UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

Facultad de Ingeniería

DISEÑO E IMPLEMENTACION DE UN OPERADOR AUTOMATICO QUE REPORTE ANOMALIAS DEL SISTEMA POR MEDIO DE SINTESIS DE VOZ EN LOS EQUIPOS SPARC SUN DE LA D. G. B.

TESIS

Que para obtener el título de

INGENIERO EN COMPUTACION

presentan

Patricia Yolanda Monzón Rodríguez
Felipe Reyna Rosey

Director de Tesis: ING. MARCIAL CONTRERAS BARRERA

México, D. F. **1999**

AGRADECIMIENTOS

Al Ing. Marcial Contreras Barrera por todo el apoyo recibido para la realización de esta tesis y sobre todo por su amistad siempre incondicional.

Al personal del Departamento de Producción de la Dirección General de Bibliotecas por la ayuda recibida.

A la Facultad de Ingeniería de la UNAM

DEDICATORIAS

A mis padres Antonia y Domingo que han sido el motivo de lo que soy y que me han enseñado a valorar ello.

A mis hermanos por su amistad, compañía y consejos presentes en todo momento de mi vida.

A mis amigos por la alegría de haberlos conocido.

Patricia

AGRADECIMIENTOS

Al Ing. Marcial Contreras Barrera
por el apoyo, tiempo, dedicación y
consejos que nos ha proporcionado
en el desarrollo de la tesis, en un
esfuerzo conjunto que ha dado frutos.

A todos mis compañeros que laboran en
el Departamento de Producción, por el
apoyo recibido y por brindarme su amistad.

A mis maestros que a lo largo del camino
del conocimiento me han transmitido su
sabiduría, formándome como persona.

A todas aquellas personas que contribuyeron
a la realización de esta tesis.

DEDICATORIAS

A Dios:
Por darme un espacio en esta vida permitiendo que la disfrute plenamente en compañía de mis seres queridos.

A mis Padres:
Felipe y Roma quienes han depositado en mí todo su cariño, afecto y confianza para lograr esta meta.

A Maru:
Que ha sido una guía a lo largo del camino, motivándome cada día para mejorar en todos los aspectos de mi vida.

A Adri y Vicky:
Por todos los momentos de alegría que hemos pasado juntos, y que a pesar de la distancia, los lazos que nos unen se fortalecen.

A Daniel:
Quien más que un hermano, es un amigo de verdad, en quien confío plenamente.

A Paty:
Por todo este tiempo que hemos convivido, disfrutando cada uno de nuestros logros, y por enseñarme cada día el valor de la amistad.

A todos mis amigos:
Con quienes comparto este trabajo y la inmensa alegría que siento.

Felipe

ÍNDICE

Introducción General

El avance en la tecnología conlleva al desarrollo de mejores equipos de cómputo que se utilizan en diversos sectores(laboral, educativo, empresarial y social) para realizar sus funciones en forma ágil, sencilla y segura. El manejo de la información es el punto primordial y de mayor interés, ya que existe la necesidad de intercambiar ésta y compartir recursos entre sistemas de diferentes arquitecturas y ambientes operativos, dada su importancia se utilizan nuevos métodos para garantizar su integridad y eficiencia.

Dentro de las comunicaciones, UNIX tiene un papel relevante, por ser un sistema operativo multiusuario, estándar, con servicios de red integrados y disponible prácticamente en cualquier plataforma de hardware existente.

En el ámbito de las computadoras personales (PC) Microsoft es el líder en desarrollo de software con la gran variedad de aplicaciones Windows y programas orientados a usuarios finales que se utilizan en casi todas las PC's. Con la liberación del sistema operativo gráfico Windows 98 que mejora las características de Windows 95, con servicios de comunicación y manejo de dispositivos integrados.

La Dirección General de Bibliotecas(DGB) consciente de este aspecto y acorde al desarrollo tecnológico ha realizado cambios sustanciales para el mejor cumplimiento de sus funciones, que se clasifican en tres tipos:

Funciones de normalización y coordinación:

- Establece la norma en los servicios bibliotecarios y de información
- Coordina el cumplimiento de las políticas generales que establece el consejo Bibliotecario de la UNAM

Funciones de operación:

- Lleva a cabo los proyectos establecidos para el desarrollo del Sistema Bibliotecario

Funciones de servicio:

- Ofrece servicios bibliotecarios y de información, utilizando nuevos métodos y tecnologías, que incluyen el desarrollo y funcionamiento de bases de datos y otras fuentes informativas
- Ofrece de forma efectiva y actualizada el sistema de información bibliotecaria

Para el logro de éstas funciones la DGB cuenta con una estructura distribuida por subdirecciones: Subdirección de Informática, Subdirección Técnica, Subdirección de Servicios Bibliotecarios y Subdirección de Planeación y Desarrollo.

La Subdirección de Informática desarrolla, actualiza e implementa software aplicable en el Sistema Bibliotecario, así mismo administra los equipos de cómputo que contienen las bases de datos creadas en la DGB.

El área laboral donde se desarrolla el proyecto de Diseño e Implementación de un Operador Automático que Reporte Anomalías del Sistema por Medio de Síntesis de voz en los Equipos Sparc Sun de la DGB es en el Departamento de Producción que a su vez forma parte de la Subdirección de Informática. El objetivo es proporcionar una herramienta de apoyo en la administración de los equipos de cómputo de manera sencilla y eficaz. Para realizarlo se requieren aspectos como la programación de rutinas de comunicación entre computadoras y otras que capturen los mensajes de los equipos UNIX, los cuales son manipulados y enviados a una PC que tiene entre otras funciones la traducción de texto a voz con lo cual se generará una señal de alerta audible en caso de presentarse algún error, que amerite la revisión inmediata del sistema y tomar medidas pertinentes.

Este trabajo está dividido en 6 capítulos, dando a continuación el contenido de cada uno de ellos:

Capítulo 1. Antecedentes del proyecto. En este capítulo se plantea la problemática existente y se explican los conceptos que utilizamos en el desarrollo del proyecto.

Capítulo 2. Lenguajes visuales de desarrollo. Se describen las características de algunas herramientas de desarrollo en ambiente Windows.

Capítulo 3. DLL's. En este capítulo se describen las características, estructura y ventajas en la programación.

Capítulo 4. Protocolos de comunicación TCP/IP. Se explica su modelo, capas que lo constituyen y los protocolos que lo integran. Además se compara con el modelo OSI.

Capítulo 5. Sockets. En este capítulo se hablará de las características, creación en ambiente *UNIX* y *Windows*. Así como función y relación con los sockets.

Capítulo 6. Diseño e implementación del proyecto. Presenta la aplicación práctica de los conceptos antes mencionados.

Conclusiones. Daremos nuestro punto de vista sobre el sistema desarrollado, así como las experiencias adquiridas en el desarrollo de la aplicación.

Capítulo 1.

Antecedentes del Proyecto

1.1 Bases de datos de la Dirección General De Bibliotecas (DGB)

El programa de automatización de información en la DGB, desarrolló las bases de datos de *libros, tesis, préstamo(circulación), adquisición, revistas, más otras bases de datos que se han ido adicionando a éste,* las cuales están distribuidas en 7 equipos de cómputo y almacenadas en arreglos de discos(storage arrays). Las bases de datos más importantes por el volumen de información que contienen son descritas a continuación:

LIBRUNAM. En esta base de datos se encuentra contenido el registro bibliográfico de cada uno de los libros que forma parte del Sistema Bibliotecario de la UNAM.
Para realizar consultas, la forma de recuperación se da a través de diferentes llaves de búsqueda, por ejemplo: autor, título, tema o clasificación, además permite búsquedas combinadas, es decir, la unión de una o más llaves de búsqueda, con el objeto de lograr una búsqueda más selectiva, reduciendo el tiempo de búsqueda. El resultado de la búsqueda contiene información detallada del material solicitado(autor, título, tema, clasificación y ubicación física) .
Actualmente se tienen registrados más un total de 721,000 títulos y un acervo aproximado de 5'000,000 de ejemplares.

SERIUNAM. Contiene el registro bibliográfico de publicaciones periódicas (revistas), que se encuentran en bibliotecas dentro y fuera de la UNAM. La recuperación de registros se realiza también por llaves de búsqueda como título, serie, o clasificación. En la actualidad esta base de datos consta de más de 49,500 títulos y 5'700,000 fascículos aproximadamente.

TESIUNAM. Esta base de datos contiene el registro de las tesis de egresados de la UNAM así como de diferentes universidades; las llaves de búsqueda para la recuperación de la ficha bibliográfica son: autor, título, escuela, asesor y tema. En la actualidad esta base consta de más de 266,200 registros.

PRÉSTAMO. Contiene el registro de todos los lectores registrados en la biblioteca central, ocupándose de llevar en forma automatizada el control de préstamos a domicilio.

ADQUISICIONES. En esta base de datos, se encuentran el registro de cada biblioteca perteneciente a la UNAM en materia de presupuesto asignado para la compra de libros indicando entre otras cosas, el presupuesto ejercido y por ejercer, facturación, y control de proveedores.

MAPAMEX. Contiene el registro de los planos y mapas geográficos del país a escalas variables. Esta base también contiene registros de otras dependencias que no pertenecen a la UNAM.

La figura 1.1 muestra en forma esquemática las diferentes bases de datos existentes en la Biblioteca Central.

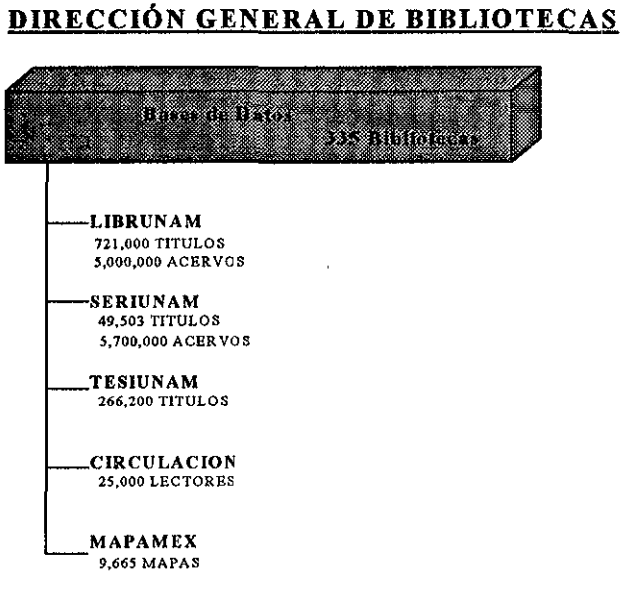


Figura 1.1 Bases de datos de la Dirección General de Bibliotecas.

Las aplicaciones que se utilizan en el manejo de las bases de libros, revistas, tesis, así como demás utilidades se encuentran desarrolladas y almacenadas en equipos de cómputo SPARC SUN con que cuenta la Dirección General de Bibliotecas.

En estos equipos se realizan las tareas de administración de las bases de datos, control de aplicaciones y prestación de servicios, las cuales son:

- Administración de usuarios.
- La elaboración de etiquetas de clasificación de libros.
- Elaboración de tarjetas catalográficas.
- Elaboración de listados de acervo bibliográfico.
- Servicios de impresión.
- Conexión remota a través de Internet para realizar consultas en línea desde cualquier parte del mundo.
- Actualización automática de registros bibliográficos a través de conexiones remotas.
- Correo electrónico.
- Servicio de FTP anónimo.

1.2 Planteamiento del Problema

La Dirección General de Bibliotecas (DGB) cuenta con 7 servidores SPARC SUN, los cuales se utilizan principalmente para dar servicio de consulta de sus bases de datos a nivel nacional ya sea vía Internet o con sesiones remotas. Esto nos obliga a esforzarnos para dar el mejor servicio como administradores de los servidores que se encuentran en la DGB.

Por ello, surge la idea de realizar un sistema que automatice las tareas de monitoreo de los estados de los servidores y que reporte periódicamente las anomalías del sistema a través de una computadora personal y haga la función de un operador humano observando y reportando los errores que existan en los sistemas de cómputo. La PC con el software de monitoreo integrado, será instalada en un lugar que permita a los operadores o administrador del sistema escuchar los reportes generados por el sistema de monitoreo de mensajes de error dados por el sistema. Así de igual manera evita que el operador se encuentre en un medio ambiente con bajas temperaturas que pueda ocasionar malestar o enfermedad.

Los elementos que se utilizan para el desarrollo del proyecto son: sistema operativo SunOS Unix, herramientas de programación en shell UNIX, Visual "C", lenguaje "C" y Windows; para la comunicación entre equipos de cómputo usamos el conjunto de protocolos TCP/IP implementado en UNIX y Windows, bibliotecas DLLs para comunicación y síntesis de voz específicamente text-to-speech (TTS).

Como base para obtener información del estado de los servidores nos ayudaremos con los archivos de registro de errores del sistema UNIX, por ejemplo el *messages*, que se encuentra en la ruta /var/adm de cada equipo SPARC SUN (o también conocidos como servidores por las funciones que realizan) de la DGB.

Con este archivo podremos manipular los errores diarios del sistema y crear un reporte periódico del estado de los servidores, con un programa que llamaremos cliente que este monitoreando la información del archivo messages y desde una computadora personal que será el servidor, preguntaremos al cliente si existe algún reporte, para activar la alarma la cual será voz sintética que nos dé el reporte automáticamente.

1.3 Esquema General

Como ya hemos mencionado, el objetivo de nuestro proyecto es monitorear los servidores UNIX misma actividad que observamos de manera gráfica en la figura 1.2; dentro de este esquema se podrá ver la disposición de los dispositivos usados para ello.

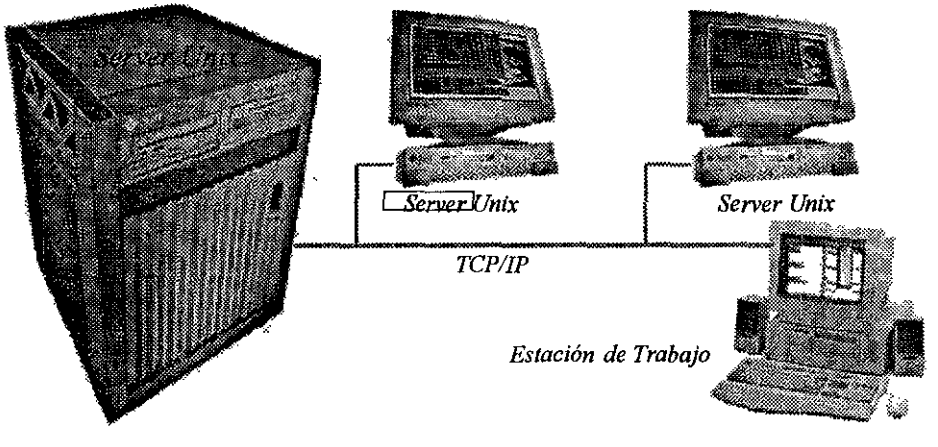


Figura 1.2 Esquema General de Monitoreo

Para tener una idea clara de cada uno de estos dispositivos (servidores UNIX, computadoras personales y protocolos de comunicación) que se encuentran en el esquema general antes visto, daremos un panorama más desglosado de ellos en los apartados siguientes.

1.4 Computadoras Personales

1.4.1 Definición

Una computadora es un dispositivo o sistema capaz de realizar una secuencia de operaciones en una forma definida explícitamente. Con frecuencia, las operaciones son cálculos numéricos o tratamiento de datos, pero también comprenden una entrada y salida; las operaciones dentro de la secuencia pueden depender de valores particulares de datos. La definición de la secuencia es denominada programa almacenado o cableado. El primero puede estar en una memoria RAM o ROM.

Las características típicas de las computadoras son la velocidad, memoria interna y el programa almacenado. La velocidad de la computadora es resultado de sus circuitos electrónicos y también de la memoria que contenga, ver figura 1.3.

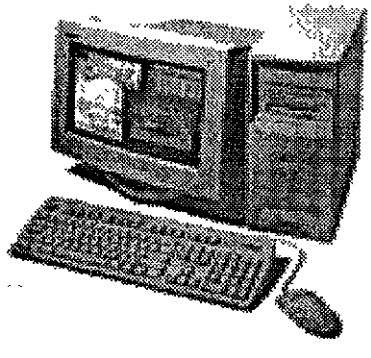


Figura 1.3 Computadora Personal

A continuación haremos un breve resumen del nacimiento y avance tecnológico de la computadora de acuerdo a la división por generaciones, así como las partes que la componen según la definición anterior.

1.4.2 Generaciones de la Computadora

Las primeras cuatro generaciones de la computadora se diferencian por sus componentes electrónicos. Existen dos posibles generaciones, las cuales se caracterizan por las aplicaciones avanzadas.

