar ciertas acciones del cliente y algunas del servidor. A continuación describiremos algunos:

nfsd.- Este corre sobre servidores NFS, y atiende los requerimientos de los clientes, tiene un parámetro llamado nservers, el cual especifica cuantos demonios deberán ser inicializados, el default es ocho. Si se estan utilizando Pc´s como clientes, el demonio que se deberá activar en el server es el pcnfsd, si no se hace esto, el cliente puede montar los file systems del server, pero con el mínimo de privilegios (read only).

biod.- Corre sobre el cliente, maneja la parte del cliente relativa a los bloques de entrada/salida. El parámetro nservers indica cuantos daemons deben ser ejecutados, de igual forma ocho es el valor más común.

rpc.lockd.- Este maneja las solicitudes de locks sobre archivos, tanto el cliente como el servidor ejecutan este demonio, el primero solicita el lock, mientras que el servidor lo garantiza.

rpc.statd.- Este demonio es solicitado por el rpc.lockd para proveer servicios de monitoreo. En particular permite que los locks puedan ser inicializados correctamente después de una colisión en el sistema. Tanto el cliente como el servidor ejecutan este demonio.

rpc.mountd.- Procesa las solicitudes de montaje del cliente, éste es ejecutado por el servidor NFS.

Estos demonios son inicializados desde los archivos de inicialización de las estaciones. Los servidores NFS ejecutan todos los demonios descritos anteriormente, mientras que el cliente solo requiere de biod, rpc.statd, y rpc.lockd.

III.3.3 Exportando Sistemas de Archivos

Antes de configurar un servidor, se debe decidir que file systems serán exportados y que restricciones se pondrán a los usuarios de estos. Sólo deberán exportarse los file systems que provean de algún beneficio para los clientes, así que se deberá preguntar que beneficios va a traer al cliente, algunas razones de peso podrían ser las siguientes:

- Proveer espacio en disco a clientes sin disco duro.
- Prevenir duplicidad innecesaria de la misma información.
- Proveer soporte centralizado a programas y datos.
- Compartir datos entre usuarios en un grupo.
- Proporcionar ambiente gráfico a estaciones, que por limitaciones en el disco duro, no puedan accesarlo localmente.

Existe un archivo denominado /etc/exports, el cual es el archivo de configuración del servidor NFS, y que controla que archivos y directorios serán exportados, que usuarios los pueden accesar y que clase de acceso tienen permitido. El formato en que se especifican los directorios a ser accesados es el siguiente:

directorio [-opción][,opción]...

En **directorio**, se especifíca toda la ruta del directorio a compartir, en cada opción se especifica una condición para la exportación de dicho directorio. Algunas de las opciones válidas son las siguientes:

ro.- Read Only, previene que los clientes puedan escribir en el directorio. Si este no se especifíca, los clientes pueden escribir en el mismo.

rw[=cliente][:cliente].- Permite que uno o más clientes en particular puedan leer y escribir sobre el directorio, si no se ponen los parámetros de cliente, cualquier cliente puede leer y escribir en el directorio.

access=cliente[:cliente].- Limita el permiso para montar este directorio a los clientes especificados, si no se especifica este parámetro, cualquiera puede montarlo. Cuando un servidor está conectado a Internet, siempre deberá tener la opción de acceso, si no todo el mundo en la red tendría montado este directorio.

III.3.4 Configuración inicial del sistema.

Dependiendo de la implementación NFS y en la naturaleza del sistema operativo, casi todos los sistemas no-UNIX solo permiten un modo específico de operación.

La integración de NFS en un cliente es mucho más compleja que la implementación en un server. Los fabricantes de mainframes y minis casi siempre ofrecen implementaciones de server que hacen buen uso del almacenamiento de espacio en disco fijo, lo cual es común en estos sistemas.

Para PC´s las implementaciones de cliente son casi siempre la regla, ya que la distribución de archivos y manejo del espacio en disco para un gran número de PC´s es más barato usando un server central.

NFS puede ser usado para establecer una gran variedad de relaciones cliente/servidor entre sistemas. Las dos razones más prominentes para usar NFS, son el facilitar el manejo del sistema y decrementar los costos en equipo de cómputo mediante la compartición de recursos. En términos reales esto quiere decir que el acceso a archivos remotos a través de NFS viene en dos principales formas:

1. Centralizado: donde uno o más server proveen almacenamiento en disco para la población entera del sistema, de las cuales los clientes pueden no tener disco duro, o tienen limitado el espacio de almacenamiento. Esto tiene ventajas en el manejo de ambos sistemas (el cliente y el servidor), por ejemplo, respaldos centrales y compartición de recursos.

2. Dedicado: donde un server contiene un árbol de archivos en particular, por ejemplo, un server de código fuente o un server en el cual todos los directorios de los usuarios están localizados. Este último es particularmente útil, ya que es posible crear una vista única del sistema en donde un usuario puede conectarse a través de cualquier sistema en la red y puede siempre accesar su directorio personal bajo el mismo path o ruta.

Claramente, montajes remotos tienen también sus desventajas, si por alguna razón, el sistema remoto que actúa como server no está activo, las aplicaciones y utilerías pueden cesar de operar si ellos requieren datos o alguna respuesta del server remoto. Con muchos montajes remotos activos, la posibilidad de falla de un server se incrementa considerablemente.

Un ejemplo clásico de este problema es cuando se introduce el comando "df" en la estación, el cual muestra todos los file systems activos en el momento. Si algún file system remoto no está disponible (por que la maquina se apago o se perdió la conexión), el comando "df" muestra el status del problema, indicando que no puede conectarse a ese file system.

III.3.5 El protocolo NFS como estándar de UNIX.

NFS fue anunciado en 1984. La primera implementación fue introducida el año siguiente en una estación de trabajo SUN. Este producto tenía versión 2.0.

En 1986 se implemento NFS Versión 2.0 al Sistema V Versión 2 para una DEC VAX. Esta implementación mostró que era posible hacer disponible a NFS en prácticamente cualquier computadora UNIX. En este mismo año se liberó la versión de NFS para PCs bajo MS-DOS.

En 1987 se implemento NFS al Sistema V Versión 3. La versión 4.0 de NFS, liberada en 1989 ya contenía facilidades de encriptamiento las cuales pueden ser usadas para garantizar que el acceso a archivos no sea violado.

Una implementación de NFS para UNIX está disponible para todos los fabricantes interesados y muchos de estos fabricantes han introducido este sistema o una variante de ella a sus máquinas pagando licencias a SUN. En el mundo UNIX ha llegado a ser un estándar por omisión para acceso de archivos distribuidos y hoy en día está disponible prácticamente para todos los sistemas UNIX.

Está confirmado que un sistema UNIX sin NFS no tiene oportunidad en el mercado. NFS es un componente del Sistema V Versión 4 de diciembre de 1989 y fue incorporado al System V Interface Definition (SVID).

La mayoría de las estaciones de trabajo UNIX incluyen los protocolos NFS y TCP/IP y comandos de los mismos como parte de su configuración básica. Esta tendencia se incrementará con la integración de NFS dentro del Sistema V y otros fabricantes líderes en UNIX y con la inminente tecnología de red de todas las estaciones y computadoras personales. Los productos NFS están disponibles para otros sistemas operativos como: MS-DOS, VMS, MVS, etc.

La idea original del NFS es que el acceso a sistemas de archivos remotos sea transparente y equivalente al acceso de sistemas de archivos locales.

Un aspecto de esta transparencia es la velocidad de acceso de datos sobre la red, la cual debe ser tan alta que no haya diferencia notable de un acceso a disco local. La meta original en el desarrollo NFS fue lograr una tasa de acceso de datos de un 80% en comparación con un disco duro local.
Una ventaja de la implementación de NFS bajo UNIX es esta interface transparente. Un programa no se da cuenta cuando está accesando un archivo localmente o sobre NFS. Para lograr esto las llamadas al sistema deben ser ejecutadas a un nivel externo para el programa en el kernel del sistema operativo UNIX. En otros sistemas operativos, este mecanismo es conocido como "redirector", ya que este redirecciona las llamadas normales al sistema. Este redirector actúa entre el server y el cliente.

El server NFS no solo soporta accesos a archivos normales y directorios, sino también a archivos especiales que pueden ser abiertos, y ellos son interpretados

y procesados como recursos en la computadora local. Todos los archivos especiales tienen algo en común: El método de acceso utilizado para intercambio de datos con ellos no está implementado en el "file system" mismo, en lugar de esto, estos procesos en el file system solo contienen apuntadores a otros módulos tales como "device drivers" en el sistema operativo UNIX. De este modo los accesos de lectura y escritura no se hacen directamente en el sitio de almacenamiento, sino en donde indica el apuntador, así, el floppy disk drive conectado al cliente, es accesado, por ejemplo, abriendo el directorio /dev/floppy en el server.

Pero además debe haber una forma de hacer equivalentes los archivos que el cliente está accesando desde NFS en comparación con los locales. Esto se logra siguiendo un procedimiento de sintaxis.

La forma de reconocer la sintaxis es la siguiente:

Cuando un programa de aplicación pide tener acceso a un archivo, usa la sintaxis de la ruta del nombre "path" para escoger entre procedimientos de acceso de archivos locales o remotos. Si la ruta (path) se refiere a un archivo remoto, el sistema usa el software NFS del server para accesar el archivo remoto. Si la ruta se refiere a un archivo local, el sistema usa el software propio de la estación para accesar el archivo.

Recordemos que NFS fue diseñado para integrar diferentes sistemas de computadoras.

Cuando el administrador del sistema instala en el sistema operativo un cliente NFS, trata de integrarlo con el esquema de convenciones de nombres del sistema de archivos, esto debido a que el path o ruta del archivo puede diferir entre los diferentes tipos de máquinas. Por ejemplo cuando una maquina corriendo MS-DOS es integrada como un cliente NFS y conectada a un server NFS corriendo UNIX, el sistema de archivos del cliente usa la diagonal invertida (\), como separador de directorios o caracteres, mientras el sistema de archivos del server una la diagonal normal (/).

Para acomodar diferencias potenciales entre la sintaxis del "path" o ruta de un archivo entre el cliente y el server, NFS sigue una regla muy simple: solo el lado del cliente interpreta rutas completas de nombres. Para seguir una ruta completa a través del sistema de directorios jerárquico del server, el cliente envía un

componente individual del path a la vez, recibiendo información acerca del directorio o archivo que se está llamando. Por ejemplo, si un cliente que usa la diagonal (/) como separador necesita encontrar el path name "**/a/b/c**", este inicia obteniendo información acerca del directorio raíz del server. Este cliente pregunta al server por el nombre "**a**" en ese directorio. El server envía la información acerca de "**a**". Y la información mostrará que "**a**" es un directorio. El cliente entonces pregunta por el nombre "**b**" en ese directorio. Cuando el server responde, el cliente verifica que "**b**" es un directorio, y si éste es, pregunta al server por el nombre "**c**" en el. Finalmente el server responderá enviando la información acerca de "**c**".

Cuando los administradores instalan un cliente NFS en UNIX ellos usan la facilidad **mount** del file system para integrar directorios remotos dentro del sistema de directorios jerárquico de UNIX. Entonces se empieza a trabajar con el "mount" del file system. El administrador crea un directorio vacío en el sistema existente y entonces "monta" un file system NFS remoto en el.

Ya que el usuario bajo NFS no ve diferencia entre el sistema de archivos montado y el disco duro local, el espera la misma velocidad en ambos casos, lo cual es algunas veces imposible. La primera limitantes es dada por el disco duro del server, ya que, por supuesto, NFS no puede transportar los datos mas rápido que este dispositivo, aunque, como mencionamos anteriormente, la meta original en el desarrollo de NFS fue lograr una tasa de acceso de datos de un 80% o más para en comparación a un disco local. La velocidad de acceso es también validada por factores tales como el performance de la implementación de TCP/IP que se tenga y el controlador ETHERNET usado, el cual debe tener la suficiente capacidad de buffers para aceptar requerimientos de lectura y escritura de longitud máxima (8800 bytes).

Especificación del Protocolo NFS.

Existen varios procedimientos principales dentro del protocolo NFS, los cuales mostraremos a continuación:

Null.- Como otros servicios RPC, el protocolo NFS comienza con el procedimiento 0, el cual no contiene ningún argumento y regresa un resultado tipo void.

Actualización de atributos de archivos.- Este procedimiento regresa los atributos del archivo especificado en el file handle,el cual contiene entradas que identifican al archivo en el servidor NFS, por ejemplo, como resultado de las llamadas UNIX: access, open, stat y fstat. Los atributos del archivo son almacenados en el cliente NFS como un caché, de tal forma que esta información constantemente utilizada este siempre lista.

Las entradas en el caché siempre tienen fecha y hora, de tal forma que si la diferencia entre el tiempo de entrada y la hora actual es mayor a 3 segundos para archivos y 30 segundos para directorios, la información es leída nuevamente del servidor.

Establecimiento de atributos de archivos.- Este procedimiento se usa para fijar atributos, por ejemplo, a través de las llamadas de UNIX: chown, fchown, chmod, fchmod, truncate, ftruncate y utime. Los atributos modificados se regresan como un resultado.

Revisar nombre de archivo.- Provee al cliente con un file handle el cual corresponde a un nombre de archivo en un directorio especificado, con lo cual hace posibles las subsecuentes operaciones con el archivo o directorio. El cliente utiliza este procedimiento para encontrar su camino dentro del sistema de archivos de otra computadora o servidor NFS.

El cliente utiliza **Revisar** para conducir una búsqueda paso a paso entre las rutas que se le han proporcionado y, en ciertos casos, puede incluso llamar a algún procedimiento RPC individual para cada componente de la ruta.

Leer el contenido de una liga simbólica.- Los sistemas BSD definen ligas simbólicas o archivos que contienen la ruta de otro archivo en lugar de datos y son por ende, apuntadores a otros archivos. Si el sistema reconoce un archivo de liga simbólica, el apuntador es leído por el sistema incluso bajo NFS. Es entonces evaluada y el archivo especificado es direccionado.

Leer un archivo.- Con este procedimiento, que se activa después de una llamada de lectura o ejecución, el cliente NFS lee un archivo perteneciente al servidor, su última parte regresa atributos de los datos y del archivo, la variable offset se fija exclusivamente para el contador de lecturas del archivo en el servidor con cada llamada.

Escribir en un archivo.- Este procedimiento, que es activado después de una llamada de escritura, es utilizado por el cliente NFS para escribir en un archivo perteneciente al servidor. La longitud de los datos a ser escritos está dada en la longitud del campo del área de datos.

Crear archivo.- Esta rutina es activada por las llamadas del sistema create, mknod y open, y genera nuevos archivos en el servidor. El resultado de esta llamada es el file handle para el nuevo archivo.

Borrar archivo.- Este procedimiento, que es activado por la llamada del sistema: unlink, borra archivos del servidor.

Renombrar archivo.- Soporta el renombramiento de un archivo o directorio, por ejemplo, en la llamada del sistema: rename.

Generar liga.- Este procedimiento es activado por la llamada del sistema: link. Genera un liga dura dentro de un sistema.

Generar liga simbólica.- Este procedimiento es activado por la llamada del sistema: symlink, y genera una liga simbólica.

Crear directorio.- Este procedimiento es activado por la llamada: mkdir, y genera un directorio vacío.

Borrar directorio.- Esta rutina es activada por la llamada: rmdir, y borra un directorio.

Leer directorio.- El resultado de este procedimiento es una lista de entradas de directorio, cada una de las cuales consiste de un nombre de archivo, un número de índice de nodo y un apuntador a una posición en el directorio. Este procedimiento es necesario porque la estructura de directorios de cada sistema UNIX es diferente. Cuando se hace un intento de leer un directorio en NFS con el comando READ, éste es automáticamente rechazado por el servidor, sin embargo, algunos sistemas UNIX convierten estas operaciones a llamadas READDIR de NFS.

Leer atributos del sistema de archivos.- Este procedimiento proporciona información sobre el sistema de archivos montado, el número de bloques

presentes o libres, sus tamaños y el tamaño óptimo de transferencia para solicitudes de lectura/escritura.

Características particulares del protocolo NFS.

Los diseñadores de NFS estaban interesados en diseñar un servicio de acceso a archivos sumamente robusto, el cual erradicara el doloroso problema de reinicializar después de una falla en el servidor o el cliente.

La solución fue un protocolo libre de estado, en el cual el servidor no tuviera que almacenar ninguna información sobre el estado actual o el progreso del diálogo con el cliente. Por lo tanto, el protocolo NFS no tiene las operaciones comunes de abrir y cerrar. A diferencia de las rutinas estándares de E/S de UNIX, las posiciones de lectura y escritura siempre son proveídas por el servidor.

Dado que el file handle de un archivo debe ser siempre no-ambiguo, es posible que después de una falla en el servidor y un arranque subsecuente, continuar con la operación de lectura en forma transparente, en lo que al cliente corresponde.

Mientras que el servidor no esté disponible, el cliente retransmite las solicitudes de lectura con los mismos parámetros para el servidor a intervalos regulares hasta que el servidor responda.

Este comportamiento es particularmente ventajoso para estaciones sin disco, que no pueden ejecutar funciones importantes sin un servidor.

Dado que NFS no puede ser 100% protegido contra servidores sobrecargados con grandes tiempos de respuesta o paquetes perdidos, aquí también debemos trabajar con RPCs redundantes, de tal forma que estas llamadas no causen problemas a la implementación de un servidor NFS bajo UNIX que usa un cache para cargar llamadas previamente terminadas junto con su identificador de transacción. El caché se utiliza para reconocer repeticiones de dichas solicitudes de transacciones RPC ya almacenadas en el cache.