Primera Generación (1951 -1958)

- Usaban tubos al vacío para procesar información.
- Usaban tarjetas perforadas para entrar los datos y los programas.
- Usaban cilindros magnéticos para almacenar información e instrucciones internas.
- Eran sumamente grandes, utilizaban gran cantidad de electricidad, generaban gran cantidad de calor y eran sumamente lentas.
- Se comenzó a utilizar el sistema binario para representar los datos.

Segunda Generación (1958 - 1964)

- Usaban transistores[1] para procesar información.
- Los transistores eran más rápidos, pequeños y más confiables que los tubos al vacío.
- 200 transistores podían acomodarse en la misma cantidad de espacio que un tubo al vacío.
- Usaban pequeños anillos magnéticos para almacenar información e instrucciones.
- Se mejoraron los programas de computadoras que fueron desarrollados durante la primera generación.
- Se desarrollaron nuevos lenguajes de programación como COBOL y FORTRAN, los cuales eran comercialmente accesibles.
- Se usaban en aplicaciones de sistemas de reservaciones de líneas aéreas, control del tráfico aéreo y simulaciones de propósito general.
- Surgieron las minicomputadoras y los terminales a distancia.
- Se comenzó a disminuir el tamaño de las computadoras.

Tercera Generación (1964 - 1971)

- Se desarrollaron circuitos integrados para procesar información.
- Se desarrollaron los "chips" para almacenar y procesar la información.
- Un "chip" es una pieza de silicio que contiene los componentes electrónicos en miniatura llamados semiconductores.
- Los circuitos integrados recuerdan los datos, ya que almacenan la información como cargas eléctricas.
- Surge la multiprogramación.
- Las computadoras pueden llevar a cabo ambas tareas de procesamiento o análisis matemáticos.

[1] Transistor, en electrónica, denominación común para un grupo de componentes electrónicos utilizados como amplificadores u osciladores en sistemas de comunicaciones, control y comunicación.

- Emerge la industria del "software".
- Se desarrollan las minicomputadoras IBM 360 y DEC PDP-1.
- Otra vez las computadoras se tornan más pequeñas, más ligeras y más eficientes.
- Consumían menos electricidad, por lo tanto, generaban menos calor.

Cuarta Generación (1971 - 1984)

- Se desarrolló el microprocesador.
- Se colocan más circuitos dentro de un "chip".
- "LSI - Large Scale Integration circuit".
- "VLSI - Very Large Scale Integration circuit".
- Cada "chip" puede hacer diferentes tareas.
- Un "chip" sencillo actualmente contiene la unidad de control y la unidad de aritmética/lógica. El tercer componente, la memoria primaria, es operado por otros "chips".
- Se reemplaza la memoria de anillos magnéticos por la memoria de "chips" de silicio.
- Se desarrollan las microcomputadoras, o sea, computadoras personales o PC.
- Se desarrollan las supercomputadoras.

Quinta Generación (1984 – 1990)

Se caracterizó principalmente por la aceptación del procesamiento en paralelo. Hasta ese tiempo el paralelismo fue limitado a entubamiento y procesamiento vectorial o pequeños trabajos de procesamiento compartido. La quinta generación vio la introducción de máquinas con cientos de procesadores que podían estar trabajando en partes diferentes de un programa. La escala de integración en semiconductores continuó a pasos increíbles -por 1990 fue posible construir chips con un millón de componentes y memorias semiconductoras transformándose en un estándar en todas las computadoras; así como también por:

- Inteligencia artificial
- Robótica
- Sistemas expertos
- Redes de comunicaciones

Sexta Generación (1990-)

La transición de generaciones entre tecnología de computadoras son difíciles de definir especialmente cuando ellos están tomando lugar.

Desde 1990 los adelantos surgidos durante este tiempo reflejan mejoramientos graduales sobre sistemas ya establecidos, ello representa una transición a una nueva "generación", pero otros desarrollos comprobarán ser cambios significativos para esta.

Algunos avances de esta generación son:

- Combinación de arquitecturas paralelas/vector
- Crecimiento explosivo de las redes de área amplia

1.4.3 Arquitectura de las Computadoras

El término arquitectura de computadoras hace referencia a la naturaleza de las diferentes unidades funcionales que componen al hardware, así como a la forma en que dichos agrupamientos de componentes electrónicos se interrelacionan para proporcionar uso al equipo.
Aquí veremos la arquitectura del Matemático J. Von Neumann por que de todas las arquitecturas es la que mayor tiempo ha influenciado a la industria de cómputo, en la figura 1.3 mostraremos un diagrama de bloques.

Los elementos funcionales de una computadora son:

- ***La unidad de aritmética-lógica***, encargada de efectuar las operaciones sobre los datos.
- ***La unidad de control***, encargada de dirigir la operación de la computadora e interpretar los códigos binarios con los que se expresan las instrucciones, y posteriormente ejecutarlas. Los principales componentes que forma el CPU y la forma en que trabajan es: La unidad de control identifica el lugar donde se encuentra la instrucción a ejecutar, para ello cuenta con el registro program counter o PC (contador del programa) que le indica la ubicación en la memoria (dirección de memoria), de la instrucción. Una vez determinada la dirección, la instrucción es tomada de la memoria principal (auxiliándose de los registros de dirección de memoria y datos de memoria) y llevada a la unidad de control, colocándose en el denominado registro de instrucciones (IR); una vez hecho esto, la unidad de control identificará (decodificará) la instrucción que corresponde a la instrucción obtenida y guiará su desarrollo. Finalmente la unidad de control "ajusta" al PC, para conocer la dirección de la siguiente instrucción y repite las etapas descritas anteriormente.
- ***La unidad de memoria*** (almacenamiento), donde se guardan tanto los datos como las instrucciones que se efectuarán sobre ellas.
- ***La unidad de entrada/salida*** (E/S, o I/O por sus siglas en inglés), encargada del procesos de comunicación de la computadora con su entorno.

Es común que se considere a la unidad de control integrada(física y lógicamente) con la unidad aritmética-lógica, formando lo que se ha dado en llamar ***Unidad Central de Procesamiento*** CPU.

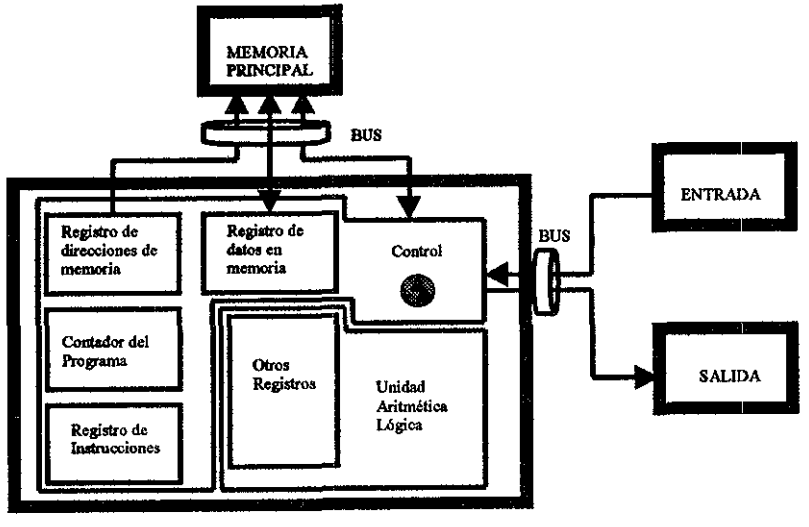


Figura 1.4 Diagrama general por bloques de la arquitectura propuesta por J. Von Neumann.

La arquitectura de J. Von Neumann fue dirigida hacia el empleo compartido de memoria principal y los periféricos; también introdujo la capacidad de interrumpir al CPU.

1.5 Equipo de cómputo SPARCstation

Las computadoras **SPARCstation** son estaciones de trabajo(workstation) basadas en procesadores con tecnología RISC (Reduced Instructions Set Computer). Las características de estas estaciones de trabajo las hacen atractivas para ser usadas como servidores de una red local; éstas pueden ejecutar hasta 15 MIPS (millones de instrucciones por segundo) y contar con memoria de 32 Megabytes a 4 Gigabytes. Esta velocidad permite aplicaciones gráficas, o el manejo de procesos simultáneos en modo multiusuario.

Las workstation están diseñadas para trabajar en ambiente de red, y cuentan con tarjeta Ethernet con entrada para conexión con RJ-45; el protocolo de comunicación que utilizan

es el TCP/IP, que tiene la posibilidad de comunicar máquinas de diferentes arquitecturas. Las estaciones de trabajo cuentan con sistema operativo SunOS versión 5.6, el cual está constituido por un conjunto de programas y archivos que controlan las operaciones de las workstation.

CARACTERÍSTICAS

- Memoria RAM de 32MB a 4 GB.
- Tarjeta Ethernet.
- Unidad de cinta de respaldo con capacidad de 4 y 8 Mbytes y unidad de CD ROM.
- Monitor gráfico alta resolución.

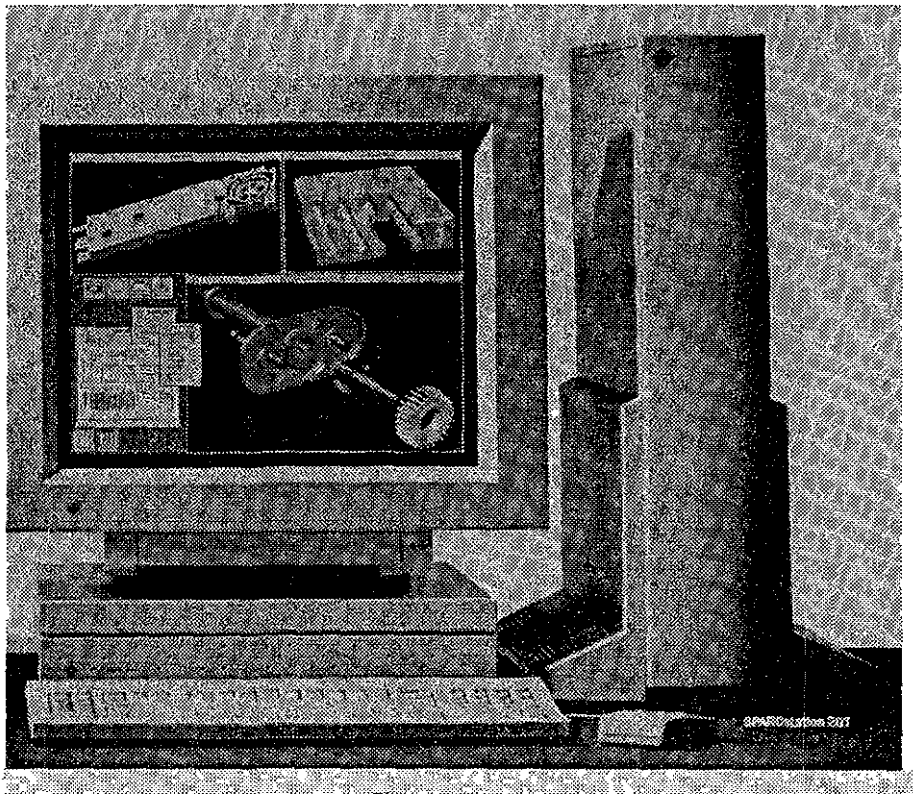


Figura 1.5 SPARCstation

La SPARCstation cuenta con una herramienta importante que ofrecen los fabricantes del equipo, que es el Network File System(NFS) el cual permite que desde cualquier computadora se puedan utilizar los discos del equipo SPARC. Además se ofrece la realización de procesos multitarea desde un ambiente gráfico como lo es OPENWINDOWS que facilita la administración de la estación de trabajo.

1.6 Sistema Operativo UNIX

1.6.1 Historia del Sistema Operativo UNIX

UNIX se originó en los laboratorios Bell de la compañia AT&T (American Telephone and Telegraph), una de las instituciones de investigación mejor dotadas de los Estados Unidos; su historia es casi única en comparación con otros sistemas operativos, debido a que los avances son en gran parte aportaciones de personas con ideas creativas. La implicación es que los avances no se han dado por decisiones burocráticas sino más bien de las necesidades y creatividad de los usuarios. El sistema UNIX fue diseñado por un grupo de personas que eran investigadores de AT&T en el desarrollo de un sistema operativo llamado **MULTICS**, a finales de los sesenta.

Como uno de los primeros sistemas de tiempo compartido, MULTICS incorporó la mayoría de las ideas que aparecen en los sistemas multitarea actuales. Desgraciadamente, MULTICS sufrió las consecuencias de su papel innovador y resultó mucho más complejo y pesado de lo que era necesario. A finales de los sesenta AT&T abandonó la mayor parte de su participación en el proyecto MULTICS, dejando a un grupo de personas con talento y con muchas ideas acerca de lo que un sistema de tiempo compartido debería ser.

Sin acceso al sistema MULTICS, estas personas se quedaron sin un sistema operativo moderno con el cual trabajar, de modo que crearon uno nuevo. Los investigadores Ken Thompson y Dennis Ritchie construyeron el sistema basado en un diseño elaborado con Rudd Canaday. Pronto se les unieron J.F. Ossana y R. Morris. Tras un periodo de discusiones, adquirieron una computadora DEC PDP-7 de desecho y se pusieron a trabajar.

Como muchos de los mejores proyectos, esto comenzó con la creación de un juego. Thompson y Ritchie desarrollaron un juego de viaje espacial para la PDP-7.

Después de esta experiencia, crearon una nueva estructura de sistemas de archivos y un nuevo software que es muy similar al sistema de archivos moderno. Le añadieron un entorno de procesos con planificación y completaron el resto de un sistema operativo rudimentario. El nombre UNIX pronto se aplicó a los resultados ya que su trabajo fue una simplificación del sistema MULTICS. El sistema estuvo operando sobre la PDP-7 a

principios de los setenta, y a mediados de esta década habían pasado el proyecto a una máquina DEC PDP-11 de reciente aparición.

Muchas de las ideas claves del UNIX moderno estaban presentes en las primeras versiones, incluyendo el sistema de archivo, la implantación de procesos y los comandos en línea aún utilizados hoy en día.

El desarrollo original fue codificado en lenguaje ensamblador, pero pronto se desarrolló el lenguaje de programación C dentro del grupo, empezando en 1971. El lenguaje C fue utilizado casi inmediatamente en la continuación del desarrollo del sistema UNIX, y en 1973 el núcleo se recodificó en C. Hoy sólo unas cuantas subrutinas del núcleo de alto rendimiento están escritas en lenguaje ensamblador. Éste fue el primer intento de codificar un sistema operativo entero en un lenguaje de alto nivel y la portabilidad que se consiguió está ampliamente considerada como una de las razones principales de la popularidad que el sistema UNIX actualmente goza.

El sistema UNIX captó inmediatamente la atención de los informáticos en los laboratorios Bell, y después de dos o tres años había alrededor de una docena de sistemas UNIX ejecutándose en varias máquinas diferentes. Se realizaron con frecuencia importantes mejoras de software y AT&T continuó su desarrollo dentro de los laboratorios Bell. El programa troff apareció durante este periodo, entre muchas otras innovaciones.

Sin embargo, el sistema UNIX adquirió cuerpo con el desarrollo de las máquinas PDP-11 superiores, tales como la PDP-11/45 y la PDP-11/70, a principios y mediados de los setenta. El sistema UNIX se ajustaba de forma natural a la arquitectura DEC y ocasionó la venta de muchos cientos de máquinas PDP-11 a lo largo de los años. Los programadores dentro de los Laboratorios Bell empezaron a utilizar máquinas UNIX para su trabajo de procesado de textos, y los diseñadores de productos de los laboratorios Bell comenzaron a utilizar la PDP-11 con sistemas UNIX para sistemas dentro del negocio telefónico.

Simultáneamente, AT&T remitió muchas copias del sistema UNIX a todas las Universidades del mundo. Esto dio lugar a otra fértil ola de innovaciones y el sistema ampliamente utilizado UNIX BSD (Berkeley Software Distrubution) apareció en la Universidad de California en Berkeley. Al tiempo que AT&T fortalecía el sistema UNIX y lo optimizaba en la dirección de la computación comercial, las versiones BSD resultaban dominantes en las comunidades universitarias y técnicas.

A finales de los setenta, AT&T comenzó un nuevo esquema de nominación para su versión del sistema UNIX. Anteriormente las versiones principales se designaban según las nuevas versiones que salían del área de investigación, y dos de las más populares fueron las denominadas Revisión Sexta y luego Revisión Séptima. Siguiendo una reorganización interna del soporte del sistema UNIX, AT&T cambió su numeración a Sistema III y Sistema V. Realmente estas nuevas versiones eran descendientes directas de la Revisión Séptima y el Sistema V reemplazó al Sistema III a mediados de los ochenta. El Sistema IV fue utilizado internamente en los Laboratorios Bell, pero se consideró un producto de transición que nunca fue liberado.

Conforme los microcomputadores se han desarrollado en velocidad y potencia y su costo ha disminuido, estas máquinas se han movido al rango del sistema UNIX. Las máquinas 8088 originales eran casi lo bastante potentes para soportar al sistema UNIX y algunas versiones podrían ejecutarse sobre estas máquinas. El Sistema operativo XENIX es una versión adelgazada del sistema UNIX para IBM PC; esta versión se ejecuta en máquinas 80286 y 80386.

Recientemente, los Laboratorios Bell han desarrollado una nueva versión genérica denominada Revisión Octava o sistema UNIX de investigación. Aunque no se venda comercialmente, esta versión ha sido ampliamente distribuida a universidades.

Los descendientes de las versiones BSD están siendo constantemente mejorados y las realizaciones conjuntas entre AT&T y Microsoft, AT&T y Sun, y AT&T y Amdahi están permitiendo integrar más extensamente las versiones para microcomputadoras y supercomputadoras. Finalmente se espera que las versiones SVR3 (Sistema V Revisión 3), BSD, XENIX converjan en una versión única del sistema UNIX que pueda ejecutarse en casi cualquier entorno hardware. Este producto combinado podría también permitir la compatibilidad de código objeto entre diferentes versiones para la misma máquina.

1.6.2 UNIX DE AT&T

Paradójicamente, AT&T no liberó formalmente su versión de UNIX 1982, años después de que se distribuyeron el XENIX y la versión 4.1 de BSD. Al primer lanzamiento comercial se le llamó UNIX System III, que se basó principalmente en la versión 7 y en algunas características de programación de la versión 6. En 1983 se liberó el UNIX System V que incluía importantes utilidades de BSD. Se incorporó el proceso init de inicio de tareas, siendo diferente el procedimiento al de la versión 7.

AT&T liberó el UNIX Sistema V Revisión 2 (SVR2) en 1984, introduciendo una versión propia de la base de datos Termcap, llamada Terminfo, la cual consiste en un serie de archivos que describen las capacidades de cada modelo y tipo de terminal. Otros cambios incluyeron modificaciones menores al sistema jerárquico de archivos, y la utilidad Remote File System en respuesta al NFS de Sun. El actual UNIX Sistema V Revisión 3 (SVR3) es la correspondiente de las plataformas Intel y la base de los ambientes gráficos para UNIX.

1.6.3 BSD (Berkeley Software Distribution)

En 1974, el campus Berkeley de la Universidad de California se involucró en el desarrollo del **UNIX** cuando el Profesor Faby adquirió la versión 4. En 1975, Ken Thompson visitó esta Universidad, su Alma Mater, y ayudó a instalar la versión 6 en una PDP-11/70. El mismo año, dos graduados llegaron a Berkeley: Bill Joy y Chuck Haley, quienes tuvieron un papel determinante en el desarrollo del sistema. Ellos y Thompson trabajaron en un compilador en Pascal y un editor llamado EX, y posteriormente volcaron su interés en las operaciones internas del kernel. A este sistema se le dio el nombre de Berkeley Software Distribution.

Posteriormente Bill Joy siguió trabajando sobre el EX para añadirle capacidad de direccionamiento del cursor sobre terminales y producir además el C Shell, que se llamó así por su similitud con el ambiente de programación **C**. En 1978 se actualiza la organización interna del sistema, llamándola Second Berkeley Software Distribution, que también se conoce como 2BSD.

1.6.4 El Sistema Operativo Xenix De Microsoft

El sistema operativo **XENIX** está basado en la versión 7 de AT&T. Microsoft liberó el **XENIX** 2.3 en 1980 como una versión para microcomputadoras. De la misma manera que la versión 7, el **XENIX** tomo algunas utilidades de la 4.1 BSD.

La versión 3.0 de **XENIX** incorporó algunas características del Sistema III y el **XENIX** 5.0 se diseñó tratando de cumplir con los estándares de la definición de interfaces de Microsoft. Se logró que **XENIX** y **UNIX** convergieran en un solo producto al cual se le denominó SCO **UNIX**.

1.6.5 Sistema Operativo Unix

El sistema UNIX es un sistema operativo multitarea, esto significa que puede administrar la ejecución simultánea de varios procesos, por ejemplo, la computadora puede ejecutar sus procesos mientras ejecuta procesos de cualquier otro usuario que esté conectado al equipo de cómputo, de forma que parece simultánea la ejecución de las tareas. En realidad la computadora divide la atención entre todas las peticiones de ejecución de procesos de múltiples usuarios; a este tipo de funcionamiento se le llama de tiempo compartido. Durante el desarrollo del sistema operativo UNIX han surgido diferentes versiones de diferentes fabricantes; esto ha ocasionado que algunas aplicaciones no siempre puedan

ejecutarse en diferentes equipos con diferentes versiones de UNIX, y es por ello que se busca la estandarización de este sistema, dando como resultado la versión de sistema operativo UNIX Sistema V versión 4. Dentro de las características principales del sistema operativo se pueden mencionar las siguientes:

- Es un sistema operativo multiusuario y multitarea
- Las especificaciones de diseño están disponibles públicamente, lo cual hace que se adapte a exigencias particulares
- Es un sistema operativo escrito en lenguaje de alto nivel, lo que lo hace portable
- Está enfocado al desarrollo de programas y manejo de grandes volúmenes de información
- Cuenta con redireccionamiento, filtros e interconexión entre los comandos
- Cuenta con un sistema de archivo sencillo y eficiente
- La interfaz con los archivos y con los dispositivos se manejan igual que un archivo

1.6.6 Estructura de Directorios UNIX

En el Sistema **UNIX** los archivos se disponen en directorios jerárquicos. La disposición de la estructura de directorios del sistema **UNIX** se ha cambiado en el Sistema V versión 4 para acomodar los entornos de red y la compartición de archivos remotos. Los archivos de directorios se organizan en forma de árbol y se dividen en:

- Archivos necesarios para la carga del sistema.
- Archivos compartidos que permanecen estáticos a lo largo de la vida del sistema.
- Archivos de usuario.
- Archivos y directorios del sistema que cambian a lo largo de la vida del sistema.

La estructura de directorios está diseñada como se describe a continuación:

/etc, este directorio contiene programas, archivos y tablas para inicializar el sistema. En él se encuentra el archivo passwd el cual es usado para registrar a los usuarios del sistema. No es permitido almacenar datos en este directorio por parte de los usuarios.

/lib, este directorio contiene bibliotecas con código binario usado por el compilador C.

/bin, este directorio contiene varios de los programas de utilidades que **UNIX** requiere. Cualquier programa requerido para el proceso de inicialización del equipo de cómputo debe estar contenido aquí.

/usr, este directorio contiene datos y programas de usuarios y también se encuentran otros directorios como /usr/bin, /usr/include, /usr/include/local.

/home, es el directorio para que los usuarios guarden su información.

/var, este directorio contiene las tablas del sistema y archivos transitorios, relacionados con algunos cuentas.

1.6.7 Administración del Sistema

La versión 4 incluye nuevas características para hacer más fácil la administración y el mantenimiento del sistema. Se ha simplificado la interfaz de menú para la administración del sistema. Se han hecho mejoras sobre los procedimientos de seguridad y restauración del sistema. Se ha simplificado la instalación del software, con respecto a las versiones anteriores a ésta. Quizá la diferencia más visible e interesante respecto a las primeras versiones es que la versión 4 ofrece interfaces de usuario estándar y sistemas de ventanas para aplicaciones basadas en gráficos.

La interfaz gráfica de usuario incluida con la versión 4 se denomina OPEN LOOK, que ofrece una forma consistente, efectiva y eficiente de interactuar con las aplicaciones. Proporciona varias posibilidades de manejo y operación de ventanas, botones, menús, ventanas de presentación y facilidades de ayuda. Con esta interfaz los usuarios nuevos pueden aprender rápidamente las aplicaciones. Se proporciona una caja de herramientas gráficas que puede utilizarse para construir aplicaciones.

1.6.8 Desarrollo de Aplicaciones

Los cambios de la versión 4 que soportan el desarrollo de aplicaciones incluyen a las ampliaciones del sistema de compilación del lenguaje C y a las bibliotecas C. El sistema de compilación del lenguaje C contiene un compilador para ANSI C, la versión del lenguaje de programación C estandarizada por ANSI. El compilador también acepta código de versiones anteriores de C contenidas en el Sistema V.

1.6.9 Manejo de Memoria

La versión 4 incluye ampliaciones para el manejo de memoria, un punto de interés para los programadores. La arquitectura de la memoria virtual de la versión 4 (VM) incorporada del SunOS permite a los programadores hacer un uso más eficiente de la memoria principal. Se pueden ejecutar programas mayores que la memoria física del sistema, y el espacio del disco se puede utilizar de una forma flexible. Entre otras características se incluyen los archivos proyectados, que pueden utilizar los programadores de aplicaciones para manipular archivos como si estuviesen en la memoria principal

1.6.10 Unix en Red

Las redes son una de las áreas claves de las mejoras de la versión 4. Se han incorporado muchas capacidades de red de los sistemas BSD y SunOS. Las capacidades de red del sistema BSD incluyen los protocolos TCP/IP, los cuales son utilizados para transferencia de archivos, conexión remota y ejecución remota. También la versión 4 incluye la interfaz de red de BSD, que se utiliza para construir aplicaciones basadas en red. Del SunOS, la versión 4 ha incorporado características de redes que incluyen al sistema de archivos de red (NFS) para archivos remotos compartidos, el protocolo estándar Remote Procedure Call (RPC), para ejecución de un procedimiento sobre una computadora remota, y el External Data Representation (XDR), que especifica un formato para los datos que permite su intercambio entre sistemas que incluso tengan diferente arquitectura de hardware, diferente sistema operativo y diferente lenguaje de programación.

1.6.11 Versiones y marcas de equipos que trabajan con UNIX

Las versiones más importantes de UNIX que actualmente se encuentran en el mercado se muestran en la siguiente tabla, junto con los fabricantes de éstas.

FABRICANTE	SISTEMA OPERATIVO
AT&T	UNIX SYSTEM V
HEWLETT-PACKARD	HP/UX
IBM	AIX
NEXT	NEXT STEP
OSF	OSF/1
SANTA CRUZ OPERATION	SCO UNIX
SUN SOFT	SOLARIS
UNIVEL	UNIXWARE

1.6.12 Fabricantes de Workstations

Las distintas marcas de workstations, o estaciones de trabajo, tienen dos características que las hacen comunes: todas trabajan con sistema operativo **UNIX** en sus diferentes versiones y además poseen una interfaz gráfica que las distingue de las mainframes que trabajan con **UNIX**

Los principales fabricantes de workstations son los siguientes:

Sun Microsystems que es el fabricante más grande de workstations con un alcance del 38% en el mercado. El procesador RISC que utiliza se llama SPARC y SUN ha intentado convertirlo en estándar en el mercado por medio de las licencias de su tecnología a otros fabricantes como Fujitsu y Tatung.

HEWLETT-PACKARD (HP) adquirió a la empresa APOLO y su tecnología, otro fabricante de workstations, y está desarrollando sistemas con ésta y la suya propia. Su línea de productos Snake está basado en un circuito RISC propio llamado PA (Precision Architecture). Actualmente este circuito es uno de los más rápidos del mercado, superando los 70 MIPS (Millones de Instrucciones Por Segundo). HP también está buscando aliados en el uso de su circuito con empresas como HITACHI. En 1991 alcanzó el 20% del mercado.

DEC (DIGITAL EQUIPMENT CORPORATION) cuenta con una línea de workstation llamada Decstation, basada en el circuito con arquitectura RISC de la empresa ACE(Advance Computer Environment), la cual produce copias del circuito MIPS400. Actualmente DEC liberó un circuito llamado Alpha de 64 bits con posibilidad de superar los 200 MIPS.

IBM entró tarde al mercado de las workstations con su equipo RS-600 basado en un circuito con tecnología RISC diseñado por IBM llamado Power Architecture. IBM también ha hecho alianzas con Apple Computers y Wang para incrementar la venta de su tecnología que actualmente cuenta sólo con el 9% del mercado, pero esta participación va en aumento.

1.7 Síntesis de Voz (text-to-speech)

La verdadera síntesis del habla no necesita voz humana almacenada; es generada (sintetizada) esencialmente desde la nada. Usando conocimiento acústico, fonético, y lingüístico, un texto ordinario tal como el que esta leyendo ahora puede convertirse en voz. Este proceso es llamado texto-a-voz /text-to-speech/ y, porque son muchas las reglas usadas, también es referido como síntesis-por-regla /synthesis-by-rule/ .

La tarea de los sistemas texto-a-voz es convertir un texto entero en un discurso que represente justamente el sentido que conlleva el texto. Será considerado útil. El discurso resultante, si logra un grado alto de inteligibilidad y naturalidad. Además, si puede entregarse una interpretación vivaz en lugar de una torpe, el grado de aceptación de los usuarios será mayor.

Entre varios esquemas de texto-a-voz que se han probado hay algunos que concatenan palabras pregrabadas, pero éstos están siempre limitados a un vocabulario seleccionado por adelantado. Sistemas de texto-a-voz que generan voz humana sintética por concatenación de subpalabras segmentadas (difonos) tales como sílabas, morfemas, fonemas, difonemas o trifonemas generalmente aceptan un texto no restringido, pero, si bien ellos podrían tener un ritmo sonoro y entonación natural, su calidad de interpretación varía ampliamente. El resultado puede ser altamente inteligible, especialmente después de ganar experiencia escuchando, pero ningún sistema texto-a-voz ha sido capaz de producir resultados comparables con sistemas de almacenamiento de palabras de moderada calidad. El discurso resultante del sistema texto-a-voz tiende a sonar mecánico y definitivamente afectado.

1.7.1 Sistemas de conversión de texto a voz por síntesis por regla

El concepto de ***síntesis por reglas*** nació a finales de los años cincuenta en los Laboratorios Haskins en Estados Unidos. El proyecto inicial consistía en desarrollar el aprendizaje de la lectura de espectrogramas en sordos de modo que pudieran comprender la lengua hablada sin necesidad de recurrir a la utilización del lenguaje de los signos o a transcripciones escritas; esta idea inicial desembocó en una investigación sobre la estructura acústica del habla y, en última instancia, sobre la codificación de la información acústica que nos permite interpretar como un código lingüístico los mensajes sonoros que recibimos (Potter - Kopp - Green, 1947). El equipo de Haskins desarrolló, intentando encontrar la información acústica esencial para la comprensión del habla, un método para la generación artificial de enunciados.

El llamado Pattern Playback es un instrumento que "lee en voz alta" representaciones de la evolución de la frecuencia y la amplitud de la onda sonora en función del tiempo (espectrogramas) que dibuja el propio investigador. El objetivo de este trabajo consistía en eliminar toda la información no significativa que se encuentra en las señales acústicas del habla a fin de reducirla a sus unidades mínimas. Se desarrollaron así una serie de reglas para crear las representaciones espectrográficas que, producidas según este conjunto de reglas, dieran como resultado un mensaje identificable. Con un conjunto finito y limitado de reglas pudo llegar a sintetizarse cualquier frase (Liberman 1959).

Es importante señalar la necesidad de trabajar con unidades menores que los fonemas. Por otra parte, las investigaciones en otros campos, especialmente en fonología estructuralista, llevaron al concepto de rasgo distintivo, unidad en las que puede dividirse el fonema. La conjunción de las investigaciones en fonética acústica y en teoría lingüística llevó a comprobar la necesidad de recurrir a elementos subfonémicos para explicar de qué manera se codifica un mensaje en la onda sonora, ya que un determinado fonema no mantiene una relación biunívoca con un único punto de la cadena sonora (Jakobson - Fant - Halle, 1952; Jakobson - Halle, 1956).

Simultáneamente el científico sueco C.G. Fant desarrolló un modelo acústico de producción del habla (Fant, 1960) que constituye la base de muchos de los sintetizadores actuales. La idea esencial consiste en considerar el tracto vocal como un filtro variable y en

tratar la señal acústica producida como el resultado de la interacción entre una fuente y un filtro. Esto hizo posible la descripción del habla a partir de un conjunto limitado de parámetros relacionados con el comportamiento acústico del tracto vocal a la vez que permitió disponer de un modelo fácilmente implementable sea en forma de circuito electrónico o de programa.

Desde el momento en que fue posible describir el habla como una representación parametrizada y contando con un cuerpo de conocimientos sobre las propiedades acústicas de las señales sonoras en relación con unidades lingüísticas como los rasgos distintivos, no había, en teoría, barreras para poner a punto un mecanismo que transformara en una representación sonora los valores de los parámetros acústicos relacionados con el habla, sistema que constituye la base de un conversor de texto a voz.