III.3.6 Montaje y Desmontaje Remotos.

Antes de decidir que directorios de NFS serán montados en el sistema, debemos saber que servidores están conectados a la red y que directorios están disponibles de cada uno de ellos. Debemos recordar que un directorio no puede ser montado a menos que primero sea exportado por un servidor.

El administrador de la red es una buena fuente de información a este respecto. El puede decir que sistemas están brindando servicios NFS, que directorios se están exportando y cual es el contenido en estos directorios. También se puede obtener información sobre los directorios exportados directamente desde los servidores utilizando el comando **exportfs** (para la versión de SunOS). Al teclear este comando, se listarán los directorios que el server exporta para poder ser utilizados por otras estaciones.

Generalmente, cada usuario decide que archivos va a compartir con los demás y cuales prefiere mantener en forma local, por lo cual es conveniente definir políticas para la exportación y montaje de sistemas de archivos remotos.

Montaje.

Existe un comando llamado **mount** que sirve para montar los sistemas de archivos que se hayan seleccionado para utilizar por el cliente. Su estructura general es la siguiente.

mount servidor:directorio-remoto directorio-local

El servidor indica el nombre del servidor NFS donde está el directorio a compartir, directorio-remoto indica la ruta en el servidor donde se encuentra el directorio a compartir. El comando **mount** agrega el directorio remoto al file system del cliente y lo identifica con el nombre que se le haya asignado en directorio-local. Este directorio-local se denomina punto de montaje y debe ser creado antes de que se ejecute el comando mount con un simple mkdir. Una vez completado el comando mount, los archivos localizados remotamente pueden

ser accesados a través del directorio local que se definió para este caso, como si fueran archivos locales.

Desmontaje.

Una vez montado un directorio remoto permanecerá así hasta que específicamente sea desmontado con el comando umount, como parámetro se debe especificar, ya sea el nombre asignado al directorio-local o el del directorio-remoto indicando el servidor en que se encuentra localizado dicho directorio-remoto.

Si se llegara a apagar la computadora del cliente, automáticamente serían desmontados los file systems remotos que en ese momento estuvieran activos. Existe la posibilidad de remontar automáticamente los sistemas de archivos remotos después de una inicialización de la estación.

El archivo /etc/fstab provee la información que se utiliza para remontar los directorios de NFS después de bootear el sistema. Es muy fácil construir este archivo porque la opción **-p** del comando mount crea el archivo de la información existente sobre los montajes realizados (este archivo puede ser creado o actualizado directamente con el editor vi). Una vez montados todos los sistemas de archivos remotos, se teclea lo siguiente:

```
mount -p > /etc/fstab
```

Finalmente, se coloca un código de inicialización en los comandos de arranque de NFS donde se indica que debe montar los sistemas de archivos remotos, el cual es: mount -vat nfs. Donde los parámetros -v,-a y -t indican desplegar mensaje sobre cada directorio montado, montar todos los file systems especificados en fstab y montar sólo los sistemas de archivos relacionados con NFS respectivamente.

III.4 Demonios del sistema y del usuario.

Se puede revisar que los demonios correspondientes al sistema se encuentren corriendo sin problema alguno y checar si cada uno de los usuarios tienen demonios.
A nivel de administración se pueden inicializar y dar de baja los demonios.

Comandos relacionados : cron.

El sistema V de UNIX permite ejecutar automáticamente programas en tiempos predeterminados. La utilización de la orden **at** proporciona una forma fácil de construir demonios de usuarios, que se definene como: procesos subordinados que realizan trabajo útil para un usuario específico.

A los usuarios se les puede conceder o denegar selectivamente permiso para utilizar **at** en la planificación de los trabajos si no existen los archivos adecuados (**at.allow ni at.deny**) en su sistema, entonces no se permite a los usuarios normales el uso de **at**; solamente **root** (el superusuario) puede planificar un trabajo. El contenido de estos archivos determina quien puede utilizar la orden **at**.

Ejemplo de creación de un demonio. al borrar **(del)** los archivos, estos son movidos a una papelera en lugar de ser eliminados. Aunque esto es conveniente, existe un problema. La papelera se llenará con archivos viejos y requerirá alguna limpieza para eliminar los archivos inútiles.

Se puede automatizar esta limpieza con un simple demonio:

```
#
# deamon - un demonio de limpieza para mantener limpia la papelera.
#
cd $ [WASTEBASKET:-$HOME/.JUNK]
rm -rn *
at midnigt Friday << !
deamon
!
```

Cuando se ejecuta este guión DEAMON cambiará el directorio en WASTEBASKET y eliminará todos los archivos ahí contenidos.

La idea anterior puede extenderse a varios demonios cada uno planificado para realizar un trabajo diferente (con comandos como: touch, set, trap, exit, xargs, etc.)

III.6 Funciones del administrador de un sistema UNIX.

En todos los sistemas Unix debe haber, por lo menos, una persona a cargo del mantenimiento y operación del sistema; dicha persona es conocida como superusuario o administrador del sistema. Es responsabilidad del administrador del sistema asegurar la correcta operación del sistema, así como llevar a cabo las tareas que requieran de privilegios especiales.

Dependiendo del tamaño del sistema, las aplicaciones que se ejecuten y el número de usuarios en el mismo, el trabajo de un administrador será requerido desde una vez al día hasta la necesidad de tener un administrador de tiempo completo. Aún si el sistema es pequeño, el administrador del sistema deberá realizar cuidadosamente cada una de las tareas de mantenimiento requeridas, ya que un mantenimiento incorrecto puede degradar el desempeño del sistema.

El administrador del sistema deberá mantener un registro de todas las modificaciones al sistema, así como de eventos que en éste ocurran. Cada uno de los eventos, mensajes, respaldos, o modificaciones deberían ser registradas con la fecha, hora, y nombre de la persona que lo ejecute, así como de las circunstancias alrededor del evento. Por ejemplo, si se adicionó una nueva aplicación al sistema, se deberá crear una historia en el archivo de registro. Este registro deberá incluir la hora, fecha, y el nombre de la persona que realizó la instalación, y cualquier nota acerca de la aplicación o de la instalación que pudiera ser de utilidad. Un archivo de registro detallado es una gran ayuda para realizar el diagnóstico de problemas del sistema y tener controlado el crecimiento y uso del sistema.

Aunque todas las funciones que se mencionarán en esta sección se presenten desde el punto de vista de un administrador, varias de ellas podrán ser efectuadas por usuarios normales. No obstante, debido a que algunas de las

tareas podrían cambiar dramáticamente la operación y desempeño del sistema, se recomienda que, siempre que sea posible, el administrador sea el que realice dichas tareas. Sin importar si es el administrador o alguno de los usuarios normales quien realice alguna de las operaciones, deberá quedar la huella en el archivo de registro. Apegándose a estas reglas se pueden prevenir cambios indeseados o innecesarios en el sistema.

Algunas de las tareas básicas del administrador las tiene que efectuar en forma diaria.

- Asegurar que la integridad del sistema no sea violada, a través del uso de mecanismos de seguridad.
- Asegurar que los respaldos (copias regulares de archivos del sistema) sean hechos correctamente y almacenados para su uso futuro.
- Controlar los problemas relacionados al uso de recursos limitados del sistema, tal como espacio en disco, número de procesos.
- Ofrecer a todos los usuarios un soporte y consultoria general.

A continuación se presenta una lista de las tareas que debe realizar un administrador, agrupadas de acuerdo a la frecuencia con que éstas se deben efectuar. Dichas tareas se deberán realizar en el siguiente orden más o menos, dependiendo del tamaño y la complejidad del sistema.

- Registrar todas las modificaciones al sistema y eventos que se presenten en el archivo de registro.
- Responder a las llamadas por pánicos, caídas del sistema, consultas de usuarios.
- Realizar el mantenimiento de la seguridad del hardware, el software y acceso a archivos de datos.

III.6.1 Tareas diarias.

- Realizar respaldos
- Llevar control sobre los niveles de utilización del sistema.
- Verificar los procesos que se estén ejecutando.
- Checar espacio en disco.

- Checar la funcionalidad del mail, conexiones.
- Checar el estatus de las impresoras.
- Checar las salidas de las auditorías, en caso de que esta opción esté habilitada.
- Checar las ligas de comunicaciones, en caso de que haya activadas.
- Checar sesiones abiertas no atendidas.
- Borrar archivos core y *.out.

III.6.2 Tareas mensuales.

- Correr fsck en todos los filesystems.
- Checar los reportes del administrador de impresión.
- Checar los archivos de registro, tal como /etc/wtmp y los de /usr/adm y /usr/spool y limpiar, cortar o truncar la información.
- Utilizar **sar** para generar un reporte de la actividad.
- Generar un reporte detallado de la utilización de espacio en disco por parte de cada uno de los usuarios.
- Borrar archivos temporales y archivos de **lost+found**, para liberar espacio.
- Realizar respaldos generales.
- Archivar archivos críticos, en caso de que estos sean modificados.
- Reafinar el sistema y reasignar recursos, en caso de ser necesario.
- Realizar el mantenimiento de hardware.
- Cambiar passwords de entrada, en caso de ser necesario.
- Cambiar el password del superusuario, en caso de ser necesario.

III.6.3 Tareas ocasionales.

- Realizar actualizaciones del sistema operativo y aplicaciones, en caso de ser necesario.
- Corregir permisos en programas.
- Realizar la redistribución del espacio en los filesystems.

III.7 Procesos y puntos a considerar para el monitoreo y administración.

Para efectos de este proyecto, las funciones y procesos que vamos a considerar para realizar el monitoreo y las funciones básicas de administración son :

III.7.1 Servidores activos en la red.

Como monitoreo, se puede verificar cuantos servidores se encuentran vivos dentro del sistema y la dirección **IP** de cada uno.

III.7.2 Sistema de Archivos.

A manera de monitoreo, se puede obtener la información correspondiente a los file systems existentes montados en un sistema Unix. Información tal como nombre del dispositivo que controle el file system, directorio en el que se encuentre montado, tipo de file system y porcentaje ocupado del mismo. La información de la capacidad ocupada y porcentaje disponible de cada uno de los file systems se puede considerar como crítica ya que uno de los problemas que se presenta en los sistemas Unix donde no hay una constante administración es que, sin darse cuenta los usuarios llenan el espacio de trabajo y empiezan a aparecer errores como consecuencia.

A nivel administración, lo que se puede permitir es el listado de archivos, navegación en los directorios, borrado de archivos, borrado de directorios y reubicación de archivos y directorios.

Comandos relacionados : ls, df, mv, cp, copy, cd, du, fstat, fsck.

III.7.3 Usuarios.

Para efecto de monitoreo de usuarios, se puede checar toda la información relativa a los mismos y; a nivel administración, se puede llevar a cabo al alta, baja y modificación de usuarios.

Comandos relacionados : finger, (mkuser o adduser), rmuser

III.7.4. Sesiones y procesos.

Por lo que respecta a las sesiones, se puede monitorear cuantas sesiones y usuarios correspondientes se encuentran vivos en el sistema. Verificar en que dispositivo se encuentra cada sesión, el tiempo ocioso de las mismas. Adicionalmente, se pueden checar los procesos que se encuentren vivos dentro del sistema y, de alguna manera, identificar a que usuario pertenecen.

A nivel administración se permite terminar sesiones y procesos.

Comandos relacionados : who, kill, ps, pstat.

Capítulo IV.Interfaces Gráficas de Usuario (GUI's).

IV.1. Introducción.

Uno de los desarrollos de más éxito para el sistema Unix, y el que probablemente tenga mayor efecto en el modo en que se utiliza Unix, es la llegada de las Interfaces Gráficas de Usuario (GUI). Las interfaces Gráficas de Usuario sustituyen al estilo de línea de orden de interactuar con el sistema Unix por el basado en menús, íconos y selección y manipulación de objetos. En vez de tener que recordar órdenes y acciones de órdenes, estas interfaces permiten trabajar directamente con representaciones gráficas de objetos (archivos, programas, listas) y seleccionar acciones.

Aunque las interfaces gráficas han sido de uso común durante bastante tiempo en algunos otros sistemas, el desarrollo de una interface gráfica de usuario para el sistema Unix ha dependido de la creación de entornos gráficos que satisfagan las necesidades especiales de las aplicaciones del sistema Unix. En particular par ser utilizables de forma general con el sistema Unix, los entornos gráficos deben soportar aplicaciones en RED, deben permitir que las aplicaciones sean independientes del Hardware específico de pantalla y terminal y deben permitir que las aplicaciones gráficas sean fácilmente portables a través de la variedad de hardware en que el sistema Unix corre. Se han desarrollado dos entornos gráficos que satisfacen estas necesidades: el sistema **X Windows** del MIT y **Motif** de Open Software Foundation.

Para resolver estos problemas se han desarrollado productos pensados para proporcionar una interface de usuario tanto para el sistema Unix como para aplicaciones aportadas por diferentes vendedores. Dos de estas interfaces son: OPEN LOOK, desarrollado por AT&T y MOTIF desarrollada por Open Software Foundation.

IV.2 Interfaces

En los siguientes puntos explicaremos algunas de las principales interfaces que con el transcurso del tiempo se han venido perfeccionando hasta llegar a lo que hoy en día significa un ambiente amigable para un usuario final.

IV.2.1 X Window System.

X Window es un sistema de ventanas distribuido, multitarea y transparente a la red, originalmente desarrollado por el MIT para comunicaciones entre terminales X y estaciones de trabajo Unix.

El sistema X Window proporciona un entorno completo para el desarrollo y ejecución de aplicaciones que aporta interfaces gráficas de usuario en red. Los conceptos principales en los que está basado incluyen un modelo *cliente-servidor* para el modo en que las aplicaciones interactuan con los dispositivos terminales, un *protocolo de red*, varias herramientas de *Software* que pueden ser utilizadas para crear aplicaciones basadas en X Window y una colección de aplicaciones de utilidad que proporcionan características de aplicación básicas.

El sistema gráfico X Window proporciona un entorno para desarrollo de interfaces gráficas de usuario además del contexto en el que las interfaces gráficas se ejecutan. Sin embargo, por sí mismo no proporciona una "apariencia y tacto" específico. Por ejemplo, dos aplicaciones diferentes basadas en el sistema X Window pueden tener aspectos y estilos de operación muy diferentes. Los comandos en que se representan los menús, las acciones, el modo en que la aplicación convierte una ventana en un icono, y otras características fundamentales, pueden funcionar de forma distinta. Las inconsistencias resultantes pueden ser superiores a los beneficios potenciales de tener interfaces gráficas.

Un concepto fundamental de X Window es la separación de las aplicaciones con respecto al Software que maneja la entrada y salida de terminal. Todas las interacciones con dispositivos terminales, -la visualización de información en una pantalla, la recogida de pulsaciones de tecla o de pulsaciones de botones de

ratón- son manejadas por un programa dedicado (*Servidor*) que es totalmente responsable del control de la terminal. Las aplicaciones (*Clientes*) envían al servidor la información a visualizar y el servidor envía a las aplicaciones información referente a las aplicaciones de entrada de usuario.

La separación de las aplicaciones (*Clientes*) que gestiona las pantallas (*servidor*) significa que sólo el servidor necesita conocer los detalles del Harware de la terminal y como controlarla. El servidor "oculta" las características del Hardware de las terminales a las aplicaciones. Esto hace más fácil el desarrollo de las aplicaciones y hace que sea relativamente fácil de portar aplicaciones X Window existentes a nuevas estaciones.

Por ejemplo supongamos que las instrucciones para dibujar una línea difiere en dos terminales diferentes. Sí una aplicación se comunica directamente con la estación, entonces, terminales diferentes requieren versiones diferentes de la misma aplicación.

Sin embargo, si las instrucciones de Hardware específicas son manejadas por los servidores, una aplicación puede enviar la misma instrucción al servidor asociado con cada terminal, y el servidor de terminal puede hacer corresponder la misma instrucción con las señales de control adecuadas para la terminal. Como resultado, la misma aplicación puede ser utilizada con muchos dispositivos terminales diferentes.

Con el modelo Cliente-Servidor, cada nuevo dispositivo terminal requiere un nuevo servidor, pero una vez proporcionado un servidor, las aplicaciones existentes pueden funcionar con esa estación sin modificaciones.

La figura siguiente ilustra el modelo Cliente-Servidor del modelo X Window. muestra aplicaciones X (Clientes) que corren en dos máquinas Hosts y en una estación de trabajo. Estas aplicaciones son accesibles desde estaciones de trabajo o terminales X (Servidores) bien en la misma máquina o distribuidas en una red. Se puede observar que en la pantalla no hay distinción entre una aplicación X que corre en la máquina local X y otra que corre en una máquina remota.