Probablemente fue Jonathan Allen (1976) uno de los primeros investigadores en sugerir la idea de llevar a cabo la síntesis automática de un texto escrito sin restricciones de ninguna clase. Este proyecto contribuyó a establecer un conjunto de reglas que convierten las representaciones ortográficas convencionales en su representación fonológica o alofónica como primer paso para su conversión en sonido. Hay que destacar que las concepciones de Allen estuvieron fuertemente influidas por el desarrollo de la lingüística generativa en el Massachusetts Institute of Technology: el conversor de texto a voz por él realizado se concibe como una transformación entre dos formas superficiales - texto escrito y cadena sonora - que se lleva a cabo a partir de una representación lingüística subyacente común a ambas. Las reglas de conversión de letra a fonema utilizaron al principio los trabajos en fonología generativa de Chomsky y Halle (1968), hasta integrar en las últimas versiones del sistema los resultados de las investigaciones sobre fonología y morfología que se llevan a cabo en el marco de diversos modelos lingüísticos de inspiración chomskiana. Esta interacción entre la teoría lingüística y las necesidades de la tecnología se ha llevado a cabo de forma bidireccional: en ciertos casos la necesidad de explicitar las reglas ha llevado a estudiar en profundidad ciertos problemas de la lengua para los que no se disponía de una descripción sistemática.

1.7.2 Aplicaciones para sistemas hechos con síntesis por regla

Las posibilidades de interacción con las PC's mediante la voz es tal vez uno de los temas privilegiados dentro de la investigación actual en el campo de la comunicación hombre-*máquina*.

Las ventajas de la salida vocal en las PC's es(LListerri,1985; Nadeu - Mariño, 1985; Witten, 1982):

- Libertad para la realización de otras tareas: cuando la PC nos hace llegar sus mensajes mediante la voz podemos tener las manos y la vista libres y el acceso a la información no está limitado a la existencia de un entorno bien iluminado; una salida sonora es omnidireccional y permite una mayor movilidad en el trabajo; por otra parte, los mensajes orales generados por la máquina no interfieren con otras actividades.

- Acceso telefónico. Para acceder por vía telefónica a una terminal no se necesita un equipo complejo y se trata además de un método rápido e independiente de la distancia.

- Economía. Las salidas vocales requieren únicamente componentes electrónicos que pueden fabricarse a bajo costo. Desde el punto de vista del ahorro de tiempo, la cantidad de información que puede ofrecerse mediante el habla supera con mucho a la que se consigue leyendo un texto en la pantalla de una terminal.

- El habla es el modo de comunicación más natural y universal de la especie humana. La interacción con los ordenadores es mucho más accesible si se utiliza el lenguaje natural. No hay que olvidar tampoco que así se facilita la utilización de los equipos informáticos a los invidentes.

1.8 Tarjetas de sonido

1.8.1 Características Generales

Las tarjetas de audio o sonido son adiciones a la expansión de la mother board para producir una alta calidad de sonido a través de bocinas externas o audífonos. La tarjeta de sonido es considerada un bloque básico construido para audio en la PC.

Al principio, estas tarjetas tenían como aplicación más directa proporcionar a los juegos de la PC un fondo musical más aparente y atractivo. Actualmente, sigue siendo uno de los motivos de adquisición.

Sin embargo ahora las tarjetas de sonido son más que un bonito sonido para juegos. Éstas son verdaderos procesadores de información de sonido. Y permiten hacer cosas como controlar la PC solo con la voz, oír automáticamente archivos de texto en cualquier idioma, realizar muestreos de sonidos reales y utilizarlos después en aplicaciones Windows, capturar sonidos o música de CD, discos, cassetes, instrumentos musicales, etc.

Además, se dispone de varios mezcladores para combinar las distintas fuentes de sonido, intercalar filtros de frecuencia, regular volúmenes y tonalidades, etc.

También se puede editar en pantalla, como si se tratará de un osciloscopio, las formas de onda de los sonidos capturados, pudiendo añadir a éstos efectos de eco, reverberación, repeticiones, cambiar su frecuencia de reproducción, volumen; en suma, moldear sonidos a nuestro gusto.

Las tarjetas de sonido ayudan a la computadora ha generar sonido que es de mejor calidad que el sonido de la bocina interna de la PC. Estas también proporcionan una salida externa para conexión de CD en los modelos antiguos, además se puede conectar un micrófono

sobre un puerto y recibir voz. Otras tarjetas pueden ser usadas para grabar sonidos de fuentes externas como el CD. Adicionalmente, muchas tarjetas soportan el MIDI[2] (Musical Instrument Digital Interface), la cual tiene un puerto que lo habilita.
La cualidad mas importante de la tarjeta de sonido es la de poder muestrear sonido real.

Para escuchar sonidos con la tarjeta de sonido, son necesarias bocinas, las cuales se compran separadamente. Se pueden usar audífonos, conectándolos en el puerto que se encuentra en la parte trasera de la tarjeta, otros puertos que puede incluir la tarjeta son un joystick o un puerto SCSI para CD-ROM. Un puerto SCSI o una interface pequeña , es una conexión que permite alta velocidad de transferencia de información entre la computadora y un dispositivo externo.

Características de las tarjetas de sonido

Hay tres tipos básicos de tarjetas de sonido. Hay tarjetas de sonido monofónicas, tarjetas de sonido estereofónicas y tarjetas que pueden primeramente ser usadas para sonidos MIDI.

La diferencia básica entre los tres tipos de tarjetas de sonido es debido a su capacidad de muestreo[3].

Es responsabilidad de la tarjeta colocar la tasa de muestreo para sonidos que graba. Esto lo hace a través de un chip llamado convertidor Analógico Digital (ADC) que convierte audio analógico que entra por el micrófono a forma digital que la computadora puede almacenar en RAM o en disco duro. También éstas realizan la función inversa, convierten la señal digital a analógica a través del convertidor Digital Analógico. La cualidad de los sonidos depende de la tasa de muestreo de un sonido(medida en kilohertz) y el tamaño de la muestra(medida en bits).

Existen tres tasas de muestreo estándar para tarjetas de sonido: 11.025, 32 050 y 44.1 KHz. Los tamaños de muestreo son: 8, 12, 16 y 24 bits.

Las tarjetas de sonido monofónicas son las más baratas Sin embargo, tienen mucho menos capacidad –los sonidos que producen son de una fuente simple. El sonido de éstas es mejor que el de la bocina interna de la PC.

La tarjeta estereofónica produce muchos más exacto de reproducción para archivos MIDI. Cada nota tocada en archivos MIDI requiere una voz separada.

Las tarjetas más caras hechas para profesionistas quienes trabajan sonidos al mismo tiempo y puede hacer uso de dos fuentes. Tiene más voces, las cuales dan sonido con MIDI y reproducción de sonido de alta calidad, pueden producir calidad de audio de CD, el cual es muestreado a 44.1 KHz y tamaño de muestreo de 16 a 24 bits.

---

[2] Comunicación estándar que permite a computadoras e instrumentos musicales comunicarse.
[3] Proceso de conversión de señales analógicas a un formato de señal digital, que es lo que la computadora maneja

Con la inclusión de extensiones multimedia, un nuevo concepto que ha revolucionado el mundo, en Windows 3.1 y posteriores, la demanda de tarjetas de sonido se ha incrementado significativamente.

1.9 Modelo Cliente/Servidor

La arquitectura cliente/servidor es una forma de distribución de computadoras que muy frecuentemente divide las aplicaciones en dos partes, conectadas por una LAN[4]. La interface de usuario y manipulación de datos ocurre sobre la estación de trabajo del usuario generalmente PC's , mientras que el almacenamiento, manejo de datos, control de la red, herramientas de administración del sistema como servicios de impresión y utilidades del usuario, reside sobre un servidor separado.

La ventaja de usar esta arquitectura es que maneja un proceso distribuído, en que las estaciones de trabajo realizan sus propios procesos después de cargar la información del servidor, éste se libera y se puede dedicar a funciones de administración de la red.

Cliente/servidor es una de las arquitecturas más eficientes, en la actualidad, para la administración de la información, ayuda a dar rapidez a la respuesta del sistema.

[4] LAN, red de área local

Capítulo 2.

Lenguajes Visuales de Desarrollo

2.1 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.

2.1.1 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 soporta 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.

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 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 maneja las pantallas (servidor) significa que sólo el servidor necesita conocer los detalles del hardware 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

Si las instrucciones de hardware especificas 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 2.1 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 clientes se comuniquen con los servidores. Esto lo proporciona el protocolo de red X. (Fig. 2.1)

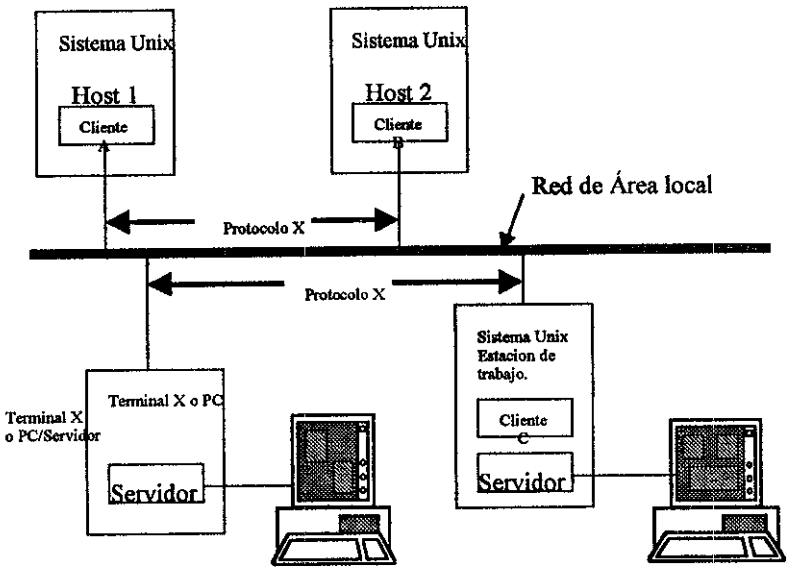


Fig. 2.1 Modelo cliente-servidor del sistema X Window

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.

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

OPEN LOOK y Motif proporcionan 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 manejador 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 cortar 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 características del teclado y que aplicaciones se incluyen con el sistema. Motif proporciona una aplicación xterm, pero no incluye un manejador de archivos.

2.1.3 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 las 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 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 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. La 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 consciente 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 funciones de la GDI, las cuales accesan una estructura llamada contexto de dispositivo. Windows establece una correspondencia entre la estructura de contexto 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.

2.2 Programación en ambiente Windows

La programación en Windows, tiene como principal diferencia de la programación tradicional en que aquí se define cada entidad como un objeto, el cual contiene ciertas propiedades que determinan su apariencia y comportamiento, por ejemplo para crear 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 a partir de cierto objeto en la pantalla, se ejecuten procedimientos al ser seleccionado este 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, posteriormente se establecen las propiedades que contendrá cada uno 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.

2.2.1 Programación Basada en Recursos

En MS-DOS la programación es controlada por datos, esto es, los datos tienen que ser codificados como valores inicializados, 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 ligado, por medio de un proceso llamado enlace. Los archivos de recursos pueden incluir mapas de bits, íconos, 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 crear o modificar 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.

2.2.2 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 sea un problema en las aplicaciones Windows, entre las cuales se incluye una técnica de swapp o intercambio 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 desempeño 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.

2.2.3 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 variables 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, Lstbox, 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, ubicación, contenido, función, etc.

2.2.4 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 diálogo, 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 su variedad está limitada por la misma imaginación del programador.

2.2.5 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 menús 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.

2.2.6 Cajas de Diálogo, Barras de Herramientas y Barras de Estado

2.2.6.1 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.

2.2.6.2 Barras de Herramientas y Barras de Estado

Existen en Windows 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 Mareo 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 línea 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.

2.2.7 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 Programa 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 registradas 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 de 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 convertido 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.

2.2.8 Intercambio Dinámico de Datos (DDE)

Las aplicaciones de Windows corren en un ambiente llamado multitarea[1], debido a esto, puede ser que otras aplicaciones estén siendo ejecutadas simultáneamente a nuestra aplicación y alguna de esa información será 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 manipular 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.

---

[1] **Multitarea**, en informática, modo de funcionamiento disponible en algunos sistemas operativos, mediante el cual una computadora procesa varias tareas al mismo tiempo. Dado que el sentido temporal del usuario es mucho más lento que la velocidad de procesamiento de la computadora, las operaciones de multitarea en tiempo compartido parecen ser simultáneas.[1]

2.2.9 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, 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 archivos 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 forma determinada 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 de disco y

el formato de presentación de Windows que es un metafile con una presentación visual de los datos subyacentes.

2.2.10 Bibliotecas de Enlace Dinámico (DLL Dynamic Link Library)

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.

2.3 Herramientas de desarrollo

El mercado de las herramientas de desarrollo de aplicaciones. El modelo 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. *SQL Windows* y *Power Builder* caen dentro de las herramientas orientadas a objetos, mientras que otras como *Visual Basic* y *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:

Power Builder de Powersoft
SQL Windows de Gupta
Object View de Knowledge Ware
Uniface Six de Uniface
Visual Basic y Visual "C++" de Microsoft
Visual Works de ParePlace Systems
Visual Age 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 Small Talk, como Visual Age y Visual works 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 SQL Windows y Power Builder termina compitiendo entre ellos. Aunque Object view es una herramienta buena, bastante capaz, no tiene más que ofrecer que Power Builder y SQL Windows, además de que es mucho menos popular que estas dos. Visual Basic y Visual C++ se utilizan más por su facilidad de uso y generalmente Visual Basic se utilizan para crear aplicaciones de bases de datos sencillas, así como Visual C++ es una herramienta que proporcionará al programador crear aplicaciones más sofisticadas en cuanto a lo poderoso que puede resultar el uso de este. Así como también se pueden 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 SQL Windows, Power Builder 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.

2.3.1 SQL WINDOWS (Gupta Corp.)

SQL Windows 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 SQL Windows a convertirse en una herramienta de desarrollo visual que es mucho más fácil 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 SQL Windows Application language es capaz de

completar la mayoría de las tareas de programación. El compilador convierte las funciones internas de SQL Windows en código de C y luego compila las sentencias en DLL's.

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.

SQL Windows 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 interactuar 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 SQL Windows, 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 SQL Windows que crea aplicaciones MDI y que forman el marco de la aplicación; en ésta se encuentran 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 SQL Windows, 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, 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.

2.3.2 Power Builder (Power Soft Corp.)

Power Builder es una herramienta poderosa que permite desarrollar fácilmente aplicaciones cliente/servidor en Windows, Power Builder ofrece un rico complemento de facilidades para su uso y un significante grado de orientación a objetos. A pesar de que competidores como SQL Windows tiene mayor orientación a objetos, Quick Objects, etc., Power Builder ha logrado entender mejor al mercado y jugar con su imaginación. Entre las características de Power Builder 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 Power Builder 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 Power Builder, encapsulación, herencia y polimorfismo son todos soportados en esta herramienta. En el corazón de Power Builder está el almacén central de objetos y atributos que pueden ser usados por los programadores para definir el aspecto y el comportamiento de nuevos objetos.

El Painter de Datawindows crea 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 es la pantalla en la cual se administran todos los objetos de una aplicación, ayuda a integrar Power Builder 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 poderosa herramienta simplifica la creación de formularios complejos y de detalles a nivel maestro, permitiendo a los usuarios la construcción rápida de plantillas para la entrada de datos.

2.3.3 Uniface Six (Uniface)

Uniface Six está preparado para competir con Power Builder y SQL Windows, 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.

Este lenguaje 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. Aunque Uniface Six pueda 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.

2.3.4 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 mucho más al desarrollar 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 Exchjange (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 (DLL's).

Otra característica importante de Visual Basic, es que se obtiene como resultado final un ejecutable que utiliza una DLL para las rutinas y que puede ser distribuida 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 utilizaron cada vez que se escribe una aplicación, sin importar la complejidad de ésta.

El ambiente de desarrollo de Visual Basic contiene una serie de herramientas y utilerias 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 box, combo box, botones de comando, list box, etiquetas, menús, etc.

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

Forma.- Sirve como una ventana que se formatea como la interface de la aplicación. A ésta se agregan los controles, gráficas e imágenes para crear la presentación que se desea tenga a 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 tiene 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 la programación, 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. El sistema se debe dividir en procedimientos. El procedimiento de un evento contiene instrucciones que se ejecutan cuando ocurre un evento tal como la selección de un botón.

Una vez terminada 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 aplicación de Windows.

Visual Basic es una interface amigable que permite, sin ser un experto programador, crear aplicaciones. 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.

2.3.5 Visual C++ (Microsoft Corp.)

Visual C++ da continuidad a la larga línea de herramientas de Microsoft para el desarrollo en Windows. El paquete de Visual C++ no solo contiene un compilador, sino también todas las bibliotecas, ejemplos y documentación que necesita para crear aplicaciones para Windows 95 y Windows NT.

Esta herramienta utiliza el entorno de trabajo Visual Wordbench (VWB), que es el entorno principal de edición y depuración, proporciona utilidades para facilitar la programación. A continuación daremos una breve explicación de estas.

Manejador de Aplicaciones AppWizard Una aplicación generada por un AppWizard se le llama *esquema de aplicación* que es un programa mínimo o básico de Visual C++.
Además con esta herramienta se pueden crear y diseñar cuadros de dialogo, menús, mapas de bits, y otras clases de recursos. AppWizard genera una aplicación que contiene los elementos básicos de una aplicación basada en Windows, incluyendo los siguientes:

- *Una ventana principal* y las restantes ventanas requeridas por la clase de aplicación que se esté creando.
- *Una barra de menú* equipada con los *menús estandar* de Windows, como los de archivo, edición y ayuda.
- Todos los elementos de los *menús y cuadros de dialogo* necesarios para abrir archivos, guardarlos, imprimirlos, e implementar la funcionalidad de preparar página.
- Una *barra de herramientas* y una de estado.
- *Soporte OLE* para objetos contenedores y servidores.
- Todos los *archivos fuente* y de *recursos necesarios* para crear una aplicación construida sobre las clases de Visual C++.

Cuando AppWizard ha generado un esquema de aplicación, se podrá transformar en la clase de aplicación que nosotros deseamos diseñar.

Manejador de recursos AppStudio.- El entorno de trabajo Visual tiene un conjunto de herramientas de programación denominados ***Asistentes*** (Wizards). Los cuales realizan complicadas secuencias de tareas por nosotros.

Manejador de clases ClassWizard.- Se usa para definir las clases en un programa creado con AppWizard. Mediante el uso de classWizard, se pueden definir clases en el proyecto. También puede añadir funciones que controlan la manera en que se manejan los mensajes recibidos por cada clase. ClassWizard también ayuda a manejar controles que están contenidos en los cuadros de diálogo, permitiéndole que asocie un objeto MFC o una variable miembro de clase con cada control.

Selector de código de Visual C++.- Este utiliza información generada por el compilador para ayudarle a examinar y editar los símbolos relacionados, así como estudiar las relaciones existentes entre diversos símbolos. El selector presenta una vista jerárquica que puede utilizar para examinar los símbolos en los que se encuentre interesado. Puede seleccionar cualquier función, variable, tipo, macro, o clase y ver dónde se encuentra definida exactamente en el proyecto.
El compilador crea un archivo con la base de datos para el selector – con la extensión .BSC en su nombre archivo – que incluye información relativa a dónde se define y se utiliza cada uno de los símbolos de su código fuente. Una vez que se ha creado el archivo .BSC, forma parte del proyecto y se actualiza cada vez se construye.

Depurador de Visual C++.- Para ayudar a la detección de errores sintácticos del lenguaje, variables no declaradas, o palabras reservadas mal escritas en programas y corregirlos, Visual C++ incluye esta herramienta de depuración interactiva, basada en gráficos, integrada con el editor C++.

Editor de Visual C++.- Este editor esta basado en Windows estándar equipado con funciones especiales para redacción de programas en Visual C++.
La funcionalidad de coloreado del código basado en la sintaxis facilita la localización de clases especiales de frases o palabras clave en Visual C++.
Otras características son el sangrado automático de las líneas en las funciones, utilidad de buscar y reemplazar, depurador de código fuente, etc.

Compilador de Visual C++.- Determina el lenguaje de un archivo comprobando la extensión de su nombre de archivo: El trabajo del compilador de Visual C++ es traducir archivos con código fuente (archivos de texto con extensión .C o .CPP) a archivos en código máquina llamados archivos objeto (archivos con la extensión .OBJ).

Mientras el compilador está trabajando, visualiza un informe de su avance en una ventana de salida. Se presentan mensajes de error o de aviso si se detectan puntos conflictivos. El compilador se detiene y presenta un mensaje de error si encuentra un error fatal.

Enlazador de Visual C++.- Liga los archivos OBJ generados por el compilador de Visual C++ a otros archivos objeto que la aplicación pueda necesitar. Cuando el enlazado va a ligar los archivos OBJ, realiza una búsqueda a través del código que está compilando, de los nombres de las funciones que no están implementadas , que ha recibido el compilador:

Para la creación de la interfaz, 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 tiene 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 la programación, dentro de la ventana de código. Esta ventana está compuesta de instrucciones en el lenguaje de Visual C++, contiene instrucciones adicionales para manejo de objetos, mensajes y comunicaciones. El sistema se debe dividir en *procedimientos*. El procedimiento de un evento contiene instrucciones que se ejecutan cuando ocurre un evento tal como la selección de un botón.

Una vez terminada estas tres etapas, solamente queda pendiente ejecutar la aplicación.

Visual C++ es una interface amigable que permite, que sea muy fácil crear aplicaciones para Windows. Usando las herramientas y asistentes proporcionados como parte del VWB en conjunto con la biblioteca de clases MFC, puede crear un programa en tan solo unos minutos.

Con Visual C++, 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 Exchjange (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).

Capítulo 3.

DLL (Dynamic Link Library)

3.1 Definición

Una DLL es un módulo ejecutable que contiene funciones y recursos, que permiten a las aplicaciones compartir código y recursos entre diferentes módulos o entre diferentes aplicaciones. Son de igual manera módulos programables basados en Windows que constituyen una parte importante en la programación basada en Windows.

La importancia que tienen las DLL´s en el ambiente Windows se debe a la forma en que las utilizan los programas basados en este ambiente.

Para las bibliotecas de enlace dinámico existen dos tipos de ellas: de recursos y funcional.

- Una DLL de recursos permite compilar varios recursos de diferente naturaleza, tales como: mapas de bits, cursores e iconos en un archivo.
- Una DLL funcional realiza algún tipo de función y puede ser una subrutina que puede o no generar un valor a la salida.

3.2 Características

Existen principalmente dos tipos de bibliotecas que se utilizan en programación: de enlace dinámico y de enlace estático. A continuación se describen sus características.

3.2.1 Bibliotecas de enlace dinámico

- Son módulos que se cargan y ligan en tiempo de ejecución.
- Permiten compartir código y recursos.
- El código de las funciones que contiene no está incluído en el código del programa de aplicación, esto es, están separadas del código ejecutable.
- Es posible recompilar o revisar una DLL sin necesidad de recompilar el código fuente.
- Permiten crear programas ejecutables de menor tamaño y realizar un manejo eficiente de la memoria.

3.2.2 Bibliotecas de enlace estático

- Son módulos que se ligan en tiempo de compilación.
- El código es adicionado al final del código ejecutable.
- No permiten compartir código.
- Al modificar la librería, se necesita recompilar el código fuente.
- Son ineficientes en el manejo de la memoria.

3.3 Ventajas de las bibliotecas dinámicas con respecto a las bibliotecas estáticas

De acuerdo a las características antes mencionadas, proporcionamos las ventajas de utilizar las librerías de enlace dinámico en vez de las librerías de enlace estático.

- La eficiencia en el manejo de la memoria.
- El que permita compartir recursos.
- La creación de programas de menor tamaño.

3.4 Estructura de las DLL´s

Para crear una DLL es necesario que existan algunos archivos que nos permitan tener la estructura de la DLL, en la tabla 3.1 se muestran éstos.

Extensión del archivo	Propósito
*.C	Archivo fuente para la DLL
*.DEF	Archivo usado para manejar parámetros Windows y de exportación
*.MAK	Archivo que le indica al compilador como crear el archivo DLL. Es creado por el compilador cuando crea el proyecto

Tabla 3.1 Archivos para la creación de un DLL

3.4.1 Archivo de definición "DEF"

Este archivo se necesita siempre para crear una DLL, es usado para indicar al compilador acerca de los parámetros Windows usados en el archivo DLL, así como el nombre de las funciones y subrutinas a exportar, el código de un archivo de definición es como sigue.

```
LIBRARY         MiDLL
DESCRIPTION     'DLL propia'
EXETYPE         WINDOWS
STUB            'WINSTUB.EXE'
CODE            PRELOAD MOVEABLE
DATA            PRELOAD MOVEABLE MULTIPLE
HEAPSIZE        4096
EXPORTS         MiDLLWndProc
```

Palabra clave	Descripción
LIBRARY	Nombre de la DLL
DESCRIPTION	Descripción del módulo
EXETYPE	Indica si es aplicación Windows u OS/2
STUB	Programa a ejecutar sin Windows
CODE	Atributos de segmento de código
DATA	Atributos de segmento de datos
HEAPSIZE	Tamaño del heap local
STACKSIZE	Tamaño del stack local
EXPORTS	Funciones export
IMPORTS	Funciones import
SEGMENT	Segmento de código adicional

Tabla 3.2 Descripción de las palabras claves

LIBRARY: Si se trata de una DLL, se especifica LIBRARY.

STUB: El programa WINSTUB.EXE ya existe, y se llama cuando se intenta ejecutar la aplicación Windows desde DOS, el programa visualiza en pantalla la frase "Este programa requiere Microsoft Windows" y termina la aplicación de inmediato.

STACKSIZE: Todo programa Windows posee un stack, cuyo tamaño se especifica en bytes, éste se necesita para guardar temporalmente argumentos de función, siendo 4096 bytes un mínimo recomendado de una aplicación Windows pequeña, además se hace notar que una DLL no posee stack propio.

HEAPSIZE: Indica el tamaño del Heap local, siendo 256 Bytes el mínimo, éste se inicializa en la primera carga de una DLL en una función Startup.

CODE, DATA: Los segmentos de código y datos se colocan ambos en MOVEABLE. En la palabra clave CODE se puede especificar el parámetro DISCARDABLE, que indica que el segmento de código también se puede eliminar de la memoria en caso de falta de la misma. En la palabra clave DATA se puede encontrar la palabra SINGLE o NONE, esto es, la DLL posee o un segmento de datos para datos globales y estáticos o no posee segmento de datos. Pero nunca puede tener varios segmentos de datos. Así mismo la opción MULTIPLE indica que cada vez que se ejecuta la aplicación de nuevo, se crea un segmento de datos nuevo, indicándose siempre esta opción en una aplicación Windows, como en una DLL solo puede existir un segmento de datos, en vez de MULTIPLE se indica SINGLE.

EXPORTS: Define los nombres de todas las funciones a exportar, las que se exportan si son llamadas por Windows u otra aplicación. Todas las funciones que llama Windows, se denominan funciones CALLBACK.

IMPORTS: Indica todas las funciones que son llamadas por la aplicación, pero que no están definidas en su archivo fuente ni en las librerías enlazadas estáticamente a esta aplicación. Estas funciones están definidas en las DLL.

3.4.2 Archivo fuente "C"

En este archivo se encuentra el código para las funciones o subrutinas que la DLL tiene disponibles para ser utilizadas por las aplicaciones.

Existen tres componentes principales en este archivo: el archivo de encabezado windows.h, la función LibMain y la función WEP(Windows Exit Procedure).

- El archivo de encabezado windows.h se requiere en todo programa Windows, en este archivo se encuentran definidos los prototipos de todas las funciones de Windows, como tipos de datos, mensajes y constantes

- La función LibMain es análoga a la función main de un programa en C/C++ o a la función WinMain de cualquier programa en Windows. Esta función es automáticamente llamada cada vez que la DLL es cargada en memoria y ejecutada. Es además el punto de entrada de la DLL, siendo su función liberar el segmento de datos de la librería.

LibMain devuelve un valor de 1 indicando que la DLL se ha inicializado con éxito o devuelve un valor de 0 indicando al programa principal que la DLL no ha podido ser cargada.

- La función WEP debe ser incluída en todas las DLL de Windows. Es llamada por Windows cuando se finaliza el sistema, o cuando el contador interno de la DLL se coloca a 0. La función WEP nos permite liberar la memoria ocupada por la DLL cuando esta termina.

La función WEP tiene un solo parámetro que no se utiliza y que siempre devuelve un valor de 1, indicándonos que siempre podemos salir de la DLL.

3.5 Lenguajes que permiten el manejo de las DLL's

En este proyecto las DLL´s son importantes ya que permiten el manejo de cierto hardware y software que anteriormente para programar era complicado, los DLL´s ya son herramientas que nos simplifican la programación de ellos. La aplicación está desarrollada en ambiente Windows utiliza como base en la programación del proyecto las bibliotecas dinámicas Winsock.dll y FB_SPCH.dll.

La primera de ellas contiene las funciones necesarias para el manejo sockets en el ambiente Windows, mientras que la segunda contiene las funciones que habilitan el manejo de la tarjeta de sonido para la reproducción de texto a voz.

Los lenguajes de desarrollo comerciales que permiten el uso de bibliotecas dinámicas son SQLWindows, Visual Basic, Visual C++, Power Builder y Delphi, cada uno de ellos realizando la definición de DLL de forma diferente.

En el caso de Visual Basic, las DLL´s son bien soportadas, con el inconveniente de que con este lenguaje no existe la opción de crear una DLL.

Una opción que tenemos para poder crear una DLL es el lenguaje de programación Visual C++, que cuenta con las herramientas necesarias para la creación en toda su estructura de una DLL, incluyendo la compilación de los módulos.

Teniendo como referencia el punto anterior, y agregando algunos puntos más como: facilidad en el manejo y construcción de las bibliotecas dinámicas, y beneficios proporcionados por el lenguaje Visual C++, lo utilizaremos para el desarrollo del proyecto, en los módulos correspondientes.

Capítulo 4.

Protocolos de Comunicación TCP/IP

4.1 Introducción

El conjunto de protocolos TCP/IP (Transmission Control Protocol / Internet Protocol) es actualmente un estándar para la mayoría de las organizaciones que requieren una conexión en Internet, esto es, la conexión de redes de computadoras con diferentes arquitecturas.
El concepto general fue creado por la Agencia de Proyectos de Investigación Avanzada (ARPA Advanced Research Projects Agency), como una forma de probar la viabilidad de las redes de conmutación de paquetes. Durante el proyecto ARPA utilizó una red de líneas conectadas por nodos de conmutación, denominándose a ésta red ARPANET.

El conjunto de protocolos TCP/IP define reglas mundiales de comunicación para la transmisión y recepción de información independientemente de sistemas interconectados en Red con diferentes arquitecturas, esto es, conexión entre redes heterogéneas.

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 implantación y políticas Internet.

4.2 Modelo de referencia OSI

Para la comunicación entre sistemas de diferentes arquitecturas, se necesita una referencia común para este tipo de comunicaciones. A principios de los 80's la Organización Internacional de Estándares (ISO International Standards Organization) desarrolló un modelo para describir la estructura y función de los protocolos de comunicaciones de datos. Este fue llamado modelo de referencia OSI (Open Systems Interconnections: Sistemas Abiertos Interconectados) que define términos entendibles y ampliamente utilizados en las comunicaciones de datos. El modelo OSI no es un protocolo, ni define a los protocolos, sino que sirve para definir características y funciones que deben tener los protocolos.

4.2.1 Capas

El modelo OSI contiene siete capas, como se muestra en la figura 4.1, que definen las funciones de los protocolos de comunicaciones de datos. Cada capa representa una función realizada cuando los datos son transferidos entre aplicaciones a través de una red

Las capas de aplicación, presentación y sesión están orientadas a la aplicación, siendo responsables de la presentación de la interfaz al usuario.
Las cuatro capas restantes tienen que ver con la transmisión de datos, y se ocupan del enrutamiento, verificación y transmisión de cada grupo de datos.

A continuación se explicará cada una de las capas del modelo OSI.

Modelo OSI

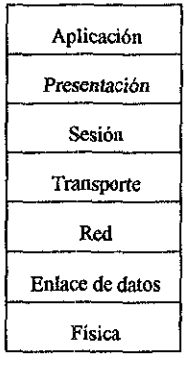


Figura 4.1 Representación en capas

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

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

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

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

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

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

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

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

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

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

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

Capa Física
La capa física está relacionada con las especificaciones eléctricas, electromágneticas y ópticas, de cómo los bits son transmitidos a través del medio de comunicación, en los sistemas que incluyen componentes eléctricos para la transferencia de datos, las especificaciones se refieren a la forma en como los ceros y los unos son representados por voltaje, y si la comunicación es simplex(1), half duplex(2) o full duplex.

La capa física también especifica el tipo de conectores usados y la asignación de pines en los conectores; la unidad de datos en el nivel físico es el bit. Ejemplos de estándares en este nivel son: 802.3(Ethernet) , RS-232C, 802.4(Token Bus) y 802.5(Token Ring).

4.3 Modelo TCP/IP

Este modelo de comunicación está constituido por cuatro capas las cuales se encargan de realizar una tarea especifica en el momento de la comunicación.