La existencia de un servidor especial para cada tipo de terminal es una de las partes del modelo Cliente-Servidor. La otra es el uso de un modo estándar para

que las aplicaciones Cliente se comuniquen con los Servidores. Esto lo proporciona el protocolo de red X. (Fig. 15)

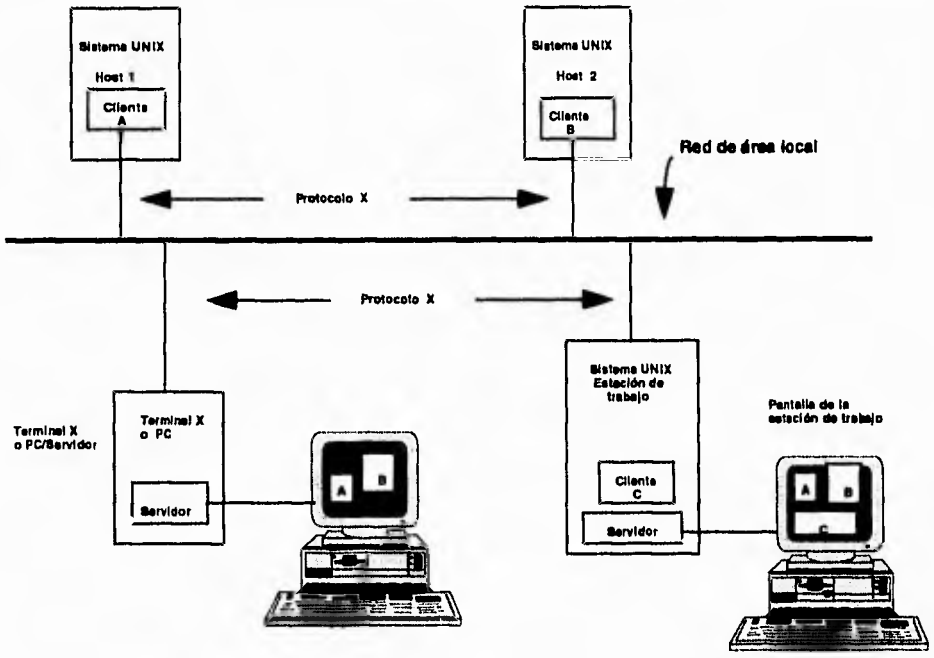


Fig 15. Modelo Cliente-Servidor del sistema X Window.

El protocolo X es un lenguaje estándar utilizado por las aplicaciones Cliente para enviar instrucciones a los servidores X y utilizado por los servidores para enviar información a los clientes. En el sistema X Window los clientes y los servidores se comunican *únicamente* a través del protocolo X.

IV.2.2 Motif

Una interface gráfica de usuario para el sistema Unix es Motif, desarrollada por Open Software Fundation y basada en trabajos de Hewlett Packard y Digital Equipment Co. Al igual que la interface gráfica de usuario OPEN LOOK ,Motif está construida sobre el fundamento proporcionado por el sistema X Window. Puesto que ambos se construyen sobre una base común, y puesto que ambos

responden a análogas necesidades de usuario, hay muchas similitudes importantes entre los dos.

Al igual que OPEN LOOK, Motif proporciona un ratón con varios botones, que utiliza operaciones de Selección y Menú. Motif emplea el modelo de - Seleccionar- después - Operar, para actuar sobre los objetos. Proporciona un gestor de ventanas que permite mover, cambiar de tamaño, crear iconos y terminar ventanas de aplicación con el ratón o mediante controles de teclado. Motif proporciona un modo estándar de accesar a menús de operaciones, operaciones de contar y pegar basadas en ratón y controles estándares análogos para OPEN LOOK.

Algunas diferencias entre Motif y OPEN LOOK incluyen el aspecto de las ventanas, detalles específicos de asignación de los botones del ratón, el modo de mover y cambiar de tamaño las ventanas, el uso de ciertas caraterísticas del teclado y que aplicaciones se incluyen con el sistema. Motif proporciona una aplicación xterm, pero no incluye un gestor de archivos.

IV.2.3 Microsoft Windows

Microsoft Windows significó el auge de las computadoras personales, ya que hasta antes de que surgiera, las aplicaciones en computadora casi siempre debían ser utilizadas por personal con ciertos conocimientos de computación, y con la introducción de esta poderosa herramienta, la computación se hizo accesible a los usuarios que querían aprovechar todo el potencial de las computadoras sin tener que aprender computación. Basado en el modelo original de Windows de Xerox y posteriormente perfeccionado por Apple Computers, hoy en día Microsoft Windows señala el camino a seguir por todas la aplicaciones que se desarrollen en el futuro.

Cualquiera que sean las herramientas de desarrollo que se utilicen, la programación Windows es diferente a cualquier estilo de programación por lotes o a la orientada a las transacciones. Para poder explicar esto es necesario conocer algunas cosas fundamentales acerca de Windows.

Procesamiento de Mensajes.
Cuando se escribe una aplicación para MS-DOS, el único requisito absolutamente necesario es una rutina principal, en el caso de la programación en C se bautiza con el nombre de main(). El sistema operativo llama a un main cuando el usuario ejecuta el programa y desde ese punto en adelante, se puede emplear cualquier estructura de programa que se desee. Si el programa necesita recibir pulsaciones de teclado o de otro modo usar los servicios del sistema operativo, entonces llama a una función especial como por ejemplo getchar().
En cambio cuando el sistema operativo Windows ejecuta un programa, llama a la función WinMain que deberá encontrarse en alguna parte de la aplicación, y que efectúa unas tareas específicas. La tarea más importante es la de crear la ventana principal de la aplicación y que debe contar con su propio código para procesar los mensajes que Windows le envía. Una diferencia fundamental entre estos dos procedimientos es que la aplicación en MS-DOS hace una llamada al sistema operativo para obtener una entrada del usuario, mientras que la aplicación en Windows procesa la entrada del usuario por medio de mensajes. La forma de procesar estos mensajes es función del Marco de la Aplicación.

Muchos mensajes en Windows están definidos en forma rigurosa, y se aplican en todos los programas. Por ejemplo un mensaje WM_CREATE es enviado a medida que se crea una ventana; se envía un mensaje WM_LBUTTONDOWN cuando el usuario pulsa el botón izquierdo del ratón; se envía el mensaje WM_CHAR cuando el usuario teclea un caracter.

Por último se envía un mensaje WM_CLOSE, cuando el usuario cierra una ventana. Otros mensajes (los mensajes de tipo [orden]) son enviados a la ventana de una aplicación y en respuesta a selecciones de menú que hace el usuario. Estos mensajes dependen de la disposición del menú de la aplicación. El programador puede definir aún otros mensajes que se conocen como [mensajes del usuario].

Por el momento no hay que preocuparse ya que el código se encarga de procesar esos mensajes. En efecto, este es el trabajo del Marco de la Aplicación. No obstante, se debe ser conscientes de que los requisitos de procesamiento de mensajes de Windows imponen una gran cantidad de estructura al programa. De tal manera, no se debe intentar forzar que los programas para Windows se parezcan a los antiguos programas para MS-DOS.

Muchos programas de MS-DOS se escribieron de forma directa en la memoria de video y en el puerto de impresión. La desventaja de esta técnica era la necesidad de tener que proporcionar programas controladores para cada tarjeta de video y para cada modelo de impresora. Windows introdujo una capa de abstracción denominada Interface de Dispositivos Gráficos (GDI). Windows proporciona los controladores para la visualización y para la impresora, de tal manera que el programa no necesita conocer el tipo de tarjeta de video ni la impresora que está conectada al sistema. En efecto, en lugar de accesar al equipo, el programa llama a una funciones de la GDI, las cuales accesan una estructura llamada contexto de dispositivo. Windows establece una correspondencia entre la estructura con texto de dispositivo y un dispositivo físico y emite las instrucciones de E/S apropiadas. La GDI es casi tan rápida como el acceso directo al video y permite además que distintas aplicaciones escritas para Windows puedan compartir la pantalla.

IV.3 Programación en ambiente Windows.

La Programación en Windows, tiene su principal diferencia de la programación tradicional en que aquí se define cada entidad como un objeto, el cual contiene ciertas propiedades que definen su apariencia y comportamiento, así que, por ejemplo para definir un cuadrado en la pantalla, se define un objeto que aquí vamos a denominar cuadro, con propiedades como tamaño, forma, contenido, colores, texturas, etc. Así podemos hacer que apartir de cierto objeto en la pantalla, se ejecuten ciertos procedimientos al seleccionar dicho objeto por el usuario.

Para definir una Interface Gráfica de Usuario (GUIs) necesitamos: Crear la Interface, Definir sus propiedades y Escribir el código de programación. De esta forma, primero se dibujan en pantalla los objetos que deberán ir dibujados en ella, posteriormente se establecen las propiedades que contendrá cada objeto previamente definido y finalmente se realiza la programación de las diferentes rutinas que serán ejecutadas de acuerdo a la selección de cada objeto.

Entre los objetos que se definen para una aplicación están las formas, los controles y otros objetos. Las formas se utilizan principalmente para delimitar el área que será destinada para la captura o despliegue de información en los dispositivos de salida ya sea pantalla, impresoras, discos, etc. Los controles por

su parte se utilizan para la entrada de datos y el despliegue de información para el usuario. Algunos tipos de controles son las cajas de texto, los botones de comandos y las cajas de listados. Cada control tiene sus propiedades y eventos, esto es, las propiedades definen sus características y los eventos definen los pasos a seguir al seleccionar dicho control.

IV.3.1 Programacíón Basada en Recursos.

En MS-DOS la programación es controlada por datos, esto es, los datos tienen que ser codificados como constantes inicializadas, o bien se deben proporcionar archivos de datos separados para que el programa los pueda leer. En la programación en Windows, los datos se almacenan en un archivo de recursos usando diversos formatos. Windows se encarga de separar el archivo de recursos en un programa enlazado, por medio de un proceso llamado ligadura. Los archivos de recursos pueden incluir mapas de bits, iconos, definiciones de menús, disposiciones de cuadros de diálogo y cadenas, incluso pueden contener formatos de recursos personalizados que hayan sido definidos.

Se pueden utilizar editores de texto para editar un programa, pero generalmente se utilizan herramientas del tipo wysiwyg (What you see is what you get o lo que ve es lo que obtiene) para editar estos recursos.

IV.3.2 Gestión de Memoria.

Anteriormente el límite de 640 KB de la memoria convencional en MS-DOS restringía el tamaño de los programas. Se utilizaban técnicas de gestión de memoria como la memoria expandida para poder tener programas más grandes funcionando. Windows en cambio, ofrece características adicionales de gestión de memoria, lo cual hace que la memoria generalmente no se un problema en las aplicaciones windows, entre las cuales se incluye una técnica de swapp a disco la cual hace que prácticamente la única limitante de memoria de windows sea el espacio en disco. Cabe mencionar que en las aplicaciones de Windows, esta característica puede afectar el performance de la misma si se tienen que hacer demasiados swapping a disco, ya que el acceso a un disco es mucho más lento que el acceso a memoria RAM.

IV.3.3 Objetos.

Cuando se crea una aplicación en un lenguaje de programación orientado a objetos, se trabaja precisamente con objetos, se crean objetos de formas y se dibujan objetos de control en esas formas. Así que para ello se definen las denominadas variables de objetos, de tal forma que con ellas se pueden manipular los objetos que éstas describen.

La característica más importante de las variables de objeto es que permiten crear formas adicionales dentro de la aplicación, es decir, crear copias de las formas con características independientes. Así mismo para definir estas variables de objeto existen también tipos de variable específicos para estas denominados tipos objeto, estos pueden ser **Genéricos** o **Específicos**; los primeros se refieren a uno de los muchos tipos de objeto específicos, mientras que los segundos se refieren a una sola forma de la aplicación o a un sólo control de la misma.

Dentro de los tipos de objetos genéricos se encuentran tres: las formas, los controles y las formas MDI. Estos se utilizan cuando no se sabe de antemano el tipo específico de objeto que describirá una variable dentro de la aplicación. Entre los tipos de objetos específicos existe una gran variedad, por mencionar algunos están: Checkbox, Listbox, Label, Image, Menú, Textbox, OptionButton, etc.

Similar a cualquier lenguaje de programación, las variables de objetos deben ser definidas al iniciar el procedimiento dentro del cual serán utilizadas, ya sea en lenguaje C, en Basic o en cualquier otro lenguaje orientado a objetos.

Cada objeto que se defina deberá contener una descripción detallada de todas sus características tales como tamaño y ubicación, contenido, función, etc.

IV.3.4 Controles

Existen varios tipos de controles utilizados en las aplicaciones Windows, entre ellas se encuentran los checkboxes, los botones, los scrollbars, etc. Un control es un módulo autónomo que funciona similar a una clase en C++, hay controles que aceptan una entrada de datos, mientras que otros solamente despliegan una

salida visual. Los controles permiten una extensa participación con el usuario sin necesidad de que participe la aplicación misma.

Toda la manipulación de estos controles se realiza mediante el uso de mensajes de notificación y funciones que se deben codificar de forma manual y que posteriormente son asignadas al control. Para cada control se deben definir una serie de propiedades que definen su aspecto, función y relación con otros objetos de la aplicación.

Cuando se realiza un evento sobre un control, éste envía un mensaje de notificación a un díálogo, que, a su vez, dispara otra serie de eventos de acuerdo al control y al mensaje que se haya generado.
Los botones de comando permiten al usuario ejecutar una acción con simplemente seleccionarlos. Cada vez que el usuario presiona el botón de control, se invoca al procedimiento asociado con ese botón. Se puede seleccionar el botón con el mouse, con <Enter>, con <Alt> y la letra subrayada del botón. Existe un tipo especial de botones llamado controles de imagen, los cuales son pequeños botones que en lugar de un texto, contienen una imagen que describe la acción que ejecutan al presionarlo.

Para el despliegue de texto se utilizan varios tipos de controles, entre ellos las etiquetas y las cajas de texto, la principal diferencia entre éstas es que las etiquetas no permiten que el usuario ingrese ningún texto, simplemente se limitan a presentarla.

Por su parte, las cajas de texto, son muy versátiles, ya que permiten que el usuario ingrese información o también se pueden utilizar simplemente para desplegar alguna información. Recordemos que usualmente se utilizan para hacer modificaciones a algún texto, así que si no se desea que el usuario modifique el texto, se deberán utilizar las etiquetas.

Existen otros controles que presentan opciones para el usuario, éstos pueden ser en la forma de botones de opción check boxes, list entries o de menús de barras, para que seleccionen un valor. Los check boxes se utilizan para encender o apagar opciones, es decir, darles un valor de verdadero o falso.

Los botones de opción dan a escoger entre una serie de opciones, de las cuales se puede seleccionar sólo una de ellas, por ello los botones de opción siempre

deben venir en grupos. El seleccionar un botón de opción, automáticamente limpia cualquier otro botón que haya sido seleccionado previamente.

Las cajas de listados presentan una lista de opciones para el usuario, por omisión, éstas son desplegadas en forma vertical en una sola columna, aunque se pueden configurar múltiples columnas también. Si el número de elementos excede los que pueden ser desplegados en la lista de opciones, automáticamente aparecerán unas scroll bars en el control. Así el usuario puede ver hacia arriba o hacia abajo de la lista.

Estos diferentes objetos pueden ser combinados entre sí para crear controles más complejos pero al mismo tiempo más poderosos y versátiles. Así mismo, se pueden crear arreglos de controles de tal forma que la variedad de arreglos está limitada por la misma imaginación del programador.

IV.3.5 Menús

Un menú es el elemento familiar de una aplicación que consiste en una lista horizontal de elementos de nivel superior, que está asociada con menús emergentes (pop up) que aparecen cuando el usuario selecciona un elemento de nivel superior. La mayoría de las veces se define un recurso para un menú por defecto para una ventana marco que se carga cuando se crea la ventana. También se puede definir un recurso de menú independientemente de una ventana marco. En ese caso, el programa debe encargarse de llamar a las funciones que son necesarias para cargar y activar el menú.

El recurso de un menú define completamente su aspecto inicial. Los elementos de un menú pueden estar atenuados o tener marcas de comprobación, y para separar grupos de elementos de menú se pueden usar barras. También es posible tener múltiples emergentes.

Adicionalmente a los menús, se pueden asignar a cada elemento del menú una tecla aceleradora, esto es que se puede llamar a ese elemento del menú sin necesidad de entrar al mismo, simplemente con presionar la tecla aceleradora que se le asignó, normalmente las teclas aceleradoras son combinaciones de las teclas <shift>,<ctrl>,<alt> y alguna letra del teclado o alguna tecla de función.

Las opciones de los menús deberán ser agrupadas por categorías o por funciones que desempeñan, algunas de ellas ejecutan acciones directamente, mientras que otras despliegan cajas de diálogo. Estas son ventanas que requieren que el usuario proporcione información requerida por la aplicación para desempeñar una acción.

A los menús se les puede mejorar asignándoles teclas de función para ciertas opciones de los mismos de tal forma que la ejecución de alguna opción de uso frecuente pueda ser realizada sin tener que entrar al menú mismo. Así mismo, se puede controlar el acceso a ciertas opciones de acuerdo a la ubicación dentro del sistema en el momento de llamar al menú o de acuerdo al nivel de acceso del usuario actual.

IV.3.6 Cajas de Diálogo, Barras de Herramientas y Barras de Estado.

Cajas de Diálogo

Las cajas de diálogo se utilizan en las aplicaciones de Windows para pedir captura de información al usuario de tal forma que la aplicación pueda proseguir o para desplegar información al mismo. Existen dos tipos de cajas de diálogo, modales y no modales, las modales deben ser cerradas antes de que prosiga la aplicación, es decir, siempre pedirán que se de un "OK" o un "Cancelar" para que se cierre la ventana y se continúe con la siguiente forma o caja de diálogo. Por su parte las no modales permiten que se cambie entre diferentes cajas de diálogo sin tener que cerrarlas.