Las capas que componen el modelo son: ***aplicación, transporte, red e interfaz física de red.***

El diagrama siguiente muestra la organización del modelo TCP/IP. Figura 4.2

(1) Transmisión que se realiza en un solo sentido.
(2)Transmisión que se realiza en ambas direcciones pero no al mismo tiempo.

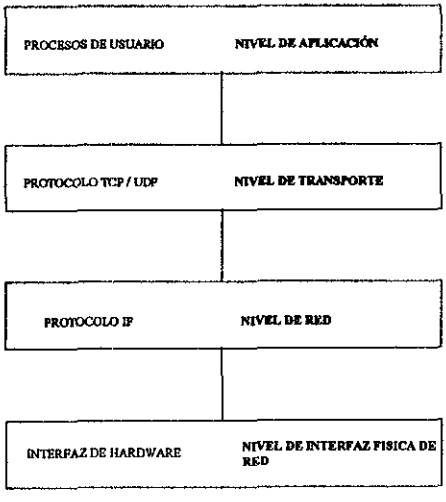


Fig. 4.2. Capas del modelo TCP/IP

Las cuatro capas que forman el modelo TCP/IP realizan tareas especificas en el momento de la comunicación de datos en las redes. Las capas realizan las siguientes tareas:

4.3.1 CAPA DE APLICACIÓN

Esta capa incluye todos los procesos que usan los protocolos de la capa de Transporte para entregar datos. Existen muchos protocolos de aplicaciones, la mayoría proporciona servicios al usuario y nuevos servicios están siempre siendo añadidos a esta capa. Las aplicaciones más conocidas son: emulación de terminal remota TELNET, protocolo de transferencia de archivos FTP, protocolo simple de transferencia de correo SMTP y programas de aplicación para comunicación personal como lo es TALK y el correo electrónico MAIL.

4.3.2 CAPA DE TRANSPORTE

Tiene como función principal proporcionar la comunicación entre un programa de aplicación y otro, a este tipo de comunicación se le llama extremo-a-extremo. Esta capa puede regular el flujo de información. Los dos más importantes protocolos de comunicación en esta capa son protocolo de control de transmisión (TCP) y el protocolo de datagrama de usuario (UDP). TCP proporciona una transferencia confiable de datos, asegurando que los datos lleguen sin error y en secuencia con una detección de error y corrección extremo-a-extremo. Para realizar esto, dispone que la parte receptora envíe de regreso una confirmación, para retransmitir paquetes perdidos. UDP es un protocolo orientado a comunicaciones sin conexión, lo que significa que no tiene un mecanismo para

la retransmisión de datagramas. UDP no es muy confiable, pero tiene fines particulares; si las aplicaciones que utilizan UDP tienen su propia verificación de confiabilidad, se pueden superar los inconvenientes de UDP.

4.3.3 CAPA DE RED

Esta capa permite manejar la comunicación de una computadora a otra en una red, el Protocolo Internet (IP) es el corazón de TCP/IP y es el protocolo más importante en esta capa. IP proporciona el servicio de entrega de paquetes en la cual las redes TCP/IP son construidas. Todos los protocolos de las capas superiores e inferiores a IP, usan el Protocolo Internet para entregar datos.

4.3.4 CAPA DE LA INTERFAZ FISICA DE RED

Es el responsable de aceptar datagramas IP y transmitirlos sobre una red especifica, las funciones desempeñadas en este nivel incluyen encapsulación de datagramas IP dentro de los frames transmitidos por la red, y mapeo de direcciones IP a direcciones físicas usadas por esta. Además proporciona el acceso al medio físico de comunicación (cable coaxial, par trenzado, fibra óptica, etc.), controla el flujo de la información, detecta y corrige errores.

4.4 Correspondencia entre el Modelo OSI y el TCP/IP

Los modelos descritos con anterioridad cuentan con algunas similitudes, ambos tienen una arquitectura en capas, aunque la arquitectura OSI está definida más estrictamente y las capas son más independientes que las de TCP/IP.
Por otro lado las diferencias entre las arquitecturas están relacionadas con las capas encima del nivel de transporte, y las que corresponden al nivel de red. El modelo OSI cuenta con una capa de sesión y una de presentación , mientras que el modelo TCP/IP combina ambas en una capa de aplicación. Además TCP/IP combina las capas física y de vínculo de datos del modelo OSI en la capa de interfaz física de red. En la figura 4.3 se muestra un diagrama esquemático de la estructura de capas TCP/IP comparándola con la del modelo OSI.

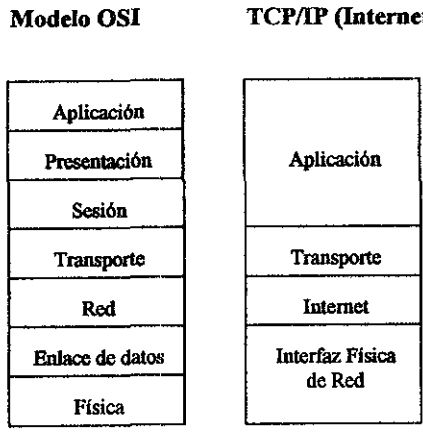


Figura 4.3 Analogía entre modelos

4.5. Conjunto de Protocolos TCP/IP

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

El modelo de niveles permite realizar las funciones de red en cada nivel, solo es necesario conocer el servicio que el nivel necesita para proveérselo. Las aplicaciones desarrolladas por TCP/IP generalmente requieren de varios de los protocolos que forman el conjunto TCP/IP para poder comunicarse. El conjunto de los niveles de los protocolos también es conocido como protocolo stack o pila y funciona como sigue: el nivel más alto de la computadora fuente pasa información a los niveles más bajos del modelo TCP/IP, hasta llegar al último nivel que es el físico. El nivel físico transfiere la información a su destino. Los niveles más bajos de la computadora destino pasan la información a sus niveles más altos, los cuales llevan a los datos a la aplicación destino.

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

En una aplicación para la transferencia de archivos usando TCP/IP, se realizan las siguientes operaciones para mandar el contenido de un archivo a otra computadora:

1. El nivel de aplicación pasa un conjunto de bytes al nivel de transporte en la computadora fuente.

2. El nivel de transporte divide el conjunto de bytes(stream) en segmentos TCP y adiciona un encabezado con un número de secuencia para este segmento, y pasa el segmento al nivel Internet IP.

3. El nivel IP crea un paquete con la porción datos conteniendo el segmento TCP y adiciona un encabezado al paquete conteniendo la dirección IP fuente y destino. El nivel IP también determina la dirección física de la computadora destino o computadoras intermedias en el camino de la computadora destino, y pasa el paquete y la dirección física al nivel de enlace de datos.

4. El nivel de enlace de datos transmite el paquete IP dentro de la porción de datos del frame a la computadora destino.

5. En la computadora destino, el nivel de enlace de datos descarta el encabezado del nivel de enlace de datos de la computadora origen y pasa el paquete IP al nivel IP.

6. El nivel IP verifica el encabezado del paquete IP para determinar su longitud. Si los datos contenidos en el encabezado no son iguales con la suma calculada por el nivel IP, descarta el paquete.

7. Si la suma concuerda, el nivel IP descarta el encabezado IP y pasa el segmento TCP el nivel TCP. El nivel TCP verifica la secuencia para determinar si el segmento es correcto en la secuencia.

8. El nivel TCP realiza una suma para el encabezado y los datos, y si el cálculo no concuerda con la suma transmitida en el encabezado, el nivel descarta el segmento. Si la suma es correcta y el segmento está en la secuencia correcta, el nivel TCP descarta el encabezado TCP y pasa los bytes al nivel de aplicación.

9. La aplicación en la computadora destino recibe un conjunto de bytes, tal como si ésta estuviera conectada directamente a la aplicación en la computadora fuente

El conjunto de protocolos TCP/IP esta compuesta por varios protocolos, dentro de los cuales destacan TCP, IP, ARP, RARP e ICMP. A continuación se describen brevemente cada uno de ellos.

4.5.1 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 "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 datos a la cual se le llama "Acknowledgment" (ACK). Así entonces, la seguridad estará establecida por:

1. Si una 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.

4.5.2 UDP (User Datagram Protocol)

Provee un servicio de datagramas sin 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, debido a que no ofrece capacidades de recuperación de errores.

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

4.5.3 IP (Internet Protocol)

El Protocolo Internet (IP) fue desarrollado para "proveer las funciones necesarias para enviar los paquetes de bits (en datagramas 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:

1. Provee un ruteo de información de una máquina a otra (siendo el ruteo su función primaria).
2. 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.
3. 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.

La conmutación de paquetes se realiza por medio de la transmisión de información en la red de pequeños segmentos o paquetes. Si una computadora transmite un archivo grande a otra computadora el archivo es dividido en varios paquetes en el origen y reensamblados en el destino. El protocolo TCP/IP define los formatos de estos paquetes que deben tener la dirección origen, la dirección destino, la longitud del paquete y el tipo de paquete, así como la ruta a seguir en la red.

4.5.4 ICMP (Internet Control Message Protocol)

Este protocolo detecta errores y controla la transmisión de la información entre gateways y servidores, cuando sé esta llevando a cabo un proceso de comunicación de datos a través de la red.

Es responsable de verificar y generar mensajes sobre el estado de los dispositivos de una red. Puede utilizarse para informar a otros dispositivos de la falla de una máquina en particular. Por lo general, ICMP e IP funcionan juntos.

ICMP forma parte integral del Protocolo Internet y debe incluirse en cualquier implementación. Esto permite el envío de mensajes y señales de error consistentes y comprensibles a través de distintas versiones. Resulta útil pensar en ICMP como un paquete IP comunicándose con otro paquete IP a través de la red. Los mensajes generados por ICMP son tratados como un paquete IP. Los mensajes tienen un encabezado de la misma forma que los de las aplicaciones de usuario, y no se diferencian hasta que la máquina receptora procesa correctamente el datagrama.

En casi todos los casos, los mensajes de error enviados por ICMP son devueltos al emisor del datagrama original. Esto es debido a que solamente las direcciones IP del emisor y receptor están incluídas en el encabezado; y debido a que el error no tendrá ningún significado para el destino, el emisor es el receptor lógico del mensaje de error. El emisor determinará, a partir del mensaje ICMP, el tipo de error que ocurrió y establecer la mejor forma de volver a enviar el datagrama fallido.

Los mensajes ICMP pasan por dos encapsulados. Cuando se incorporan a un datagrama IP común y a continuación en un grupo de red. Los encabezados ICMP tienen un formato distinto a los encabezados IP, y el formato difiere ligeramente del tipo de mensaje.

Usualmente, cualquier ICMP que esté informando de un problema en la entrega también incluirá el encabezado y los primeros 64 bits del campo de datos del datagrama sobre el cual ocurrió el problema. La inclusión de los 64 bits del datagrama original tiene dos objetivos: permitir al emisor identificar el fragmento del datagrama comparándolo con el original; también, debido a que la mayor parte de los protocolos involucrados se definen al inicio del datagrama, la inclusión del fragmento original del datagrama permite algunos diagnosticos.

4.5.5 ARP (Address Resolution Protocol)

Es un protocolo que mapea la dirección IP a la dirección física.

Su tarea de ARP es convertir las direcciones IP a direcciones físicas (de red y local), y al hacerlo elimina la necesidad de que las aplicaciones sepan direcciones físicas. Esencialmente, el ARP es una tabla con una lista de direcciones IP y sus direcciones físicas correspondientes. La tabla se conoce como caché ARP.

Cuando ARP recibe la dirección IP de un receptor, busca en el caché ARP alguna coincidencia. Si encuentra alguna, devuelve la dirección física. Si el caché ARP no encuentra alguna coincidencia correspondiente a la dirección IP, envía un mensaje a la red. El mensaje conocido como solicitud ARP, es una difusión que se recibe en todos los dispositivos de la red local. La solicitud ARP contiene la dirección IP del dispositivo receptor deseado. Si su dispositivo reconoce la dirección IP como suya, envía un mensaje de respuesta con la dirección física de regreso a la máquina que generó la solicitud ARP, y ésta coloca la información en su caché ARP para uso futuro. De esta forma, el caché ARP puede determinar la dirección física de cualquier máquina basada en su dirección de IP.

Siempre que un caché ARP recibe una solicitud ARP, éste utiliza la información incluida en la solicitud para actualizar su propia tabla. Por lo tanto, el sistema se puede adecuar en forma dinámica a direcciones físicas cambiantes y a nuevas adiciones a la red, sin tener que generar una solicitud ARP propia.

4.5.6 RARP (Reverse Address Resolution Protocol)

Es el protocolo que mapea las direcciones físicas a direcciones lógicas.

Es una variante de ARP. RARP también convierte direcciones, pero al contrario de lo que hace ARP. Este convierte direcciones Ethernet a direcciones IP.

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

4.6 Direcciones Internet

Para la entrega de datos en Internet entre dos servidores, es necesario trasladar los datos a través de la red hacia el servidor especificado y dentro del servidor al usuario o proceso al que se desea entregar los datos, para realizar lo anterior, TCP/IP usa algunos esquemas para llevar a cabo estas tareas como son: Direccionamiento, Ruteo y Multiplexaje.

4.6.1 Direccionamiento IP

El Protocolo Internet mueve datos entre servidores en forma de datagramas. Cada datagrama se entrega a la dirección que contiene la dirección destino del encabezado del datagrama. La dirección destino es una dirección IP estándar de 32 bits que contiene información suficiente para identificar únicamente a una red y a un servidor específico en esa red.
Cada dirección IP es dividida en 2 identificadores: "Host" y "Red. Se pueden tener 5 clases de red (A, B, C, D y E), las cuales se diferencian en los primeros 4 bits y en el número de bits que almacena el "ID Host" y el "ID Red".

Clase A

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

Clase B

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

Clase C

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 de red.

Clase D

Es usada para multicasting a un número de nodos. Los paquetes son pasados a un subconjunto seleccionado de usuarios en la red. Solamente estos nodos son registrados y serán quienes 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 1-1-1-1-0

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

4.7. TCP/IP de Microsoft

Este consiste en una implementación completa del protocolo estándar orientado a conexión (TCP/IP). Windows incorpora únicamente el soporte en modo protegido para este protocolo.

El protocolo TCP/IP de Microsoft ofrece las siguientes ventajas:

- Soporte para la conectividad de internet y para el protocolo punto a punto (PPP) utilizado en comunicación asincrona.
- Conectividad a través de redes con diferentes sistemas operativos y plataformas, incluyendo la comunicación con muchos sistemas que no son de Microsoft, como hosts de internet, sistemas Apple Macintoch, grandes sistemas de IBM, sistemas UNIX o sistemas Open VMS.
- Soporte para configuración automática de TCP/IP mediante los servidores del Protocolo de configuración de hosts dinámicos (DHCP) de Windows NT.

- Soporte para resolución automática de nombres de computadoras de direcciones IP a NetBIOS mediante la utilización de los servidores del Servicio de Nombres de Internet para Windows (WINS) de Windows NT.
- Soporte para la interfaz de conexiones lógicas de red (Winsockets ver 1.1), que son utilizadas por gran número de aplicaciones cliente-servidor y por muchas herramientas de Internet de dominio público.
- Soporte para la interfaz de NetBIOS, que normalmente se conoce como NetBIOS sobre TCP/IP.
- Soporte para muchas de las utilidades más comúnmente usadas, que se instalan con el protocolo.

Capítulo 5.

Sockets

5.1 Introducción

Los sockets fueron introducidos en 1981 en la versión del sistema operativo UNIX de Berkeley 4.2; en la actualidad son parte integral de cualquier sistema operativo UNIX.

Un API(Application Program Interface) es una interfaz para el programador. La validez de un API depende del sistema operativo que sea usado y el lenguaje de programación. Existen APIs para los sistemas UNIX, sockets Berkeley y el Sistema V de AT&T (TLI Interfaz de la Capa de Transporte). Ambas interfaces fueron desarrolladas en lenguaje "C"

5.2 Definición

Las bases para aplicaciones de red en 4.3 BSD son una abstracción referida como un socket. Un socket es el punto final de comunicación entre servidores, es la combinación de una dirección IP y un número de puerto y puede usarse para enviar y recibir información. Los sockets son componentes básicos para la intercomunicación entre procesos. Usando sockets es posible accesar el medio físico de una red directamente o bien utilizar alguna familia de protocolos diferente a TCP/IP.

5.2.1 Características

Los principales puntos a tomar en *cuenta son*:

1. El tipo, el dominio y el protocolo.
2. El estado del socket (conectado o sin conexión, enlazado, en estado de recibir o enviar datos).
3. Apuntadores de datos, tanto en emisión como en recepción.