Barras de Herramientas y Barras de Estado

Existen en Windows unos objetos denominados barras de control, donde se despliegan una serie de botones y opciones que permiten manipular las aplicaciones. Un objeto barra de herramientas es una ventana que consiste en una cantidad de botones gráficos dispuestos de forma horizontal que también pueden estar reunidos en grupos. La interface de programación determina el agrupamiento. Las imágenes gráficas de los botones se guardan en un solo mapa de bits que se añade al archivo de recursos de la aplicación. Cuando se pulsa sobre los botones, envían mensajes de órdenes de la misma forma que los menús y los aceleradores de teclado. Los controladores de mensajes de órdenes

de actualización se usan para actualizar los estados de los botones, cuyos estados son utilizados a su vez por el Marco de la Aplicación para modificar las imágenes gráficas de los botones.

La ventana de la barra de estado no acepta entradas del usuario, ni tampoco genera mensajes de órdenes. Su tarea consiste sencillamente en visualizar texto en divisiones de ventana bajo el control del programa. La barra de estado soporta dos tipos de divisiones de texto, una división de linea de mensajes y una división de indicación de estado. Para usar la barra de estado para datos que son específicos a una aplicación, en primer lugar hay que desactivar la barra de estado estándar que visualiza el indicador de menú y el estado del teclado.

IV.3.7 Aplicaciones MDI (Multiple-Document Interface).

El MDI (Multiple-Document Interface) permite crear una aplicación que mantenga múltiples formas dentro de una sola forma denominada container. Aplicaciones tales como el Program Manager de Windows, Excel, Word , etc., tienen MDI dentro de su programación.

Una aplicación MDI permite al usuario desplegar múltiples documentos al mismo tiempo, con cada documento desplegado en su propia ventana. Las ventanas de documento están contenidas a su vez en una ventana padre, que provee un espacio de trabajo para todas las ventanas de documento de la aplicación.

Para crear una aplicación MDI, primero se debe crear la forma padre y a partir de ella se van generando las formas hijas, cuando éstas son diseñadas no deben ser restringidas a un área específica dentro de la forma MDI, así que pueden definirse sus propiedades, diseñar características y escribir código de las formas hijas en cualquier parte de la pantalla.

Debemos recordar que las formas hijas siempre serán desplegadas dentro del área de trabajo de la forma padre. Por otra parte, cuando una forma hija es minimizada, aparece su icono sobre la forma MDI, en lugar de en el área general de trabajo. Por otra parte, cuando una forma MDI es minimizada, todas las formas hijas que se encuentren contenidas en esa forma serán representadas por un solo icono.

De igual manera, se pueden definir menús para una forma MDI y para sus formas hijas, de tal manera que los menús serán siempre desplegados en la forma MDI tanto para la forma padre como para las formas hijas. También es posible definir barras de herramientas, las cuales son representaciones gráficas de ciertos comandos, representadas por un botón de control con una imagen dentro de él que describe la acción que será realizada al presionar dicho botón y se han covertido en un estándar dentro de las aplicaciones en ambientes Windows, ya que proveen rápido acceso a las opciones del menú más frecuentemente utilizadas dentro de una aplicación.

IV.3.8 Intercambio Dinámico de Datos (DDE).

Las aplicaciones de Windows corren en un ambiente llamado multitarea, debido a esto, puede ser que otras aplicaciones estén siendo ejecutadas simultáneamente a nuestra aplicación y alguna de esa información sea requerida para la ejecución de nuestra aplicación, para ello, existe el intercambio dinámico de datos (DDE), el cual se encarga de extraer datos de otras aplicaciones, actualizarlas automáticamente con datos nuevos e incluso enviar comandos para manipularlas a control remoto.

El intercambio dinámico de datos es un mecanismo soportado por el ambiente operativo de Windows que permite que dos aplicaciones "platiquen" entre si mediante el intercambio de datos continuo y automático. El DDE automatiza el corte y pegado manual de datos entre aplicaciones, brindando un vehículo rápido para actualizar información.

Para intercambiar información dos aplicaciones, entablan una conversación DDE, esto es similar a una conversación entre dos personas, la aplicación que inicia la conversación es denominada la aplicación destino, o simplemente el destino, la aplicación que responde al destino es la aplicación fuente o simplemente la fuente. Una aplicación puede entablar varias conversaciones al mismo tiempo, actuando como destino en algunos casos y como fuente en otros. No existe nada especial en una aplicación que la convierta en destino o fuente, simplemente es el papel que ésta adopte en la conversación.

De igual manera, se pueden definir menús para una forma MDI y para sus formas hijas, de tal manera que los menús serán siempre desplegados en la forma MDI tanto para la forma padre como para las formas hijas. También es posible definir barras de herramientas, las cuales son representaciones gráficas de ciertos comandos, representadas por un botón de control con una imagen dentro de él que describe la acción que será realizada al presionar dicho botón y se han covertido en un estándar dentro de las aplicaciones en ambientes Windows, ya que proveen rápido acceso a las opciones del menú más frecuentemente utilizadas dentro de una aplicación.

IV.3.8 Intercambio Dinámico de Datos (DDE).

Las aplicaciones de Windows corren en un ambiente llamado multitarea, debido a esto, puede ser que otras aplicaciones estén siendo ejecutadas simultáneamente a nuestra aplicación y alguna de esa información sea requerida para la ejecución de nuestra aplicación, para ello, existe el intercambio dinámico de datos (DDE), el cual se encarga de extraer datos de otras aplicaciones, actualizarlas automáticamente con datos nuevos e incluso enviar comandos para manipularlas a control remoto.

El intercambio dinámico de datos es un mecanismo soportado por el ambiente operativo de Windows que permite que dos aplicaciones "platiquen" entre si mediante el intercambio de datos continuo y automático. El DDE automatiza el corte y pegado manual de datos entre aplicaciones, brindando un vehículo rápido para actualizar información.

Para intercambiar información dos aplicaciones, entablan una conversación DDE, esto es similar a una conversación entre dos personas, la aplicación que inicia la conversación es denominada la aplicación destino, o simplemente el destino, la aplicación que responde al destino es la aplicación fuente o simplemente la fuente. Una aplicación puede entablar varias conversaciones al mismo tiempo, actuando como destino en algunos casos y como fuente en otros. No existe nada especial en una aplicación que la convierta en destino o fuente, simplemente es el papel que ésta adopte en la conversación.

Cuando una aplicación inicia una conversación DDE, debe especificar dos cosas: el nombre de la aplicación fuente con la que desea platicar y el tema de conversación denominado tópico. Cuando una aplicación fuente recibe una solicitud de una conversación con relación a un tópico que reconoce, ésta responde y se inicia una conversación. Una vez que se estableció la conversación, no puede cambiar de tópico o de aplicaciones. La combinación de tópico y aplicaciones identifica en forma única a la conversación y permanece constante mientras dure la conversación. Si cualquiera de los dos, la fuente o el destino, cambian la aplicación o el tópico, la conversación será finalizada.
Durante la conversación, tanto el destino como la fuente, pueden intercambiar información relativa a uno o más puntos, Los puntos son referencias a datos que son significativos para ambas aplicaciones. Ya sea el destino o la fuente pueden cambiar el punto sin afectar el estado de la conversación.

IV.3.9 Object Linking and Embedding (OLE).

El sistema operativo Windows ha ido evolucionando a lo largo de los años. La incrustación de Objetos (OLE) permite mejorar los programas combinando objetos creados en distintas aplicaciones. Por ejemplo, se pueden incluir gráficos, sonidos, dibujos y otros, en un solo documento. Cuando el usuario activa un objeto OLE, se ejecuta el programa Windows que creó dicho objeto, con lo cual el usuario puede manipularlo. La programación OLE con herramientas como SDK de windows era extremadamente difícil, pero las nuevas herramientas como Microsoft Foundation Class Library, se simplifica enormemente esta tarea.

Básicamente OLE es un conjunto de protocolos que permiten a los programas de Windows cooperar unos con otros, es un tipo de bloque de construcción para crear aplicaciones. Hay que tener presente que en esta sección sólo se abarcarán principios básicos del OLE

En OLE existen dos tipos de programas sensibles al mismo, los clientes y los servidores. Los servidores OLE generan elementos OLE, los clientes OLE incrustan estos elementos dentro de sus documentos. Algunos programas pueden realizar ambas funciones, pero son casos especiales. Un documento

cliente puede contener uno o más elementos del servidor. El servidor OLE debe poder prestar soporte a múltiples clientes.

Hay dos formas de agregar objetos a un documento cliente, insertar o pegar, mientras que, al insertar un objeto, éste se contiene en su totalidad dentro del documento cliente, cuando se pegan, se utilizan los datos que están almacenados en un archivo en disco, de tal forma que el nombre del archivo queda registrado dentro del documento cliente. Esta segunda opción proporciona varias ventajas con respecto a un elemento incrustado, ya que el tamaño del documento cliente se reduce y además se elimina la redundancia de datos.

Por regla general los enlaces OLE son creados automáticos, esto es que se actualizarán los elementos asignados cada vez que se cargue el documento cliente, sin embargo puede establecerse también una actualización manual si así se desea.

Windows maneja un archivo donde registra todos los servidores OLE que se encuentran en el sistema, de esta forma si un cliente desea utilizar un servidor OLE, primero buscará en esta lista si está contenido dentro de los disponibles y lo buscará en la ruta que ahí se tenga indicada.
La aplicación cliente y la aplicación servidor son programas separados del sistema operativo Windows, sin embargo deben interactuar de formas determinadas y específicas. Se debe entonces concluir que el cliente debe tener acceso a suficientes datos del servidor como para poder interpretar su contenido y poder integrarlo a la ventana del cliente, además de guardar la información del objeto, de tal forma que después pueda ser recuperada y vuelta a editar. Esto se tiene que hacer de tal forma que el servidor no tenga que estar presente cada vez que se repite el elemento, para ello existen dos formatos que se utilizan conjuntamente, el formato primitivo que es el de escritura directa en un archivo del disco y el formato de presentación de windows que es un metafile con una presentación visual de los datos subyacentes.

IV.3.10 Bibliotecas de Enlace Dinámico (DLLs).

En el entorno MS-DOS todos los módulos objeto de un programa quedaban enlazados estáticamente durante el proceso de construcción. Windows permite

un enlace dinámico lo cual significa que unas bibliotecas especialmente construidas pueden ser cargadas y enlazadas en tiempo de ejecución. Múltiples aplicaciones pueden compartir bibliotecas de enlace dinámico (DLL), con lo cual se ahorra memoria y espacio de disco. El enlace dinámico incrementa la modularidad de un programa porque se pueden compilar y comprobar las DLL de forma separada. Originalmente se crearon DLL´s para ser utilizadas en lenguaje C, y el C++ ha añadido algunas complicaciones. Los desarrolladores de la biblioteca de clases "MicroSoft Foundation Class Library" lograron combinar todas las clases del marco de la aplicación en una sola DLL. De esta forma en una aplicación se pueden enlazar las clases estáticas y dinámicas.

IV.4 Herramientas de desarrollo.

El mercado de las herramientas desarrollo de aplicaciones Cliente/Servidor es sumamente competitivo, conjuntando un universo muy variado de productos y proveedores altamente reconocidos. La mayoría de estos productos están basados en 4GLs propietarios, muchos de los cuales tienen extensiones orientadas a objetos. Estos pueden ser basados en objetos o totalmente orientados a objetos. SQLWindows y PowerBuilder caen dentro de las herramientas orientadas a objetos, mientras que otras como VisualBasic Visual C++ caen dentro de las herramientas basadas en objetos. Adicionalmente los proveedores de bases de datos como Oracle y Sybase proveen herramientas propias sumamente competitivas, aunque sus productos generalmente están diseñados para su propia base de datos.

En el mercado de windows, la mayoría de la competencia ha sido de las herramientas independientes entre las que están:

PowerBuilder de Powersoft.
SQLWIndows de Gupta.
ObjectView de KnowledgeWare.
Uniface Six de Uniface.
VisualBasic de Microsoft.
VisualWorks de ParcPlace Systems.
VisualAge de IBM.

Al seleccionar una herramienta de desarrollo cliente/servidor, hay que realizar algunas decisiones básicas que van de lo general a lo particular. Generalmente se comienza por eliminar a los proveedores de bases de datos, ya que la gente busca mayor poder de desarrollo y las herramientas que ellos proveen no cumplen con esta característica. Después se quitarán las herramientas que se basan en SmallTalk, como VisualAge y Visualworks a menos que se desee ser un purista de la programación orientada a objetos.

Una vez que se tomen estas decisiones, la gama de productos es mucho menor, La mayoría de las veces SQLWindows y PowerBuilder terminan compitiendo entre ellos. Aunque Object view es una herramienta buena, bastante capaz, no tiene más que ofrecer que PowerBuilder y SQLWindows, además de que es mucho menos popular que estas dos. Visual Basic se utiliza mucho por su facilidad de uso y generalmente se utiliza para crear aplicaciones de bases de datos sencillas, pero también se puede utilizar cuando se desee intercomunicar con otros productos de Microsoft como el Office.

Por lo tanto el único otro que puede realmente competir con los grandes gigantes es Uniface Six, particularmente si el cliente busca portabilidad entre plataformas.

Las aplicaciones desarrolladas en SQLWIndows, PowerBuilder y Object view consisten en GUI's con cierta lógica de negocios integrada que corre en estaciones de trabajo clientes. Estas son herramientas de desarrollo cliente/servidor automáticamente generan gran parte de la interface de usuario y el acceso a la base de datos y utilizan SQL como interface con servidores de bases de datos. Esto puede tener como consecuencia un mal comportamiento del sistema, así como implicaciones en el mantenimiento de algunas aplicaciones, especialmente las empresariales.

El ambiente de desarrollo de estas herramientas es totalmente orientado a Windows, incluyendo generación automática de código de SQL y muchas opciones de conectividad a diferentes bases de datos.

A continuación analizaremos brevemente algunas de estas herramientas de desarrollo orientadas a objetos.

IV.4.1 Visual Basic (Microsoft Corp.)

Visual Basic es una herramienta que permite crear rápida y fácilmente aplicaciones para el ambiente Windows de Microsoft. El sistema de programación de Visual Basic permite crear aplicaciones atractivas al usuario y muy útiles, que además exploten la interface gráfica de usuario (GUI- Graphical User Interface).

Como Visual Basic provee herramientas adecuadas para los diferentes aspectos del desarrollo de GUI's, permite hacer al desarrollador mucho más productivo. Se puede crear la interface gráfica del usuario para la aplicación simplemente dibujando objetos en la pantalla de una forma gráfica. Basta con definir las propiedades de estos objetos para afinar la presentación y el comportamiento de cada objeto. Posteriormente se hace que esta interface reaccione al usuario escribiendo código que responda a los eventos que ocurran en la interface.
Utilizando Visual Basic, se pueden crear aplicaciones completas y poderosas que exploten las características de Microsoft Windows, incluyendo la Interface de Múltiples Documentos (MDI), Object Linking and embedding (OLE), Dynamic Data Exchange (DDE), gráficas, etc. Adicionalmente, se puede extender a Visual Basic mediante la implementación de controles personalizados, así como llamando a procedimientos mediante las Librerías de Enlace Dinámico (DLLs).

Otra característica importante de Visual Basic, es que se obtiene como resultado final un ejecutable que utiliza una DLL para el runtime y que puede ser distribuída junto con el software desarrollado.

Para crear una aplicación sencilla puede tomar incluso sólo algunos minutos. Primeramente debe crearse la interface de usuario "dibujando" los controles, tales como las Cajas de Texto y los Botones de Comando en una forma. Después se definen las propiedades para la forma y controles, con el fin de especificar valores como colores, tipos de letra, tamaños, etc. Finalmente, se escribe el código para dar vida a la aplicación. Los pasos básicos que se toman para crear una aplicación sencilla, muestran los principios que se utilizarán cada vez que se escriba una aplicación, sin importar la complejidad de ésta.

El ambiente de desarrollo de Visual Basic contiene una serie de herramientas y utilerías que permiten hacer el desarrollo más cómodamente. La interface de Visual Basic consiste de los siguientes elementos:

Barra de Herramientas.- Provee acceso rápido a comandos comúnmente utilizados en el ambiente de programación. Basta con seleccionar un ícono en la barra para ejecutar la acción representada por dicho ícono.

Caja de Herramientas.- Provee una serie de herramientas que se utilizan mientras se diseña para colocar controles en una forma, entre los que están los check boxes, combo boxes, botones de comando, list boxes, etiquetas, menús, etc.

Barra del Menú.- Despliega los comandos que se usan para construir la aplicación.

Forma.- Sirve como una ventana que se customiza como la interface de la aplicación. A ésta se agregan los controles, gráficas e imágenes para crear la presentación que se desea tenga la aplicación.

Ventana de Proyecto.- Lista la forma, módulos de código y archivos de controles personalizados que conforman el proyecto actual. Un proyecto es la colección de archivos que se usan para construir una aplicación.

Ventana de Propiedades.- Lista las propiedades para la forma o control seleccionado actualmente. Una propiedad es el valor de un objeto, tal como tamaño, tipo de letra o color.

Para la creación de la interface, se deben definir los objetos que la conformarán, Esto se puede hacer tomando cada objeto que se desee agregar de la caja de herramientas y colocándolo en la posición deseada dentro de la forma.

Una vez que se completó esta etapa y se tienen agregados todos los objetos que serán utilizados en la forma, se deben definir las propiedades de cada objeto creado.