5.2.2 El papel de los sockets en el modelo OSI.

Para que pueda efectuarse la *comunicación entre* computadoras son necesarios los sockets los cuales trabajan en la capa de sesión del modelo OSI y que nos permitirán tener un alto grado de confiabilidad en los mensajes originados en la capa de aplicación de la misma forma en que la capa de enlace asegura la comunicación entre nodos adyacentes. La razón por la que se ubican en este nivel es por la transmisión de paquetes de información que se realiza desde el punto origen hasta el punto destino.

5.3 Creación de Sockets en UNIX

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

Función socket

La función se define:

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

Donde el parámetro *familia* puede ser alguno de estos listados abajo.

AF_INET: Conjunto de protocolos TCP/IP
AF_UNIX: Dominio de UNIX(Comunicaciones locales)
AF_DECnet: Red DECNET
AF_SNA:SNA de IBM

Para consultar otras familias véase el archivo de encabezados socket.h

El parámetro *servicio* determina el tipo de servicio de transporte que se desea y este debe ser proporcionado por los protocolos que se encuentren en la familia especificada. Los valores posibles de la familia AF_INET son:

SOCK-STREAM	Este es un socket stream usado para la conexión-orientada, es un servicio confiable de una secuencia de bytes sin límite de registros condicionado por el protocolo TCP en la capa de transporte.
SOCK-DGRAM	Este es un socket de entrega de paquetes usado para servicios sin conexión no confiable proporcionado por el protocolo UDP en la capa de transporte.
SOCK-RAW	Es usado para interfaces directas con el protocolo IP en la capa de red.

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

5.3.1 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, en consecuencia ningún mensaje puede ser recibido en él. La comunicación de procesos es ligada por una asociación. En el dominio internet, una asociación es compuesta de nombre, 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).

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

Para asociar el socket con su dirección y un número de puerto se usará la función bind. La función bind se utiliza para especificar la dirección del socket y un número de puerto.

Función bind

Ya creado el socket, es necesario especificar una dirección para él. Se define así:

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

Donde el parámetro *sock* es el descriptor obtenido de la función socket. *Dir* es el apuntador hacia la estructura en donde se almacena la dirección del socket; para la familia AF_INET, la definición de la estructura sockaddr describe la dirección de un socket. Esta estructura se define en el siguiente punto.

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

5.3.2 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 solo 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 pueden comunicarse con los BSD si un proceso de translación 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 bytes de dirección máximo*/
};
```

El campo sa_family de la estructura especifica el tipo de dirección, el campo sa_data contiene la dirección actual.

5.3.2.1 Dominio UNIX

En este dominio los sockets son llamados BSD, que son utilizados para la comunicación de tipo local entre procesos, 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*/
        char sun_data;             /*referencia de dirección*/
};
```

5.3.2.2 Dominio INTERNET

Este dominio es usado para las direcciones INTERNET, y la estructura que lo define es sockaddr_in, que se utiliza para designar un servicio de red sobre cierta máquina en particular; A continuación se muestra la estructura:

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

donde la estructura in_addr se define así:

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

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

5.4 Obtención de Información de estado

Para obtener información sobre una conexión se usan varias funciones de estado en cualquier momento, aunque se utilizan comúnmente para establecer la integridad de la conexión en caso de problemas o para controlar el comportamiento del socket.

Las funciones de estado requieren el nombre de la conexión local y devuelven un conjunto de información, que puede incluir los nombres de los sockets local y remoto, nombre de la conexión local, número de buffers esperados, número de buffers que están esperando datos y valores actuales del estado urgente, de precedencia y de seguridad.

Dos funciones proporcionan información sobre la dirección local de un socket. La función *getsockpeername* devuelve la dirección del extremo remoto. La función *getsockname* devuelve la dirección local de un socket. Tienen los siguientes formatos.

```
getpeername( socket,struct sockaadr *dir ,longitud_dirección)
getsockname(socket,struct sockaddr * dir,longitud_dirección)
```

gethostname y sethostname son funciones que permiten que una aplicación obtenga y establezca el nombre del servidor. Sus formatos son:

```
sethostname(char nombre, int longitud)
gethostname(char nombre, int longitud)
```

La función gethostbyname, consigue la información correspondiente al nombre del host. Su formato es:

getsockbyname(char *nombre)

Donde *nombre* es el apuntador al nombre del host. Esta función regresa un apuntador a la estructura ***hostent*** que se verá con más detalle en el siguiente punto.

5.5 Servicio de Nombres

La forma para asignarle un valor al campo de direcciones del host en la estructura sockaddr_in; es en primer lugar obtener la dirección del host (debe ser de 32 bits), después obtener el nombre simbólico del servidor por la función gethostbyname y recibir un apuntador para la estructura hostent mostrada a continuación:

```
struct hostent{
        char *h_name;           /*Nombre oficial del host*/
        char **h_alias;         /*Lista de alias*/
        int  h_addrtype;        /*Tipo de dirección del host*/
        int  h_length;          /*Longitud del host*/
        char **h_addr_list;     /*Lista de direcciones del nombre del servidor*/
};
```

Esta estructura es necesaria porque el nombre del servidor podría ser asociado con varias direcciones IP o viceversa). En muchos casos, la aplicación solo toma la primer dirección en h_addr_list. Para hacer esto más sencillo, el símbolo h_addr_list es definido como h_addr_list[0].

5.6 Conexión de sockets

El protocolo TCP requiere, que antes de usar un socket, éste se conecte a un punto final remoto; por ejemplo, un cliente del servicio ***who is*** debe conectarse al puerto 41 del servidor que va a prestar servicio. La función connect se utiliza para dicha conexión.

Función connect

Se define así:

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

Los parámetros especifican el socket local que se va a conectar, un apuntador hacia la dirección del socket remoto con el que se hará la conexión, no es necesario llamar a ***connect***; si se hace la llamada, solo se almacenará localmente la dirección, para usarla en otras funciones. Connect regresa un entero positivo si no hay errores y -1 si los hay.

Mientras que los procesos clientes tienen que utilizar la función ***connect*** para inicializar la conexión, los procesos servidores deben utilizar la función ***accept*** para reconocer las conexiones pedidas a sock, el cuál debe ser un socket fijo a una dirección específica(utilizando la función ***bind***).

Función accept

Esta función crea un nuevo socket conectado con el nodo remoto que inició la conexión (con connect) y regresa su descriptor como valor. Al terminar la función, los parámetros dir y longitud apuntan a la dirección del nodo remoto, y a la longitud de dicha dirección y se define así:

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

Donde socket es el socket en el cual se aceptan solicitudes, dir es un apuntador la una estructura y longitud es un apuntador a un entero que muestra la longitud de la dirección.

Un servicio puede recibir varias peticiones de conexión simultáneamente. La función listen maneja las solicitudes de datos y los problemas que pueden ocurrir con este tipo de comportamiento mediante el establecimiento de una cola de espera para las solicitudes de conexión que lleguen. La cola impide cuellos de botella y colisiones, como cuando llega una nueva solicitud antes que una anterior haya sido totalmente manejada, o cuando llegan dos solicitudes en forma simultánea.

La función listen establece un buffer para poner en cola las solicitudes de llegada, con lo que evitará perdidas.

Función listen

El formato de la función es:

```
listen (sock, longitud_cola)
```

Donde longitud_cola es el tamaño del buffer de llegada

Orden de los bytes

Un problema de particular importancia, cuando diseñamos aplicaciones que son esperadas para trabajar en redes híbridas es el orden de los bytes. Algunas arquitecturas son *big-endian*(el byte más significativo es el primero); otras son *little-endian*(el byte más significativo es el último). Un ejemplo de este último es la familia Intel de procesadores, CPUs DEC y Motorola. Los números internet siempre son *big_endian.* Para asegurar que las versiones sean correctas de acuerdo a la plataforma que estemos usando, podemos usar las funciones htonl, htons, ntonl y ntons.

Estas cuatro funciones proporcionan la translación del orden de los bytes. La función htonl es para valores de 32 bits y htons para valores de 16 bits, convierten valores de orden **Intel** a **Internet**. Las funciones ntohl para valores de 32 bits, y ntohs, para valores de 16 bits proporcionan la conversión del orden ***Internet*** **a Intel**

5.7 Destrucción de sockets

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

Función close

Limpia los buffers internos del sistema operativo eliminando la información que pudo haber quedado acumulada en un socket, Esta función se define así:

int close(int sock)

5.8 Comunicación a través de los Sockets

En el caso del protocolo TCP la aplicación servidor hace una conexión a un puerto TCP específico y luego utiliza la función listen para monitorear posibles conexiones, inmediatamente después de llamar a la función listen, se llama a la función accept para esperar las conexiones que lleguen.

Las llamadas de funciones tanto en el servidor como en el cliente queda definido como:

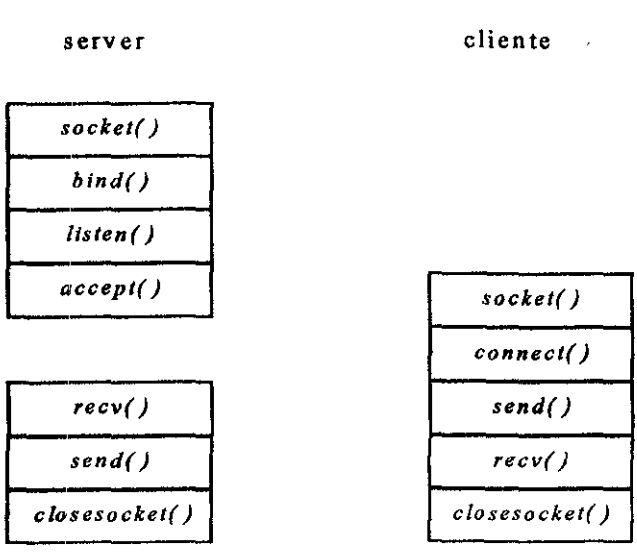


Figura 5.1 Diagrama de funciones cliente-servidor

5.9 Sockets y DLL's

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 las bibliotecas especialmente construidas pueden ser cargadas y enlazadas al mismo 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 C++. Los desarrolladores de la biblioteca de clases "Microsoft 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.

5.10 Sockets para Windows (Winsocks) y sus funciones

La especificación de los sockets Windows definen una interfaz de programación de red para Microsoft Windows la cual esta basada en el paradigma "socket" popularizado en la distribución de software Berkeley (BSD 4.3 UNIX) de la Universidad de California en Berkeley. Esta especificación fué desarrollada por un consorcio de organizaciones con la finalidad de definir una interfaz común de programas de aplicación(API) que pudiera ser usada por desarrolladores de aplicaciones para elaborar aplicaciones de red.

Los sockets Windows(Winsocks) API no son una especificación de Microsoft, aunque Microsoft participó en el diseño.

Los Winsocks fueron diseñados para ser muy similares a los sockets Berkeley, así que si se tiene experiencia en programación con sockets en UNIX será fácil hacer la transición a sockets Windows. Se podría decir que los sockets Windows son sockets Berkeley con un número de extensiones Windows específicas.

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

Winsock es una librería dinámica de enlace (DLL), en la que se encuentran las funciones necesarias para la comunicación en Internet, y se ejecuta bajo ambientes Windows 3.x, Windows para trabajo en grupos, Windows NT y Windows 95.

Es importante mencionar acerca del Winsock.DLL que éste debe ser similar a la versión de TCP/IP que se está ejecutando.

A continuación se nombran los propósitos del diseño de los sockets Windows:

- Proporcionar una API común para desarrolladores de programas y para vendedores de software de red que siguen las normas establecidas.
- Facilitar el cambio de código existente de sockets Berkeley a ambiente Windows.
- Proporcionar una interfaz independiente del protocolo.
- Tomar ventaja de la capacidad del sistema de mensajes de ambientes Windows.
- Atender las necesidades sin derecho preferente, lo mismo que los multienlaces con derecho preferente de ambientes Windows.

5.10.1 Modelo de capas de los sockets Windows

Aplicación Winsock
Winsock API
Sockets Windows DLL
Conjunto de Protocolos API
Conjunto de Protocolos (p. ej. TCP/IP)
Controlador de hardware API
Controlador de hardware(NDIS, ODI, Packet Driver)
Interfaz de hardware
Red (hardware) Interfaz (hardware específico)

Figura 5.2 Modelo de capas del Winsock

En esta representación de capas, la especificación sockets Windows define el nivel más alto de la librería de enlace de datos(DLL) de los sockets Windows lo mismo que el Winsock API. El Winsock API proporciona la interfaz para desarrolladores de aplicaciones. El desarrollo del conjunto de protocolos de red se realiza en las capas inferiores. Los fabricantes de interfaces de red proporcionan en la capa más baja los manejadores de dispositivos para atar su tarjeta de interfaz de red(NIC) en este modelo. Utilizando este modelo de capas, los desarrolladores de aplicaciones se abstraen de los fundamentales detalles de conexión de protocolo y hardware y se apartan de cualquier dependencia de hardware.

Pasos básicos para usar sockets Windows en una aplicación:

1. Inicializar la interfaz del socket.
2. Abrir(crear) un socket.
3. Atar el socket a la dirección de máquina y punto final del servicio.
4. Poner al servidor en un estado de atención.
5. Aceptar conexiones del cliente.
6. Realizar llamadas el cliente.
7. Intercambiar datos(Recibir/Enviar).
8. Cerrar el socket.
9. Destruir(limpiar) la interfaz del socket.

5.11 Funciones del Winsock API

En la tabla siguiente se muestran algunas funciones del Winsock API:

Función Winsock API	Descripción
accept()	Permite a otras computadoras conectarse a tu computadora
bind()	Define parámetros para un socket recién creado
Closesocket()	Elimina un socket
Connect()	Inicia una conexión a otra computadora
Getpeername()	Obtiene la dirección de red de la computadora conectada al socket
Getsockname()	Obtiene información acerca del socket relacionada con su direccionamiento interno
htonl()	Convierte valores de 32 bits de orden Intel al orden de Internet
htons()	Convierte valores de 16 bits de orden Intel al orden de Internet
inet_addr()	Convierte una cadena conteniendo una dirección en notación punto a un valor de 32 bits en ordenamiento de bytes de Internet
inet_ntoa()	Convierte un valor de 32 bits en ordenamiento de bytes de Internet a una cadena conteniendo una dirección en notación punto
listen()	Indica a la computadora esperar conexiones en un puerto en particular
nothl()	Convierte valores de 32 bits Internet al orden de Intel
ntohs()	Convierte valores de 16 bits Internet al orden de Intel
recv()	Lee datos del buffer de la red al puerto especificado
Recvfrom()	Lee datos del buffer de la red al puerto especificado y almacena la dirección de la computadora que envía los datos
send()	Envía datos a un socket específico
sendto()	Envía datos a un socket específico con parámetros adicionales
Shutdown()	Deshabilita lecturas y escrituras en el socket especificado
socket()	Crea un socket
WSAAsyncGetHostByAddr()	Provee una llamada sin bloqueo que obtiene el nombre DNS de una computadora especificando su dirección
WSAAsyncGetHostByName()	Provee una llamada sin bloqueo que obtiene los 32 bits de la dirección del sistema de orden Internet especificando el nombre DNS
WSAAsyncSelect()	Especifica las acciones que regresarán los mensajes a la aplicación en un particular socket sin bloqueo
WSACancelAsyncRequest()	Cancela una llamada sin bloqueo que no ha sido completada
WSACancelBlockingCall()	Cancela una llamada bloqueada que no ha sido completada
WSACleanUp()	Cancela toda actividad que la aplicación ha iniciado con el Winsock.DLL
WSAgetLastError()	Recupera el código de error de una función que regresa una condición de error

WSAIsBlocking()	Verifica si una operación bloqueada está avanzando
WSAStartup()	Inicializa el Winsock.DLL para que funcione con la aplicación

Tabla 5.1 Funciones Winsock API

A continuación se incluyen algunas rutinas de los sockets, describiendo cada una:

accept()

Descripción Acepta una conexión en un socket.

#include <winsock.h>

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

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

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

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

bind()

Descripción Asocia una dirección local con un socket.

#include <winsock.h>

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

s — Un descriptor identificando un socket sin liga.

name — La dirección que se asigna al socket. La estructura sockaddr está definida como sigue:

```
struct sockaddr {
        u_short         sa_family;
        char            sa_data[14];
};
```

namelen — La longitud de name

closesocket()
Descripción Cierra un socket.

#include <winsock.h>

int PASCAL FAR closesocket(SOCKET s);

s	Un descriptor identificando un socket

connect()
Descripción Establece una conexión a un socket remoto.

#include <winsock.h>

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

s	Un descriptor identificando un socket sin conexión.
name	El nombre del socket remoto al cual el socket va a ser conectado.
namelen	La longitud de name.

getsockname()
Descripción Obtiene el nombre local para un socket.