La ventana de propiedades permite realizar esta labor de una forma rápida y sencilla, ésta consiste de tres elementos:

La caja del objeto, que despliega el nombre del objeto para el cual se definirán sus propiedades.

La caja de definiciones, que permite editar la definición de la propiedad del objeto seleccionada en ese momento dentro de la lista de propiedades. Algunas definiciones se pueden seleccionar de entre una lista de opciones, mientras que otras serán definidas por el usuario.

La lista de propiedades, que en su parte izquierda despliega todas las propiedades del objeto seleccionado y en la parte derecha despliega las definiciones actuales para cada propiedad.

Terminada la segunda etapa, se debe entonces escribir el código, dentro de la ventana de código. Esta ventana está compuesta de instrucciones en el lenguaje de Visual Basic, que es muy similar al lenguaje original Basic, sólo que es estructurado y contiene instrucciones adicionales para manejo de objetos, mensajes y comunicaciones. Mediante el uso de esta ventana es que se edita el código de toda la aplicación. Para escribir código se debe dividir en procedimientos. El procedimiento de un evento contiene código que se ejecuta cuando ocurre un evento tal como la selección de un botón.

Una vez terminadas estas tres etapas, solamente queda pendiente ejecutar la aplicación, lo cual se realiza seleccionando la opción "Iniciar" en el menú "Ejecutar" de la Barra del Menú. Una vez terminado el proyecto se puede grabar como una ejecutable y simplemente agregarse como un elemento más dentro del ambiente de Windows.

Como se puede apreciar la ventaja de Visual Basic es su facilidad de uso ya que casi todo el mundo conoce el lenguaje Basic y su interface es tan amigable que permite, sin ser un experto programador, crear rápidamente aplicaciones que funcionan. Además las nuevas versiones incluyen acceso a las bases de datos más comunes, así como importación de rutinas en otros lenguajes, lo cual le da un poder adicional que fácilmente le permite competir con las más sofisticadas herramientas de desarrollo para Windows.

IV.4.2 Visual C++ (Microsoft Corp.)

Visual C++ está constituido por dos sistemas completos de desarrollo de aplicaciones para Windows en un sólo producto. Aquellos que estén interesados, pueden desarrollar programas en lenguaje C para Windows, usando

la API que se introdujo originalmente en el SDK de Windows. Las técnicas de programación en SDK para Windows son bastante conocidas, y han sido documentadas en muchos libros. Se pueden utilizar muchas herramientas introducidas recientemente en Visual C++ para hacer que la programación para Windows en SDK sea más fácil.

IV.4.3 SQLWINDOWS (Gupta Corp.)

SQLWindows es una poderosa herramienta de desarrollo de aplicaciones cliente/servidor, la cual está basada en Microsoft Windows y totalmente orientada a objetos. Al liberar sus tecnología Quick Objects, Gupta llevó SQLWindows a convertirse en una herramienta de desarrollo visual que es mucho más fácil de aprender y usar para la creación de sofisticadas aplicaciones que accesen bases de datos SQL, bases no SQL como Dbase, Lotus Notes y sistemas de E-mail con Quick Objects preconstruidos. Estos objetos pueden ser modificados y los desarrolladores pueden también crear los suyos propios. Team Windows y Open Repository permiten que equipos de programadores trabajen en forma cooperativa. Soportan el desarrollo de grandes aplicaciones empresariales con un fuerte control de los proyectos y reutilización de código. El SQLWindows Application Language es capaz de acompletar la mayoría de las tareas de programación. El compilador convierte las funciones internas de SQLWindows en código de C y luego compila las sentencias en DLLs.

Contiene soporte para la creación de aplicaciones basadas en Multiple Document Interface (MDI). También tiene un 4GL que se basa en SQL y que apoya a los desarrolladores en la creación de aplicaciones gráficas para accesar servidores de SQL. Se le pueden agregar además funciones externas de C o en cualquier otro lenguaje que pueda ser compilado como una DLL. Otra de las ventajas que tiene esta herramienta es que las aplicaciones que se desarrollan se pueden migrar a un ambiente de red sin necesidad de realizar cambios en la programación.

SQLWindows incluye una serie de herramientas que lo hacen sumamente poderoso y versátil tales como Quick Objects, Compilador de C y un Open Repository que le permite interfasar fácilmente con bases de datos tales como Oracle, Sybase y SQLServer, además de su base de datos propietaria SQLBase.

Quick Objects son objetos preconstruidos y reutilizables que ayudan a crear aplicaciones más rápido sin necesidad de programar. Estos objetos contienen código de SQLWindows. Un Quick Object es una clase que el desarrollador configura en el momento de diseñar la aplicación mediante una interface gráfica. Existen otras utilerías como Quick Forms, Quick Graphs y Quick Menús que no son mas que un tipo de Quick Objects.

User Interface es un diseñador de SQLWindows que crea aplicaciones MDI y que forman el marco de la aplicación, en ésta se contienen los componentes básicos de la aplicación. Este marco puede ser modificado y expandido.

Table Windows es una herramienta para editar y actualizar datos tabulares. Se utiliza para desplegar resultados de queries, actualizar, localizar y agregar datos.

Report Windows se usa para definir la apariencia de los reportes. Esta herramienta solicita y recibe datos de los programas de SQLWindows, formatea los datos, realiza los cálculos necesarios y luego crea el reporte.

Team Windows permite a los grupos de programadores crear grandes aplicaciones cliente/servidor. Automatiza las tareas administrativas de control de proyectos tales como staffing, control de seguridad y prueba los ambientes de producción. Puede mantener registro de todos los componentes de la aplicación y puede mantener control sobre el código fuente en varios niveles.

SQLNetwork y SQLGateways se refieren a la conectividad bajo windows proveida, así como conectividad a ODBC y SQLRouters. SQLNetwork provee conectividad nativa a Oracle, Sybase, SQL Server, Informix, DB2 y usa el software de comunicaciones de los proveedores para accesar a los servidores de la red, mientras que SQLGateways corre sobre OS/2 y Netware.

IV.4.4 PowerBuilder (PowerSoft Corp.)

PowerBuilder es una herramienta poderosa que permite desarrollar fácilmente aplicaciones cliente/servidor en Windows, PowerBuilder ha ganado una reputación por ser una herramienta altamente capaz, que ofrece un rico complemento de facilidades con relativa facilidad de uso y un significante grado de orientación a objetos. A pesar de que competidores como SQLWindows

tienen mayor orientación a objetos, Quick Objects, etc., PowerBuilder ha logrado entender mejor al mercado y jugar con su imaginación. Entre las características de PowerBuilder se encuentran la conectividad nativa a Oracle, Sybase e Informix, interfaces con terceros y un generador de clases de C++, una herramienta llamada User Object Painter, así como una utilería para migración y replicación de datos.

La interface de PowerBuilder consiste en una serie de painter GUI para trabajar con objetos tales como bases de datos, aplicaciones, menús y ventanas. Este ambiente modular es un excelente marco para las características de programación orientada a objetos de PowerBuilder. Encapsulación, herencia y polimorfismo son todos soportados en esta herramienta. En el corazón de PowerBuilder está el repositorio central de objetos y atributos que pueden ser compartidos por los programadores para definir el aspecto y el comportamiento de nuevos objetos.
Con el painter de Datawindows se pueden crear gráficamente ventanas que reflejan la estructura de la base de datos. Estas contienen funciones interconstruidas que permiten el acceso y actualización de las bases de datos, así como también interactuar con el usuario.

El Library Painter, la pantalla en la cual se administran todos los objetos de una aplicación, ayuda a integrar PowerBuilder con otros productos, el cual ofrece un API abierto para la integración de productos de terceros como las herramientas CASE. Los objetos se pueden exportar desde y hacia archivos de texto.

PowerBuilder utiliza las plantillas o formas llamadas Quick Style, las cuales no sólo ofrecen una serie de formatos ya predefinidos, sino que además guían al usuario mediante un proceso completamente funcional. Esta herramienta poderosa simplifica la creación de formularios complejos a nivel maestro y de detalle, permitiendo a los usuarios la construcción rápida de plantillas para la entrada de datos.

IV.4.5 Uniface Six (Uniface)

Uniface Six está preparado para competir con PowerBuilder y SQLWindows, las dos herramientas basadas en Windows más importantes del mercado. Es un software con raíces en los ambientes de desarrollo de segunda generación de

Unix, y que actualmente tiene un importante historial como herramienta de desarrollo 4GL.
Uniface Six tiene los elementos gráficos y orientados a objetos requeridos por una herramienta poderosa de desarrollo. Contiene además el Universal Presentation Interface (UPI) que permite que una aplicación, una vez construida, pueda ser portada a una variedad de GUI's o interfaces orientadas a caracter, sin necesidad de programar nada más.

A través de su manejo orientado al modelo, Uniface Six canaliza a los desarrolladores a concentrarse en los modelos de información de la aplicación y la compañía. Aunque Uniface Six puede tomar más trabajo preliminar que otros 4gls, su metodología de definición de objetos conduce a una reutilización efectiva de los mismos, ya que utiliza una arquitectura basada en componentes que está diseñada para la creación de módulos integrados que compartan información.

La gran ventaja de Uniface Six es que puede correr y generar código para OS/2, Motif, Macintosh, Windows e interfaces basadas en caracter. Además el desarrollo de las aplicaciones es independiente del DBMS que tenga como backend, así como del hardware, el sistema operativo, el GUI o el tipo de red con que se cuente.

Capítulo V. Aplicación (SmauxW).

Introducción.

El presente capítulo está enfocado a aquellos desarrolladores en ambientes Windows y Unix, interesados en conocer el como llevar a cabo un sistema basado en sockets Berkeley-Windows Sockets, utilizando como herramientas Visual Basic y lenguaje "C" para Unix. Así mismo, será de gran utilidad para aquellas personas interesadas en conocer un sistema de administración de máquinas Unix basado en una PC.

El desarrollo del SmauxW (Sistema de Administración y Monitoreo de Nodos UNIX) se presenta como una herramienta más al mercado para aquellas personas encargadas de administrar redes con estaciones de trabajo Unix, mismos que requieren de realizar tareas rutinarias como: Monitorear estaciones de trabajo en la red, verificar las características de la estación, visualizar sus procesos y poderlos eliminar, monitorear los File Systems, y por último el mantenimiento de usuarios.

El contar con una herramienta de este tipo facilitará algunas de las tareas que a manera de rutina vienen realizando día a día los administradores de red.

SmauxW fue desarrollado en Visual Basic ver 3.0 utilizando como medio de comunicación TCP/IP for Windows y Windows Sockets ver 1.1. Por el lado de las estaciones de trabajo, se realizó el desarrollo con lenguaje "C" en una Sparc IPC de SunMicrosystems, la cual tiene como sistema operativo SunOs ver 4.1.2 y se comunica por medio de TCP/IP utilizando sockets de Berkeley.

V.1 Esquema General.

Como ya se ha mencionado, el objetivo del SmauxW es monitorear y Administrar estaciones de trabajo Unix misma actividad que se observamos de manera gráfica en la figura 16.

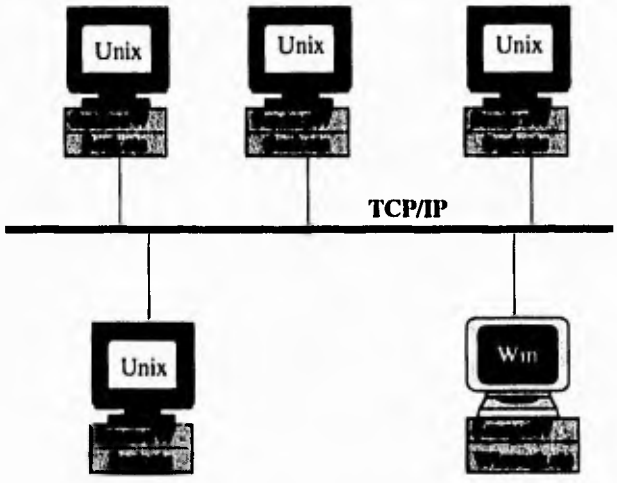


Fig. 16 . Esquema General de Monitoreo y Administración.

La aplicación consta básicamente de dos puntos: Monitoreo y Administración.

Monitor. En la parte monitoreo la aplicación es encargada de sensar las estaciones activas en la red. Para lograr esto se realiza un proceso de envío de un mensaje a las estaciones UNIX (dadas de alta en la PC), las cuales deberán de estar escuchando por un puerto (corriendo el programa), y en caso de no poder sensar la estación , la PC "NO" las mostrará dentro de la pantalla.

Administración. En este punto se parte de que se encuentra al menos una estación Unix activa y corriendo la aplicación. El administrador podrá observar las características generales de la máquina, su dirección IP, sus File Systems, Procesos y Usuarios (mismos puntos que detallaremos más adelante).

Cabe aclarar que la persona que se encuentra trabajando en la PC, también podrá trabajar con el resto de las aplicaciones montadas en la PC.

V.1.1. Requerimientos para la PC.

Antes de explicar a detalle la aplicación realizaremos algunas observaciones, normas, recomendaciones y sugerencias, las cuales se harán en base a la experiencia adquirida durante el desarrollo del sistema y durante la implantación del mismo, esto es para contar con una adecuada integración con el resto de las aplicaciones de la PC. (ver tabla 1).

Sugerencias y recomendaciones:

Hardware		Software	
Mínimos	Deseables	Mínimos	Deseables
386	486	Windows 3.1	Windows 3.11
SX	DX2	WinSockets ver 1.1	WinSockets ver 1.1
100 H.D.	240 H.D.	TCP/IP for Windows	TCP/IP for Windows
4 RAM	16 RAM		
VGA	SVGA		

Tabla 1. Requerimientos deseables vs. mínimos en PC.

V.1.2. Observaciones y normas.

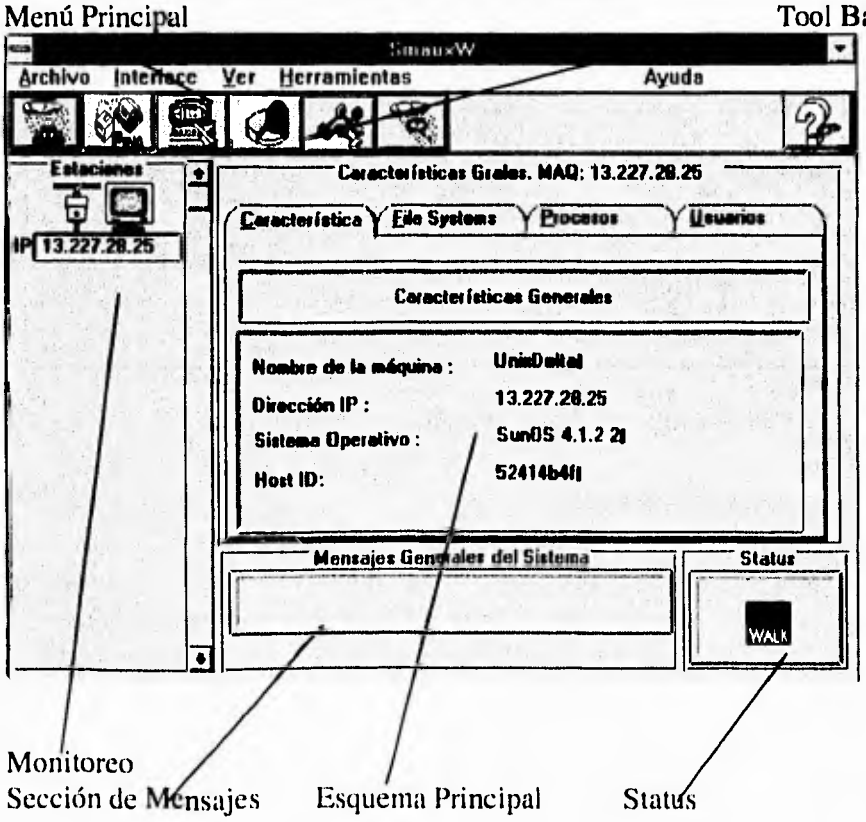
SmauxW fue desarrollado utilizando TCP/IP de Netmanage Chameleon, no obstante se realizaron pruebas con TCP/IP de diferentes proveedores sin contar con problemas. También fue desarrollado utilizando como base Windows Sockets versión 1.1, por lo que el sistema se encontrará a prueba para las siguientes versiones de WinSockets (ver 2.0).

Para poder monitorear las estaciones, estas deberán estar dadas de alta en el archivo "UHOSTS" ubicado en el subdirectorio "C:\COMUNIX", mismo que "NO" deberá de ser removido, porque podría ocasionar un mal funcionamiento de la aplicación.

La aplicación es una herramienta como tal, sin embargo, el administrador es RESPONSABLE de saber que procesos eliminar, y así mismo de saber a que usuarios dar de alta, o bien, a que usuarios dar de baja. Por lo cual **"El mal uso de este Software es completa responsabilidad del administrador".**

V.2 Diseño Funcional.

Después de saber el esquema general de la aplicación, pasaremos a conocer el diseño funcional en donde explicaremos la integración en ambos ambientes. Esto se realizará mediante las pantallas del sistema.
La pantalla principal de la aplicación es la siguiente:

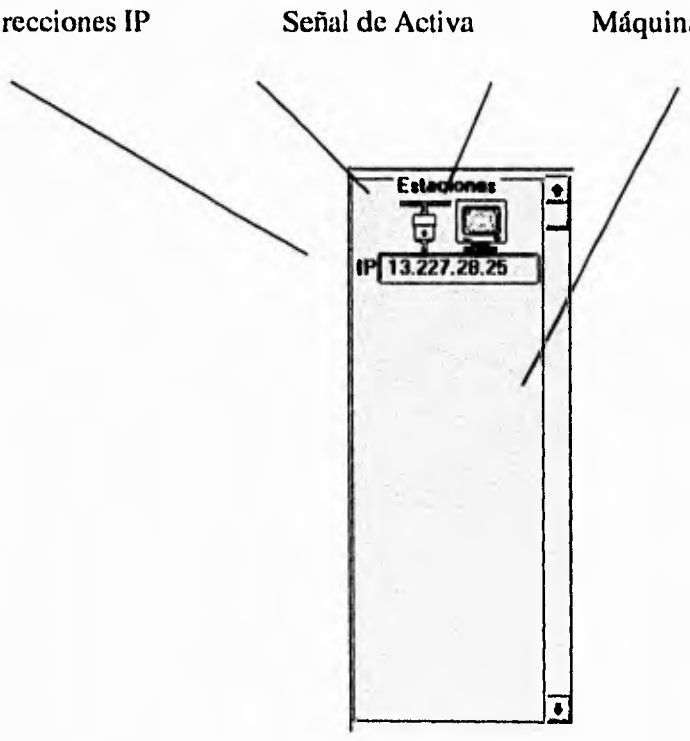


Explicaremos brevemente cada una de las pantallas , así como su funcionalidad e integración con las estaciones Unix.

V.2.1 Monitoreo

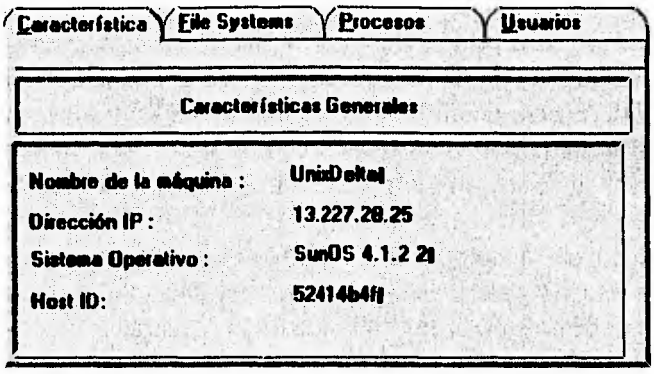
Dentro de las funciones principales de la aplicación, se encuentra el monitoreo. Esta parte es encargada de leer las máquinas que se encuentran registradas en el archivo "UHOSTS" en el subdirectorio "COMUNIX" y de enviar información para saber si las máquinas están respondiendo, de ser así, aparecerán dentro de una ventana con su dirección de IP en la parte de abajo de la máquina.

Nota. Solamente se desplegarán un máximo de 4 máquinas dentro de una ventana, pero eso no quiere decir que no se estén monitoreando más, por el contrario, se pueden observar las "n" máquinas registradas en el sistema.



V.2.2 Características.

Dentro de la parte de características podremos observar diferentes datos tales como nombre de la máquina, dirección IP, Sistema Operativo que está corriendo la máquina, y el Host ID el cual es un dato que en ocasiones representa ser de gran utilidad.

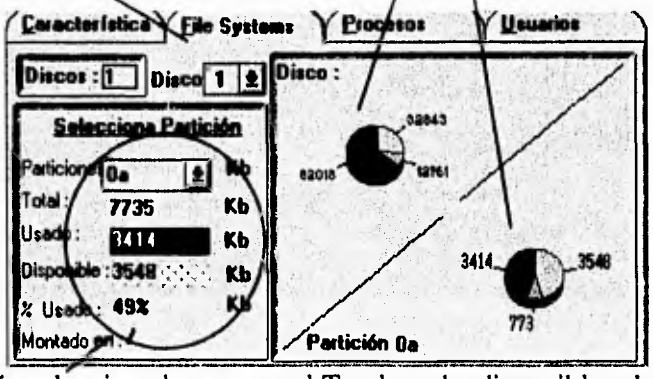


V.2.3 File Systems.

En esta sección se realiza un intercambio de información entre la máquina seleccionada y la PC monitor. Se cuenta con una opción de "Actualización de Pantalla", la cual obliga a la máquina a realizar nuevamente la petición de los valores a graficar. La importancia de esta información, radica en que el administrador estará siempre al pendiente del perfecto estado de los File Systems (espacio disponible).

Muestra el total de Discos con que cuenta la máquina

Genera gráficas del Total del Disco y de las particiones activas



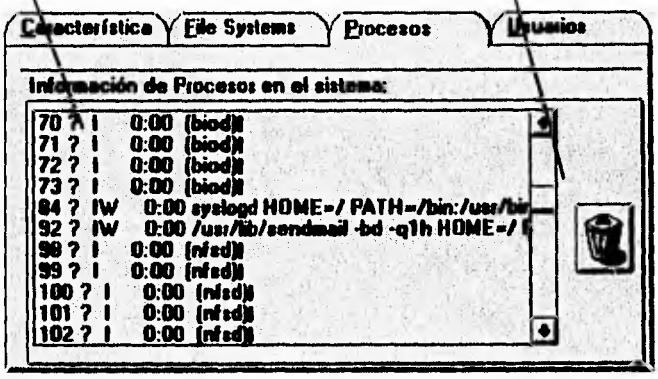
De la partición seleccionada muestra el Total usado, disponible y la capacidad.

V.2.4 Procesos.

En las secciones anteriores, se explicó la importancia del monitoreo de los procesos. El objetivo principal de esta pantalla es realizar la petición de los procesos que se encuentran corriendo en la máquina seleccionada, pudiendo seleccionar alguno y enviar la petición de "matar al proceso".

Información de procesos Activos corriendo en la máquina seleccionada.

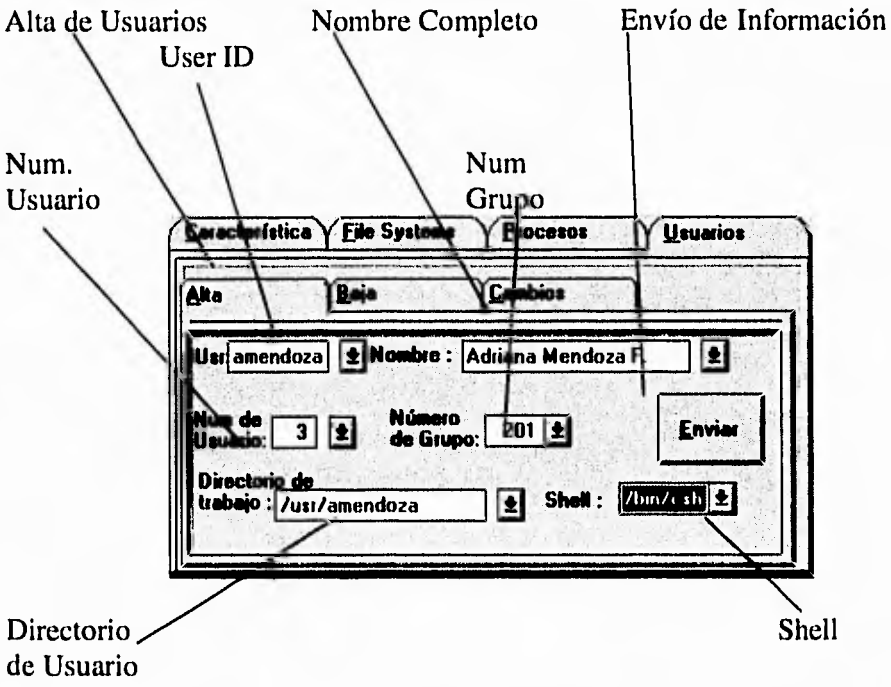
Matar Procesos



V.2.5 Usuarios.

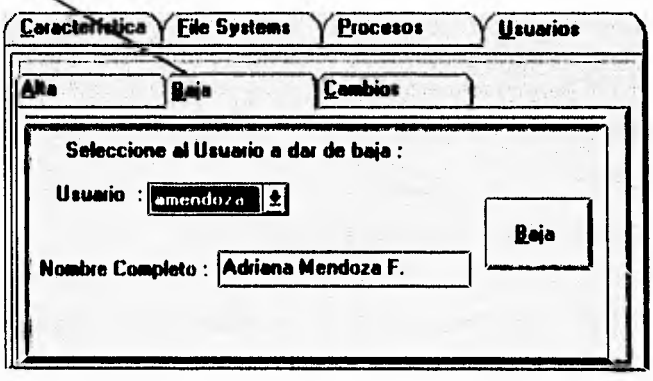
Una de las partes principales dentro de la administración de equipos de computo, es la alta, baja y cambios de usuarios. En la sección "Alta de Usuarios" se detecta los usuarios pertenecientes a la estación monitoreada, no permitiendo repetir: usuarios con el mismo nombre o número y directorios de trabajo.

En el momento en que se da un click en el Icono de Enviar, se realiza una petición a la estación seleccionada y en el momento en que se da de alta el usuario, se realiza un refresco de los usuarios automáticamente.

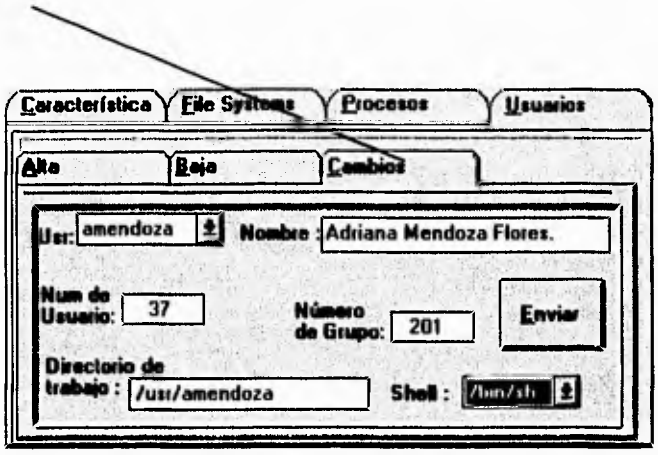


La sección de baja y cambios de Usuarios es como su nombre lo indica, seleccionar un usuario y realizar la acción deseada, el cual podemos observar en las siguientes figuras.

Baja de Usuarios:

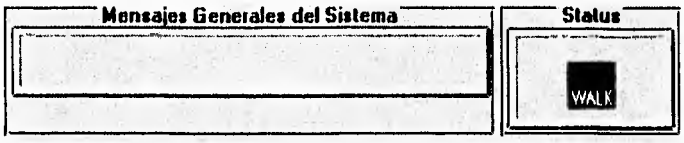


Cambio de Usuarios:



V.2.6 Mensajes y Status del sistema.

La aplicación muestra una serie de mensajes para que el administrador esté informado del funcionamiento del sistema, y del status de las máquinas a monitorear, es decir, aparecen los mensajes de comunicación, errores del sistema, mensajes de la estación, etc. La sección de status como se mencionó en un principio, es activada cada vez que ocurre un evento crítico, así mismo nos muestra la actividad del sistema.



V.2.7 Menú principal y barra de herramientas.

Dentro de la barra de herramientas tenemos diferentes iconos que serán de gran utilidad para el administrador, tales como:

 **Refresco General**. Es encargado de sensar las máquinas que se encuentran activas en la red.

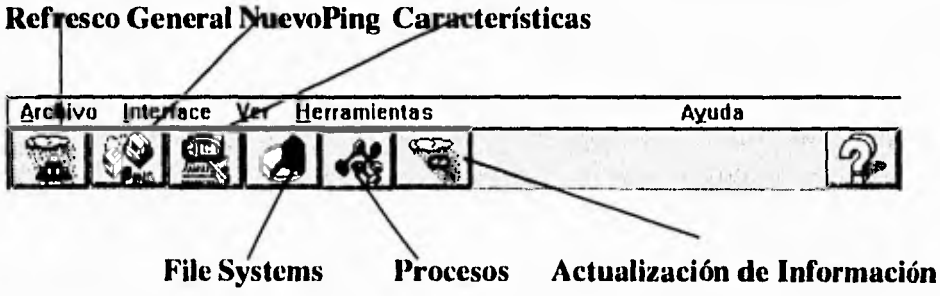
 **NuevoPing**. Es encargado de detectar si una estación en especial se encuentra en red y si se encuentra corriendo el software SmauxW Server..

 **Características**. Simplemente nos despliega las características de la estación activa.

 **File Systems**. Despliega los File Systems y Discos de la estación activa.

 **Procesos**. Muestra los procesos activos en la estación seleccionada.

Para cada una de las características se cuenta con un icono de actualización de información, el cual pide información a la estación seleccionada y actualiza los datos.



V.3 Diseño Técnico y Código.

En este punto explicaremos a grandes rasgos la integración del sistema SmauxW-Cliente (Software en Windows) / SmauxW-Server (Software corriendo en Unix) mediante Sockets.

Para explicar este punto, observaremos la figura 17.

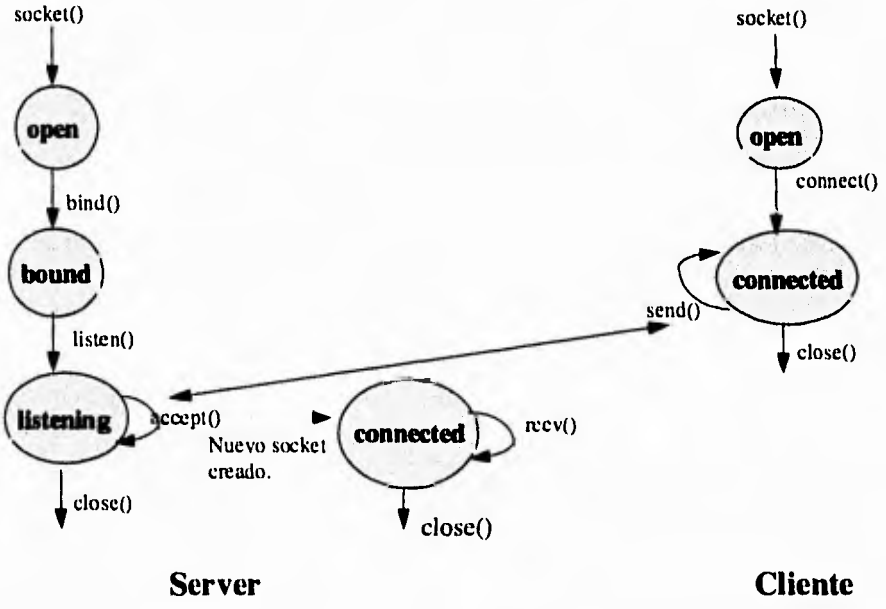


Figura 17. Comunicación Sockets BSD- Winsockets

En la figura anterior observamos el ciclo de los socket, el cual fue base para el desarrollo de SmauxW, y que mediante los cuales se logró el intercambio dinámico de información.

Como ya se explicó en el capítulo de sockets, el primer punto para realizar la comunicación es definir el socket, en donde se incluye la características del mismo con su puerto correspondiente.

V.3.1 Servidor.

Es encargado de escuchar las peticiones realizadas por parte del cliente, y así mismo, reaccionar a la información llegada.

1) Define el socket.

2) Enlace. En este punto se asocia el socket a las características definidas anteriormente, es decir: dirección, puerto familia, protocolo.
3)Escucha. Se encuentra disponible para cualquier petición de alguna máquina en la red.

4) Acepta-Conecta. En el momento en que llega una petición esta es aceptada mediante un socket "hijo", el cual permite la llegada de información.

5) Cierra. En el momento en que se desee no aceptar más entradas, se procede a cerrar el socket.

NOTA. Es indispensable cerrar el socket antes de volver a definirse, es decir, "NO" es posible crear un mismo socket si éste NO ha sido cerrado previamente.

V.3.2 Cliente.

Su misión, es realizar peticiones al servidor, para que éste devuelva la información requerida.

1) Define socket.

2) Conexión. En este punto es donde se realiza la conexión a una dirección determinada sobre un puerto específico. En caso de que la máquina destino no se encuentre en status de "escuchar", "NO" se podrá realizar la comunicación deseada.

3) Envío. En caso de ser exitosa la comunicación, se podrá enviar la información deseada.

4) Cerrar. En el momento deseado se podrá cerrar el socket.

Así entonces, la integración de este esquema C-S en la aplicación, es desde el momento en que la PC envía un mensaje a la estación, detectando si esta está activa o no (si se encuentra escuchando), y de ser así la estación regresa información de acuerdo a la solicitud por parte de la PC.

Como parte del diseño y código, mostraremos las principales funciones del sistema, así como su código.

V.3.3 Comunicación por sockets Berkeley/Winsockets v1.1

La comunicación se logra dejando en modo de espera (escuchando) a las estaciones Unix, es decir, se define un socket para escuchar y se le asigna el puerto **4**, por el cual recibe la información, mientras que en el sistema en windows se abre otro socket para envío de información desde el puerto **4**.

Lo mismo sucede para cuando el sistema Unix requiere enviar información: Se abre un socket de envío por el puerto **6**, por el cual se enviará la información, y así mismo el sistema en windows abre otro socket con el puerto **6** definido para recibir dicha información. Esto se muestra en la siguiente figura:

Definición del Socket para recepción en Unix:

Código Unix.

```
if((sd=socket(AF_INET,SOCK_STREAM,0))==-1){
     perror("Servidor:Socket");
     printf("El socket de recepción no pudo abrirse");
     exit(1);
 }
 sockname.sin_family=AF_INET;
 sockname.sin_addr.s_addr=INADDR_ANY;
   sockname.sin_port=htons(4);      <---------- Puerto 4 para recepción de información
 if(bind(sd,&sockname,sizeof(sockname))==-1){
   perror("Servidor:Bind");
```

```
  printf("La liga del socket no pudo lograse\n");
  exit(1);
 }
 if(listen(sd,1)==-1){
  perror("Servidor:Listen");
  printf("El socket de recepción no pudo escuchar del puerto\n");
  exit(1);
 }
```

Definición del socket para envío de información desde Unix:

Código Unix

```
struct sockaddr_in from,sockname;
 char buffer2[80];
  if((sd=socket(AF_INET,SOCK_STREAM,0))==-1){
    perror("Cliente:Socket");
    printf("El socket de recepción no pudo abrirse\n");
    return(0);
  }
  sockname.sin_family=AF_INET;
  sockname.sin_addr.s_addr=inet_addr(ipremota);
 sockname.sin_port=htons(6);   <-------- Puerto 6 para envío de información a Windows
  if(connect(sd,&sockname,sizeof(sockname))==-1){
    perror("Cliente:Connect");
    printf("El socket para recepción no pudo conectarse\n");
    return(0);
  }
```

Definición del socket para envío de información desde Windows:

A continuación mostraremos la rutina encargada de enviar información

Código Vb.

```
Sub envía (num_puerto As Integer, xsocket As Integer, mensaje As String,
Dir_IP As String, res As Integer)

Dim i           As Integer
Dim s_out       As sockaddr_in
Dim addr        As Long
Dim anIP            As Long
Dim aHost           As hostenttype
Dim anIPString  As String * 20

lblMensaje.Caption = "Abriendo el socket..."
xsocket = Socket(AF_INET, SOCK_STREAM, IPPROTO_IP) <--Definición
                                                          del socket.
addr = inet_addr(Trim$(Dir_IP))    <----------Dirección de envío.
s_out.sin_family = AF_INET
s_out.sin_port = htons(num_puerto)
s_out.sin_addr = addr          <----------Características del socket.
lblMensaje.Caption = "Conectando el Socket...Espere.."
i = connect(xsocket, s_out, Len(s_out))  <----------Realizamos la conexión.
Si el valor es diferente de "0" existe un error
If i <> 0 Then
      i = WSAGetLastError()   <----------Obtenemos el número de error.
          lblMensaje.Caption = "Error de conexión:" + Str$(i) & "en la
máquina: " & Dir_IP
         res = 100
Else
   En caso de no haber error en la conexión, enviaremos
   el mensaje
   i = send(xsocket, Trim$(mensaje), Len(Trim$(mensaje)), Flags) <--
                                                          Enviamos el
                                                       mensaje deseado
   lblMensaje.Caption = "Enviando información....."
   res = 0
End If
End Sub
}
```

Definición del socket para recepción de información desde Windows:

La siguiente subrutina, es encarga de escuchar de cualquier máquina que le realize una petición, sobre un puerto determinado.

Código Vb.

```
Sub escucha (xsocket As Integer, num_puerto As Integer)
Dim i As Integer
xsocket = Socket(AF_INET, SOCK_STREAM, IPPROTO_IP)
s_in.sin_family = AF_INET
s_in.sin_port = htons(num_puerto)
lblMensaje.Caption = "Construyendo el Socket ....."
i = bind(xsocket, s_in, Len(s_in))      <----------se construye el socket
'* Existe algún error en la contrucción del socket?
  If i <> 0 Then
    i = WSAGetLastError()
    MsgBox "Error de comunicación.... : " + Str$(i)
  Else
     '* Se asume que no existe algún error en la construcción del socket
    '* por lo que se procederá a escuchar.
     i = WSAASYNCSELECT(xsocket, WinMsg1.hWnd,
WinMsg1.NEWTEvent, FD_ACCEPT)
    '* Preguntamos si existe algún error en el momento de la construcción
     If i <> 0 Then
       i = WSAGetLastError()
       MsgBox "WSAAsyncSelect error : " + Str$(i)
     Else
      '* De con existir algún error se procederá a escuchar.
       lblMensaje.Caption = "Recibiendo información remota..."
       i = listen(xsocket, 0)      <----------Función que escucha.
       '* Existe algún error en el momento de escuchar?
       If i <> 0 Then
         i = WSAGetLastError()
         MsgBox "Error al conectar con la máquina remota : " + Str$(i)
       End If
     End If
  End If
End Sub
```

V.3.4 Monitoreo de estaciones Unix.

El monitoreo permite detectar cuantas máquinas están conectadas a la red, y muestra de color diferente, la estación en la que se esta trabajando. Si hay algún problema en la red, como fallas en el cable, basta con presionar el icono de actualizar los datos, para que el sistema en windows revise de nueva cuenta cuales son las estaciones activas.
La subrutina encargada de sensar la actividad de las máquinas es la siguiente:

Código Vb.

```
For i = 0 To num_maquinas Step 1
    IP = Trim$(Nodos(i))   <----------Es el vector que contiene las direcciones IP.
      If IP = "" Then
      Else
       'Llamamos a la subrutina que es encargada de preguntar si
       'la máquina está disponible para responder
        res = 0
        envia Pto_envío, asocket, "ok?", IP, res
        If res = 100 Then   'El número 100 representa
                            'simplemente que la máquina está inactiva
        Else
                  'Si la máquina está activa, se guarda 'en un arreglo de nodos ok.
         Nodos_ok(j) = Nodos(i)'máquina <-Vector actualizado de estaciones de
                                          trabajo.
          If j < 4 Then      'Dibuja las máquinas activas en la RED.
       Esconde_habilita 1, j + 1, Nodos_ok(j) <-subrutina para dibujar los iconos
                                                 activos.
           maquinas = maquinas + 1
         Else
         End If
         j = j + 1
        End If
     End If
  Next i
```

Por su parte la estación de trabajo simplemente reacciona a la petición de envío de actividad, sin realizar alguna acción específica.

V.3.5 Detección de estaciones Unix activas (Nuevo Ping)

Primeramente, al iniciar el sistema, este rastrea todas las estaciones Unix que estén "activas", es decir, las estaciones que estén encendidas, conectadas a la red, y corriendo la aplicación SmauxW. Esta búsqueda es controlada por el archivo uhosts, el cual almacena la dirección IP y el nombre de la estación. El programa toma cada dirección IP de este archivo y la procesa con la función Nuevo Ping, el cual a diferencia del Ping normal (el cual detecta que la tarjeta de la estación este activa), envía un mensaje de reconocimiento, y sensa si la estación Unix se encuentra respondiendo mediante el programa SmauxW.

Esta rutina es basada en la subrutina de envío de mensajes:

Código Vb.

```
Sub Nuevo_Ping_Click ()
Dim nmáquina   As String
Dim Res As Integer
frmPing.MousePointer = 11
envia Pto_envío, asocket, "ok?", Trim$(lblNPing.Text), Res <----enviamos el
                                                            mensaje a la máquina deseada.
frmPing.MousePointer = 0
End Sub
```

V.3.6 Peticiones desde Windows .

El sistema en windows genera un código para cada acción. Por Ejemplo si se requiere dar de alta un usuario, se manda el código 200, junto con el buffer que contiene la infirmación para dar de alta un usuario en Unix. El código es interpretado por el sistema Unix y procesa la acción según sea el caso.

El código usado es:

200: Petición de alta de usuario
301: Baja y cambio de usuarios
400: Envío de información (usuarios, grupos, características, procesos)
500: Eliminación de procesos.

La siguiente rutina mostrará básicamente el funcionamiento cuando se realiza una petición al servidor, donde "cadena" representa el buffer a enviar, y del cual se cuenta con un protocolo previamente establecido..

Código Vb.

```
cadena = cadena & "@" & Trim$(Str$(X + 1)) & "@" & "4" & "@" &
Trim$(ComboUserIdCambios.Text) & "::" & Trim$(txtUsuarioCambios.Text)
& ":"
cadena = cadena & Trim$(txtGroupCambios.Text) & ":" &
Trim$(txtNombreCambios.Text) & ":"
cadena = cadena & Trim$(txtDirTrabajoCambios.Text) & ":" &
Trim$(shellcambios) & "@"
cadena = cadena & Trim$(Dir_Cambio) & "@" & Trim$(directorio_old)
cadena = cadena & "@" & Trim$(directorio)

If acción = "OK" Then
  'Quiere decir que no hay problemas en enviar la información
   'lblMensaje.caption = "Ejecutando Alta de Usuario ....Espera un momento.
"

  frmMonitor.MousePointer = 11
  envía Pto_envío, csocket, cadena, Trim$(IP_ACTUAL), res
    If res = 100 Then  'Esto significa que existe un problema de comunicación o en la
                       dirección
        lblMensaje.Caption = "ERROR al conectarse, verifica la dirección y
la RED!!!"
    Else
      'Informamos que recibiremos información
                       i = WSAASYNCSELECT(asocket, WinMsg1.hWnd,
WinMsg1.NEWTEvent, FD_ACCEPT)
      'Detectamos si existe algún error
      If i <> 0 Then
        i = WSAGetLastError()
        MsgBox "WSAAsyncSelect error : " + Str$(i)
```

```
        Else
            lblMensaje.Caption = "Esperando status de la alta del Usuario :"
+ ComboNombre.Text
          While (Status <> "siéxito") And (Status <> "noéxito")
            DoEvents
            Status = Trim$(Respuesta)
            frmMonitor.MousePointer = 11
          Wend
          frmMonitor.MousePointer = 0
          Select Case Status
            Case "siéxito"
              lblMensaje.Caption = "Cambio Exitoso !!!!!"
              X = i
              Usuarios(i, 3) = Trim$(txtUsuarioCambios.Text)
              Usuarios(i, 4) = Trim$(txtGroupCambios.Text)
              Usuarios(i, 5) = Trim$(txtNombreCambios.Text)
              Usuarios(i, 6) = Trim$(txtDirTrabajoCambios.Text)
              'Usuarios(i, 7) = Trim$(shellcambios)
               cmdUsuarios
            Case "noéxito"
              lblMensaje.Caption = "Cambio NO Exitoso"
            End Select
        End If
      End If
Else
  MsgBox "Por favor verifica los datos...."
End If
```

V.3.7 Funcionamiento de "modo de espera de Unix".

Esto significa que el programa en Unix está siempre esperando las peticiones que son enviadas desde el sistema en Windows. Cuando se recibe una petición, el sistema Unix procesa la orden y continúa esperando para procesar la siguiente orden.

V.3.8 Reconocimiento de la dirección IP.

El sistema Unix es capaz de detectar la dirección IP de la máquina que este mandando un mensaje a esta. Es decir, cuando un mensaje es recibido a Unix, el sistema toma de la estructura del socket la dirección IP y contesta las peticiones a esta dirección.
Con esto el sistema en Unix es capaz de contestar a varios monitores windows a la vez, no importando cuantos sean y no importando si la máquina que esta corriendo el sistema en windows cambia de dirección IP.

Código "C"

```
/*****************************************************/
/************ función para obtener ip remota *************/
/********* obtiene la ip address de la estructura del *********/
/********* socket del que se esta recibiendo el buffer *******/
/*****************************************************/

obten_ip(new_sock)

int new_sock;
{
 int peer;
 char *ip;
 struct sockaddr_in sockip;
 peer=sizeof(sockip);
 if(getpeername(new_sock, (struct sockaddr *) &sockip,&peer)!=0)
   perror("getpeername"),exit(1);
 else
  {
   ip=inet_ntoa(sockip.sin_addr);
   strcpy(ipremota,ip);
    }
}
```

V.3.9 Envío de información desde Unix-PC (NuevoFTP).

Se define un socket para envío de información del sistema por el puerto **6**, y así mismo este puerto funciona para envío de mensajes internos del sistema (códigos de error y éxito). La rutina toma el archivo especificado por windows y lo envía línea por línea a través del socket por el puerto indicado hasta que esta encuentra un fin de archivo.

Código "C"

La definición del socket para NuevoFTP es:

```
/*******************************************************/
/*************** NuevoFTP con sockets ******************/
/*******************************************************/
NuevoFTP()
{
 int sd2,i=1;
 struct sockaddr_in from,sockname;
 char buffer2[120],*cadena,*codigo,*archiv="0",*file="0";
 FILE *pfichero;

if((sd2=socket(AF_INET,SOCK_STREAM,0))==-1){
  perror("Cliente:Socket");
  printf("El socket de envío de archivos no pudo abrirse\n");
  exit(1);
}
sockname.sin_family=AF_INET;
sockname.sin_addr.s_addr=inet_addr(ipremota);
sockname.sin_port=htons(6);
if(connect(sd2,&sockname,sizeof(sockname))==-1){
 perror("Cliente:Connect");
 printf("El socket de envío de mensajes no pudo conectarse\n");
 return(0);
}
```

El proceso de enviado de información es como se muestra a continuación:

```
do{
  bzero(buffer2,sizeof buffer2);
```

```
  cadena=fgets(buffer2,80,pfichero);
  if(send(sd2,buffer2,120,0)==-1){
    close(sd2);
    envía_mensaje(2);
    perror("Cliente:Send");
    printf("El buffer no pudo enviarse a windows\n");
  return(0);
  }
  printf("buffer=%s\n",buffer2);
}while(cadena!=NULL);
```

V.3.10 Recepción de información desde Windows.

Para recibir la información que es enviada desde Unix, en el sistema en windows se tiene un ciclo que lee continuamente todo lo que llega por el puerto 6 definido con un socket de recepción de Información. En algunos casos, el sistema en windows genera un archivo plano con la información recibida y luego la procesa. En otros no genera archivos para poder manipular los datos, sino que deja la información en la estructurada de datos del sistema, lo cual hace mas rápida su manipulación.

Conclusiones.

Durante el desarrollo de este trabajo adquirimos diferentes conocimientos, los cuales consideramos que forman parte importante en nuestro desarrollo profesional. El desglose de los capítulos I al IV fue pensando en los requerimientos necesarios para lograr el entendimiento de una aplicación realizada en ambiente Windows e interactuando con estaciones Unix por medio de "Sockets". De acuerdo a lo anterior, creemos conveniente destacar algunos puntos que desde nuestro punto de vista representan lo más importante de cada capítulo.

Ideas más importantes de cada capítulo.

Capítulo I. TCP/IP representa uno de los protocolos que hoy en día tienen mayor participación en las empresas a nivel mundial. Es una excelente opción para aquellas empresas que requieren contar con una comunicación segura entre sus diferentes nodos de la red. Gran parte de las ventajas que tiene, es que la mayoría de los proveedores de software y de diferentes plataformas, cuentan con esta implementación.

Capítulo II. Creemos que hoy en día no se ha explotado al máximo una gran opción para el intercambio de información entre procesos tales como "los Sockets". Consideramos también, que con el surgimiento de "los Winsockets", se puede lograr una integración e intercambio de información de manera rápida y eficaz entre diferentes sistemas y diferentes ambientes.

Capítulo III. En la actualidad, no obstante de entre la gran variedad de Sistemas Operativos que podemos encontrar en el mercado, creemos que Unix no ha sido superado y por lo tanto se mantendrá activo por un buen tiempo. Pese a la gran variedad de "versiones de Unix" por parte de las diferentes empresas, estamos convencidos en que Unix es un Sistema Operativo muy robusto y que cuenta con un gran potencial para das solución a las diferentes necesidades de las empresas.

Capítulo IV. La evolución tecnológica ha concentrado sus esfuerzos en la creación de Sistemas cada vez más amigables para Usuarios Finales. Consideramos que existen una gran cantidad de herramientas para desarrollo, y que sin importar las ventajas y desventajas de cada una, representan en la actualidad una oportunidad para la mayoría de las empresas.

Capítulo V. Como mencionamos en un principio, creemos que el software SmauxW es un producto muy reciente por la tecnología utilizada, y más que eso, realizamos la propuesta para que todos aquellos desarrolladores, empresarios, ingenieros y gente de Sistemas en general, se interesen en el desarrollo de productos de este tipo.

Las lecciones que obtuvimos con la realización del trabajo.

Durante el desarrollo del presente trabajo obtuvimos diferentes lecciones, de las cuales haremos mención.

- El desarrollo de aplicaciones basadas en "Sockets" darán como resultado aplicaciones con un alto índice de confiabilidad para el intercambio de información, sin importar el ambiente en que se esté operando.
- Sabemos también, que no es fácil desarrollar un software con estas características, sin embargo recomendamos como primer punto, entender el ciclo de comunicación entre sockets en un esquema Cliente-Servidor.

Sobre la aplicación.

SmauxW es un software que cumple con las necesidades básicas de los administradores de estaciones Unix, siendo así de gran utilidad para ellos.

SmauxW representó esfuerzo, motivación, y es un reto hacia las empresas que actualmente basan sus aplicaciones en Sockets.

Sabemos que existen diferentes puntos que se podrían agregar en dicho software, sin embargo insistimos que esta aplicación en una de las pioneras en el ramo de las comunicaciones "Sockets-WinSockets".

La integración Visual Basic-WinSocket, es altamente recomendada por nosotros, debido a su sencillez en el manejo, a su facilidad de entendimiento, y a que nos permite generar una aplicación amigable.

Estamos seguros que SmauxW es una opción de muy bajo costo comparado con sistemas que requieren de hardware especial. Pensamos que debido a esta tendencia en el crecimiento de aplicaciones en Windows utilizando sockets, los proveedores de tecnologías para Unix tendrán que disminuir sus precios, o bien, realizar diferentes ajustes en el software.

Nuestro mensaje final.

Apoyamos el desarrollo de aplicaciones en ambientes gráficos. Consideramos que hoy en día las empresas no deben perder de vista el desarrollo de los Winsockets, ya que estos son una herramienta fácil de usar y nos permiten comunicar diferentes procesos sin importar el ambiente bajo el cual estén operando.

Consideramos que las empresas deben estar preparadas para los diferentes cambios de tecnologías, y por los mismo deben pensar en la integración de los mismos con herramientas como: Sockets-Winsockets.

Por último recomendamos ampliamente a los administradores de estaciones Unix nuestra herramienta SmauxW, porque les permitirá llevar a cabo algunas de sus funciones cotidianas sin mayor esfuerzo y con un ambiente totalmente amigable.

Apéndice A. Winsockets.

Los sockets tienen una serie de funciones que son estándares en cualquier sistema operativo de red y las cuales permiten que las aplicaciones se intercomuniquen en diferentes ambientes, sin embargo, cuando se definieron los sockets para Windows, se agregaron una serie de extensiones que son propietarias de los winsockets y que les brindan una flexibilidad adicional para soportar las comunicaciones. A continuación describiremos brevemente algunas de éstas:

WSAAsyncGetHostByAddr(HWND hWnd, unsigned int wMsg, const char FAR *addr,
int len, int type, char FAR *buf, int buflen);

Esta función obtiene la información del Host correspondiente a una dirección en una comunicación asíncrona. Los parámetros que maneja esta función son los siguientes:

hWnd	Es el handle de la ventana que debería recibir un mensaje cuando sea completado el request a la comunicación asíncrona.
wMsg	El mensaje a recibirse cuando culmine el request.
addr	Un apuntador a la dirección de red del Host.
len	La longitud de la dirección, que debe ser 4 para PF_INET.
type	El tipo de dirección, que debe ser PF_INET
buf	Un apuntador al área de datos que recibirá la información del Host.
buflen	El tamaño del área de datos de buf.

WSAAsyncGetHostByName(HWND hWnd, unsigned int wMSG, conts char FAR *name, char FAR *name, char FAR *buf, int buflen);

Esta función es similar a la anterior, pero se diferencia en que obtiene la información por nombre en lugar de por dirección.

hWnd Es el handle de la ventana que debería recibir un mensaje cuando sea completado el request a la comunicación asíncrona.
wMsg El mensaje a recibirse cuando culmine el request.
name Un apuntador al nombre del Host.
buf Un apuntador al área de datos que recibirá la información del host.
buflen El tamaño del área de datos de buf.

WSAAsyncGetProtoByName(HWND hWnd, unsigned int wMsg, const char FAR *name, char Far *buf, int buflen);

Permite obtener información del protocolo, correspondiente a un nombre de protocolo con comunicación asíncrona. Utiliza los siguientes parámetros.

hWnd Es el handle de la ventana que debería recibir un mensaje cuando sea completado el request a la comunicación asíncrona.
wMsg El mensaje a recibirse cuando culmine el request.
name Un apuntador al nombre del protocolo que será resuelto.
buf Un apuntador al área de datos que recibirá la información del protocolo.
buflen El tamaño del área de datos de buf.

WSAAsyncGetProtoByNumber(HWND hWnd, unsigned int wMsg, int number, char FAR *buf, int buflen);

Obtiene la información del protocolo, correspondiente a un número de protocolo con comunicación asíncrona y cuenta con los parámetros que a continuación se enlistan.

hWnd Es el handle de la ventana que debería recibir un mensaje cuando sea completado el request a la comunicación asíncrona.
wMsg El mensaje a recibirse cuando culmine el request.
number El numero de protocolo que será resuelto.

buf Un apuntador al área de datos que recibirá la información del protocolo.
buflen El tamaño del área de datos de buf.

WSAAsyncGetServByName(HWND hWnd, unsigned int wMsg, const char FAR *name, const char FAR *proto, char FAR *buf, int buflen);

Esta función obtiene información del servicio correspondiente a su nombre y puerto en una comunicación asíncrona. Sus parámetros son los siguientes.

hWnd Es el handle de la ventana que debería recibir un mensaje cuando sea completado el request a la comunicación asíncrona.
wMsg El mensaje a recibirse cuando culmine el request.
name Un apuntador a un nombre de servicio.
proto Un apuntador a un nombre de protocolo.
buf Un apuntador al área de datos que recibirá la información del protocolo.
buflen El tamaño del área de datos de buf.

WSAAsyncGetServByPort(HWND hWnd, unsigned int wMsg, int port, const char FAR *proto, char FAR *buf, int buflen);

Permite obtener información relativa a un puerto y protocolo en una comunicación asíncrona.

hWnd Es el handle de la ventana que debería recibir un mensaje cuando sea completado el request a la comunicación asíncrona.
wMsg El mensaje a recibirse cuando culmine el request.
port El puerto del servicio.
proto Un apuntador a un nombre de protocolo.
buf Un apuntador al área de datos que recibirá la información del protocolo.
buflen El tamaño del área de datos de buf.

WSAAsyncSelect(SOCKET s, HWND hWnd, unsigned int wMsg, long lEvent);

Solicita notificación de eventos para un socket. Esta función se utiliza para solicitar que la DLL de los winsockets envíe un mensaje a la ventana hWnd siempre que detecte cualquier evento especificado por el parámetro lEvent. El mensaje que debería enviar es especificado por el parámetro wMsg. El socket para el cual se requiere la notificación es especificado por el parámetros .

s Un descriptor que identifica el socket para el cual se requiere la notificación de eventos.
hWnd Es el handle de la ventana que debería recibir un mensaje cuando se genere un evento en la red.
wMsg El mensaje a recibirse cuando ocurra un evento en la red.
lEvent Una máscara que especifica la combinación de eventos que se desean monitorear.

WSACancelAsyncRequest(HANDLE hAsyncTaskHandle);

Cancela una operación asíncrona inconclusa. Necesita del siguiente parámetro.

hAsyncTaskHandle Especifica la operación asíncrona a ser cancelada.

WSACancelBlockingCall(void);

Cancela una llamada de bloque que se encuentre en proceso. Normalmente se utiliza en dos situaciones, cuando una aplicación esta procesando un mensaje que ha sido recibido mientras se tenía una llamada de bloque en proceso, o bien cuando se encuentra en proceso una llamada de bloque y los winsockets realizan una rellamada (callback) a una función de la aplicación.

WSACleanUp(void);

Finaliza el uso de la Librería de los winsockets, ya que se requiere de una DLL para iniciar exitosamente los servicios de los winsockets, es necesario que, cuando ya no se utilicen, ésta sea removida de la memoria para optimizar el uso de la misma.

WSAGetLastError(void);

Obtiene el status de error para la última operación no exitosa. Cuando una función API de winsockets en particular indica que un error ha ocurrido, esta función debe ser llamada para obtener el código de error que se generó.

WSASetLastError(int iError);

Define el código de error que puede ser recuperado por WSAGetLastError(), permite a la aplicación definir un código de error que será regresado por una llamada de WSAGetLastError() para el proceso actual.

WSAStartup(WORD wVersionRequested, LPWSADATA lpWSAData);

Esta función debe ser la primera en llamarse por una aplicación o DLL. Permite a la aplicación especificar la versión requerida de los APIs de winsockets y obtener detalles específicos de la implementación de los mismos.

Apéndice B. Puertos Estándar (Universidad de Berkeley).

```
# @(#)services 1.16 90/01/03 SMI

# Network services, Internet style
# This file is never consulted when the NIS are running
#
tcpmux      1/tcp                   # rfc-1078
echo        7/tcp
echo        7/udp
discard     9/tcp       sink null
discard     9/udp       sink null
systat      11/tcp          users
daytime     13/tcp
daytime     13/udp
netstat     15/tcp
chargen     19/tcp          ttytst source
chargen     19/udp      ttytst source
ftp-data    20/tcp
ftp         21/tcp
telnet      23/tcp
smtp        25/tcp          mail
time        37/tcp          timeserver
time        37/udp      timeserver
name        42/udp      nameserver
whois       43/tcp          nicname     # usually to sri-nic
domain      53/udp
domain      53/tcp
hostnames   101/tcp     hostname    # usually to sri-nic
sunrpc      111/udp
sunrpc      111/tcp
#
```

```
# Host specific functions
#
tftp            69/udp
rje             77/tcp
finger          79/tcp
link            87/tcp          ttylink
supdup          95/tcp
iso-tsap        102/tcp
x400            103/tcp                 # ISO Mail
x400-snd        104/tcp
csnet-ns        105/tcp
pop-2           109/tcp                 # Post Office
uucp-path       117/tcp
nntp            119/tcp         usenet          # Network News Transfer
ntp             123/tcp                 # Network Time Protocol
NeWS            144/tcp         news            # Window System
#
# UNIX specific services
#
# these are NOT officially assigned
#
exec            512/tcp
login           513/tcp
shell           514/tcp         cmd             # no passwords used
printer         515/tcp         spooler         # line printer spooler
courier         530/tcp         rpc             # experimental
uucp            540/tcp         uucpd           # uucp daemon
biff            512/udp         comsat
who             513/udp         whod
syslog          514/udp
talk            517/udp
route           520/udp         router routed
new-rwho        550/udp         new-who         # experimental
rmonitor        560/udp         rmonitord       # experimental
monitor         561/udp                         # experimental
pcserver        600/tcp                         # ECD Integrated PC board srvr
ingreslock      1524/tcp
```

Glosario.

Adaptador: Tarjeta de una PC, normalmente instalada dentro de un máquina, que ofrece capacidades de comunicación de red desde y hacía la computadora.

Address (Dirección): Estructura de datos empleada para identificar una entidad única, como algún proceso o la localización de una red.

Alarma: Mensaje que avisa al operador o administrador sobe problemas en la red.

API: Application Programming Interface. Interface para programas de aplicación. Especificación de convenciones de llamadas a funciones para definir la interface de servicio.

BDS: Berkeley Distribution Software.

CCITT: Comité Consultivo Internacional de Telegrafía y Telefonía. (Siglas en francés). Organización Internacional que desarrolla estándares de comunicaciones, como la recomendación X25.

Cliente: Nodo o programa de software que requiere servicios de un servidor.

Client-Server Computing: Computación en modo cliente-servidor. Término empleado para describir sistemas de redes de procesamiento distribuido, en donde las transacciones se dividen en dos partes: el cliente (front end) y el servidor (back end). Ambos términos se puede aplicar tanto a programas como a dispositivo de cómputo.

Datagrama: Agrupamiento lógico de información enviada como unidad de la capa de red en un medio de transmisión, sin el establecimiento previo de un circuito virtual. Los términos paquete, marco, segmento y mensaje también se emplean para describir agrupaciones lógicos de información de varios niveles

del modelo de referencia OSI y en otras áreas de la tecnológía. Los datagramas IP son las unidades primarias de información en Internet.

Dominio: En Internet, porción de un árbol de jerarquía de nombres.

Dominio de comunicación: es una abstracción introducida para manejar las propiedades de las comunicaciones.

Frame (Marco): Agrupamiento lógico de información enviado a un medio de transmisión como una unidad de la capa de enlace. Los términos paquete, datagrama, segmento y mensaje también se emplean para describir agrupamiento lógicos de información en varias capas del modelo de referencia OSI.

FTP (File Transfer Protocol): Protocolo de transferencia de archivos. Protocolo de aplicación IP para transferir archivos entre nodos de la red.

GUI (Graphic User Interface): Es una aplicación que permite a los usuarios manejar el ambiente operativo que utilizan en sus computadoras, por medio de elementos gráficos (íconos y botones), que son activados con un mouse o por medio de teclado, pero sin tener que escribir complicados comandos.

IEEE (Institute of Electrical and Electronic Enginners): Instituto de Ingenieros Eléctricos y Electrónicos. Organización profesional que define estándares de redes. Los estándares LAN de IEEE son los predominantes en la actualidad, e incluyen protocolos similares o virtualmente equivalentes a Ethernet y Token Ring.

Internet: Término empleado para referirse al sistema de interconexión de redes más grande del mundo, que conecta miles de redes en todo el planeta, que desarrollo una cultura basada en simplicidad investigación y estandardización fundamentada en el uso real. Buena parte de la tecnología de punta en redes vino de esta comunidad. Internet evoluciono a partir de ARPANET.

IP (Internet Protocol): Protocolo Internet. Protocolo de capa 3 (Capa de red) que contiene información de direccionamiento y de control para permitir el enrutamiento de paquetes.

ISO (International Organization for Standarization): Organización Internacional para la Estandardización. Organización Internacional responsable de una amplia gama de estándares, incluyendo aquellos relevantes para las redes. ISO es la responsable del modelo de referencia de redes más popular: el modelo de referencia OSI.

LAN (Local-Area-Network): Red de área local. Red que cubre un área geográfica relativamente pequeña (usualmente no mayor que un grupo local de edificios). Comparadas con las redes WAN, las redes LAN suelen caracterizarse por velocidades en la transferencia de datos relativamente altas y una relativamente baja incidencia de errores.

NFS (Network File System): Sistema de archivos en red. Como se emplea normalmente, es un conjunto de protocolos de sistemas de archivos distribuidos desarrollado por la empresa SUN Microsystems, que permite el acceso remoto a archivos en una red. En realidad, NFS es uno de los protocolos del conjunto, que incluye NFS, XDR (External Data Representation: Representación externa de datos), RPC (Remote Procedure Call: Llamada Remota a Procedimientos), y otros. Esos protocolos son parte de una arquitectura mayor que la empresa SUN nombra como ONC (Open Network Computing).

OSI (Open System Interconnection): Interconexión abierta de sistemas. Programa internacional de estándarización, apoyado por ISO y CCITT, para desarrollar estándares para redes de datos. Facilita la interoperabilidad de equipos hechos por diferentes fabricantes.

SmauxW (Sistema de Monitoreo y Administración de nodos Unix bajo ambiente Windows): Es el nombre de la aplicación desarrollada como objetivo de este trabajo.

SNMP (Simple Network Management Protocol): Protocolo simple de manejo de redes. El protocolo de manejo de redes de Internet ofrece medios para seguir y determinar la configuración de la red y los parámetros al tiempo de ejecución.

TCP/IP (Transmission Control Prtocol / Internet Protocol): Protocolo de control de transmisiones / Protocolo Internet. Los dos protocolos Internet más conocidos, que erróneamente suelen confundirse con uno sólo. TCP corresponde a la capa 4 (capa de transporte) del modelo de referencia OSI y

ofrece transmisión confiable de datos. IP corresponde a la capa 3 (capa de red) del modelo de referencia OSI, y ofrece servicio de datagramas sin conexión.

Telnet: Protocolo estándar Internet de emulación de terminales.
UDP (User Datagram Protocol): Protocolo de datagrama de usuario. Protocolo sin conexión de la capa de transporte y pertenece a la familia de protocolos Internet.

WAN (Wire-Area-Network): Red que ocupa un área geográfica amplia.

Bibliografía.

1.- TCP/IP Network Administration.
Autor Hunt Craig.
Editorial O´Reilly & Associates, Inc.

2.- Fundamentos de los Microprocesadores y Estaciones de Trabajo.
Copyright Xerox Corporation 1992.

3.- Interworking Technology Overview.
Copyright Cisco Systems 1993.

4.- Managing NFS and NIS.
Autor Stern Hal.
Editorial O´Reilly & Associates, Inc.

5.- UNIX System Administration Handbook.
Autor Nemeth Evi, Garth Snyder, Scott Seebass.
Editorial Prentice Hall, 1989.

6.- UNIX Sistema V Versión 4.
Autor Rosen Kenneth H., Richard R. Rosinski, James M. Farber.
Editorial Mc Graw-Hill, 1991.

7.- Redes de Computadoras.
Autor Black Uyless
Editorial Macrobit, 1989.

8.- Interworking with TCP/IP. Vol. I,II,III.
Autor Comer Douglas E., David L. Stevens.
Editorial Prentice Hall, 1993.

9.- System Administration 4.1.2.
Copyright SUN 1987.

10.- TCP/IP and NFS. Interworking in UNIX Enviroment.
Autor Santifaller Michael.
Editorial Addison-Wesley.

11.- Comunicaciones en UNIX.
Autor Rifflet Jean-Marie.

12.- Sys Admin. The Journal for UNIX Systems Administration.
Enero-Febrero 1994.

13.- Sys Admin, The Journal for UNIX Systems Administration.
Julio-Agosto 1993.

14.- RED.
Noviembre 1994.

15.- Gupta Corp. SQLWindows
DATAPRO information Services Group.

16.- Software Productivity
Número 2, 1994.
Powersoft Latin America.

17.- NEWT-SDK TCP/IP for Windows Develper's Guide
NetManage Corporation

10.- TCP/IP and NFS. Interworking in UNIX Enviroment.
Autor Santifaller Michael.
Editorial Addison-Wesley.

11.- Comunicaciones en UNIX.
Autor Rifflet Jean-Marie.

12.- Sys Admin. The Journal for UNIX Systems Administration.
Enero-Febrero 1994.

13.- Sys Admin, The Journal for UNIX Systems Administration.
Julio-Agosto 1993.

14.- RED.
Noviembre 1994.

15.- Gupta Corp. SQLWindows
DATAPRO information Services Group.

16.- Software Productivity
Número 2, 1994.
Powersoft Latin America.

17.- NEWT-SDK TCP/IP for Windows Develper's Guide
NetManage Corporation