#include <winsock.h>

int PASCAL FAR getsockname(SOCKET s, struct sockaddr FAR * name, int FAR * namelen);

s	Un descriptor identificando un socket sin liga.
name	Recibe la dirección(nombre) del socket.
namelen	El tamaño del buffer name.

htonl()
Descripción Convierte un u_long (entero de 32 bits) de orden Intel a orden Internet.

```
#include <winsock.h>

u_long PASCAL FAR htonl(u_long hostlong);
```

hostlong Un número de 32 bits de orden Intel.

htons()
Descripción Convierte un u_short (entero de 16 bits) de orden Intel a orden Internet.

```
#include <winsock.h>

u_short PASCAL FAR htons(u_short hostshort);
```

hostshort Un número de 16 bits en orden de bytes de host.

inet_addr()
Descripción Convierte una cadena con el nombre del servidor a su dirección IP correspondiente.

```
#include <winsock.h>

unsigned long PASCAL FAR inet_addr(const char FAR * cp);
```

cp Una cadena de caracteres que representa un número expresado en la notación Internet.

listen()
Descripción Habilita a un socket para aceptar nuevas conexiones.

```
#include <winsock.h>

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

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

backlog La máxima longitud de la cola de conexiones pendientes.

ntohl()
Descripción Convierte un u_long de orden Internet a orden Intel.

```
#include <winsock.h>
```

u_long PASCAL FAR ntohl(u_long netlong);

netlong Un número de 32 bits en orden de bytes de red.

ntohs()

Descripción Convierte un u_short de orden Internet a orden Intel.

#include <winsock.h>

u_short PASCAL FAR ntohs(u_long netshort);

netshort Un número de 16 bits en orden de bytes de red.

recv()

Descripción Recibe datos de un socket.

#include <winsock.h>

int PASCAL FAR recv(SOCKET s, char FAR * buf, int len, int flags);

s Un descriptor identificando un socket conectado.

buf Un buffer para el dato que entra.

len La longitud de buf.

flags Especifica el camino en el que la llamada es realizada.

send()

Descripción Envía datos a un socket conectado.

#include <winsock.h>

int PASCAL FAR send(SOCKET s, const char FAR * buf, int len, int flags);

s Un descriptor identificando un socket conectado.

buf Un buffer para el dato a ser transmitido.

len La longitud del dato en buf.

flags Especifica el camino en el que la llamada es realizada.

socket()
Descripción Crea un socket.

#include <winsock.h>

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

af	Un formato de especificación de dirección. El único formato actualmente soportado es PF_INET, que es el formato de direcciones Internet ARPA.
type	Una especificación de tipo para el nuevo socket.
protocol	Un protocolo en particular para ser usado con el socket, o "0" si la llamada no desea especificar un protocolo.

5.12 Puertos de Entrada/Salida

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

5.12.1 Tipos de Puertos

Un puerto es un entero sin signo de 16 bits, y su rango de valores es de 0 a 65535. Los puertos de las *aplicaciones cliente* son dinámicamente asignados por la operación del sistema donde hay una petición de servicio. Los puertos conocidos para las *aplicaciones servidor* son preasignados por la IANA (Internet Assigned Numbers Authority) y no cambian.
Los puertos no son equivalentes a otros. Ellos son clasificados en diferentes grupos con diferentes capacidades. La siguiente tabla presenta las categorías de los puertos, su rango y una breve descripción de su categoría. Existen dos parámetros para poder comunicarse con un proceso: la dirección Internet del host y un número de puerto único asignado por el sistema.

Descripción	Rango de Puertos
Puerto asignado por el sistema	0
Puertos reservados	256-1024
Puertos conocidos	1-255
Autoasignados por rresvport()	512-1023
Puertos no reservados	1024-65535

Tabla 5.2 Número de Puertos

En casos donde no necesitamos tener un puerto específico asignado a un socket, como proceso el sistema puede asignar automáticamente un puerto no usado. Esto puede hacerse antes de la llamada a la función **bind()**. El puerto asignado por el sistema esta dentro del rango 1024 a 5000.

La autoasignación de un puerto regular esta echo con ayuda de la llamada el sistema de **getsocksame()**. Esta llamada al sistema regresa el nombre asociado con un socket. También regresa esta la dirección local y elementos del proceso local de una asociación. Para permitir la asignación automática del puerto, la aplicación debe especificar el puerto 0 antes de la llamada **bind()** y después la llamada **getsockname()**

Cada protocolo utiliza un puerto único TCP para su servicio, de esta forma muchos servicios pueden ser ejecutados en una computadora debido a que cada puerto esta monitoreando la posible conexión de este.

Puertos reservados 256 al 1024

Los puertos del 256 al 1024 son reservados para servicios específicos o privados de UNIX
A un proceso no se le permite encadenarse a un puerto reservado sin su usuario efectivo ID 0.

El trabajo de la llamada al sistema rresvport() es tener una máquina automáticamente asignada a un puerto reservado. Esta función crea un socket Internet stream y encadena un puerto reservado al socket. El socket descriptor es regresado como valor de la función, a menos que ocurra un error.

Puertos conocidos 1 al 255

Generalmente, los números inferiores al 256 se utilizan para procesos de uso frecuente (como TELNET y FTP). Internet Assigned Numbers Authority publica una lista de los números de puerto de uso frecuente. Estos también son referidos a las direcciones conocidas para entrega de mensajes. Es una técnica usada por los protocolos como TCP y UDP para identificar servicios que un host puede proporcionar.
Todos los puertos conocidos están en el rango de 1 a 255. Para listar servicios validos para el sistema UNIX. En la siguiente tabla 5.3 se muestran algunos puertos comúnmente usados por TCP

Número de Puerto	Descripción
21	FTP
23	Telnet
25	SMNP
53	Domain Name System (DNS)
139	Servicio de sesión NetBios

Tabla 5.3 Puertos conocidos

Para tener una referencia completa simplemente se puede examinar el archivo */etc/services* del sistema UNIX, en el Apéndice D se muestra este archivo para servidores SUN SPARC UNIX.

Puertos no reservados

Puertos no reservados 1024 al 65535 son considerados no privilegiados y pueden ser usados por algún proceso.

5.13 Relación entre un Puerto y un Socket

Los puertos conocidos son puertos estandarizados que habilitan a computadoras remotas para saber a que servicio de red se pueden conectar. Esto simplifica los procesos de conexión porque ambos, el cliente y el servidor, saben de hecho que los datos son enviados por un proceso específico usando un puerto específico.

Si en una computadora existen múltiples conexiones a un mismo servicio de puerto, éste debe ser capaz de recibirlas, de aquí se deriva el concepto de socket. La figura 5.3 nos permite mostrar esta idea.

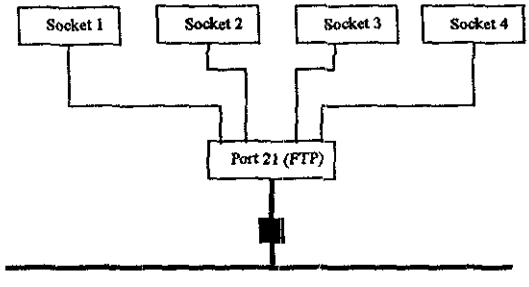


Fig. 5.3 Los sockets permiten conexiones múltiples en el mismo puerto TCP/IP

Un socket representa una conexión específica entre dos computadoras en la red usando un servicio particular. Cuando ocurre una conexión, la computadora acepta la petición en el puerto y asigna la conexión a un número particular de socket. Por un lado, otras conexiones pueden ser atendidas en el mismo puerto, pero ser asignadas a diferente número de socket.

La combinación de una dirección IP y un número de puerto es llamado socket. Un socket únicamente identifica un proceso de red dentro de un entorno Internet. Algunas veces "socket" y "puerto" son usados indistintamente. De hecho, los servicios conocidos son frecuentemente referidos como sockets conocidos.

Capítulo 6

Diseño e Implementación del Proyecto

Introducción

En este capítulo veremos el diseño, así como la implementación del proyecto para reportar fallas en equipos de cómputo SPARC por medio de síntesis de voz.

El diseño del sistema se divide en dos partes: cliente y servidor, donde el cliente es un host SPARC UNIX y el servidor una computadora personal (PC) con ambiente gráfico Windows que esta alerta a las peticiones del cliente.

Los programas para el cliente se realizaron en lenguaje "C" ANSI para UNIX y utilerías de C-Shell, utilizamos también funciones de comunicación propias del sistema con el conjunto de protocolos TCP/IP.

Los programas del servidor, se implementan para ambiente Windows en lenguaje Visual "C++", para ayudarnos en la programación utilizamos las funciones de la biblioteca de enlace dinámico winsock.dll ver. 1.1, en lo que se refiere a la comunicación en Internet y funciones de la biblioteca FB_SPCH.DLL, para habilitar la tarjeta de sonido que reproduce con síntesis de voz los mensajes de error generados en el cliente. En este capítulo daremos una explicación más detallada sobre los programas para el cliente y el servidor para dar como resultado final el sistema de monitoreo.

6.1 Diseño Técnico

En este apartado explicaremos la integración del sistema en sus dos partes: software en Windows y software en UNIX mediante los sockets.

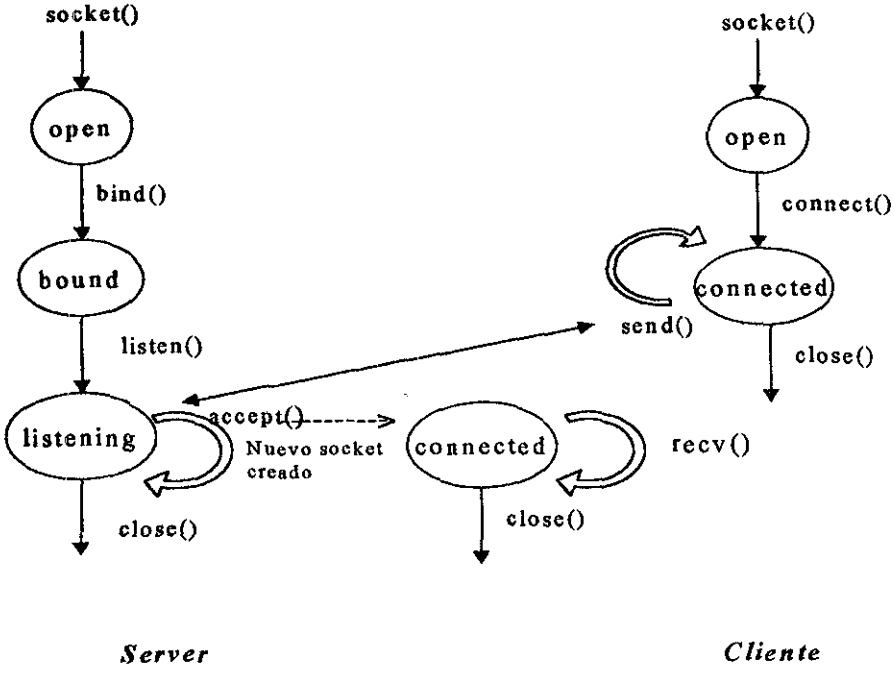


Figura 6.1 Diseño técnico.

En la figura 6.1 observamos el ciclo de los sockets, con el cual se logra el enlace dinámico de la información.

Como ya se explicó anteriormente, el primer punto para realizar la comunicación es definir el socket, en donde se incluyen las características del mismo con su puerto correspondiente.

En el siguiente punto veremos como funcionan paso a paso tanto el servidor como el cliente cuando realizan la conexión.

6.1.1 Comunicación por sockets Berkeley/Winsockets

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

6.2 Funcionamiento del servidor.

Es el encargado de escuchar las peticiones realizadas por parte del cliente, así mismo, actuar de acuerdo a la información recibida.

1. Define socket
2. Enlace. En este punto se asocia el socket a las características definidas anteriormente como dirección, puerto y 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 ésta es aceptada mediante un nuevo socket, el cual permite la llegada de información.
5. La información es guardada en un buffer y tomada por el programa que reproduce este texto a voz
6. Cierra. En el momento en el que se desee no aceptar más entradas, se procede a cerrar el socket.

Es indispensable cerrar el socket antes de volver a definirlo, es decir, no es posible crear un mismo socket si éste no ha sido cerrado previamente

6.2.1 Programación de las Funciones para el Servidor en Windows

En este programa en Visual "C++" utilizamos las funciones del winsock.DLL para la comunicación con el host UNIX desde un computadora personal (PC) bajo ambiente Windows.

Este módulo del servidor esta dividido en dos partes: la primera define las características de la ventana de salida en ambiente gráfico, como bordes, color de la ventana y cuadros de dialogo, así como la función de manejo de eventos para programas en Windows; la segunda parte es el conjunto de instrucciones dadas por la biblioteca winsock.DLL como las del propio lenguaje para definir un socket que utilice el conjunto de protocolos TCP/IP y pueda ***recibir*** peticiones de conexión hacia el servidor, habilitando un puerto de comunicación.

A continuación mostraremos el código.

```
/*Programa que hace la función de servidor que recibe peticiones de conexión a el. */
#include <windows.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include  <dos.h>
#include  <conio.h>
#include  <memory.h>
#include "c:\msvc\progra~1\winsock.h"


/*La siguiente función WindowdProc define las características de la
ventana de salida, como bordes, color de la ventana y cuadros de dialogo,
así como la función  de manejo de eventos para programa */

long FAR PASCAL  WindowProc(HWND hWnd, UINT messg, WPARAM wParam,LPARAM
lParam);

char szProgName[]="ProgName";
int PASCAL WinMain(HINSTANCE hInst,HINSTANCE hPreInst,LPSTR lpszCmdLine,
int nCmdShow)
{
  HWND hWnd;
  MSG  lpMsg;
  WNDCLASS  wcApp;
  if(!hPreInst)
  {
    wcApp.lpszClassName=szProgName;
    wcApp.hInstance=hInst;
    wcApp.lpfnWndProc=WindowProc;
    wcApp.hCursor=LoadCursor(NULL,IDC_ARROW);
    wcApp.lpszMenuName=NULL;
    wcApp.hIcon=NULL;
    wcApp.hbrBackground=GetStockObject(WHITE_BRUSH);
    wcApp.style=CS_HREDRAW|CS_VREDRAW;
    wcApp.cbClsExtra=0;
    wcApp.cbWndExtra=0;
    if(!RegisterClass(&wcApp))
      return FALSE;
  }


  /*Función para crear una ventana */
  hWnd=CreateWindow(szProgName,"crea   un   DLL",   WS_OVERLAPPEDWINDOW,
CW_USEDEFAULT,CW_USEDEFAULT,  CW_USEDEFAULT,  CW_USEDEFAULT,  (HWND)NULL,
(HWND)NULL, (HANDLE)hInst, (LPSTR)NULL);

/*Visualización de la ventana */
  ShowWindow(hWnd,nCmdShow);
  UpdateWindow(hWnd);
```

```
/*Creación de un ciclo de mensaje*/
  while (GetMessage(&lpMsg,(HWND)NULL,0,0)){
      TranslateMessage(&lpMsg);  /*Permite el uso del teclado */
      DispatchMessage(&lpMsg);   /*Devuelve el control a Windows */
  }
  return(lpMsg.wParam);

/*Windows llama a esta función que recibe mensajes de la cola de mensajes
*/

}

/*Función que habilita un puerto de comunicación por medio de los
winsockets */

long FAR PASCAL  WindowProc(HWND hWnd, UINT messg, WPARAM wParam,LPARAM
lParam)
{
struct sockaddr_in server;
int   sock,length;
int   msgsock;
char  buf[80];
int   rval;
WSADATA wsadata;


   if(WSAStartup(0x101,&wsadata))
                /*En primer lugar antes de definir una aplicación o DLL
                debe ir la función Wsastartup que especifica la versión
                de la DLL, en nuestro caso especificaremos la del
                winsock API requerido y recuperar detalles de esta .*/
   {
     printf("No puede inicializar el winsock \n");
     exit(1);
   }
   if ((sock = socket(AF_INET,SOCK_STREAM,0)) < 0)
           /*Creamos un socket, especificando formato, tipo y protocolo
           a utilizar. */
   {
     perror("Error no puede crearse el socket");
     exit(1);
   }

   server.sin_family = AF_INET;     /*Asignamos a la estructura server,
   server.sin_port = 0;             datos para crear socket*/
   server.sin_addr.s_addr = INADDR_ANY;

   if (bind(sock,(struct sockaddr *)&server,sizeof(server)) < 0)

           /*La función bind nos asignara un nombre al socket creado */
   {
     perror("Error no puede asignarse nombre en la función bind");

    exit(1);
   }
```

```
   length=sizeof(server);

if (getsockname(sock,(struct sockaddr *)&server,&length) == -1)
                  /* getsockname función recupera el nombre del socket*/

   {
     perror("Error al asignar el nombre al socket");
     exit(0);
   }
   printf("\nEl # de socket es:  %d",htons(server.sin_port));

   listen(sock,5);       /*Esta función crea una cola para 5 conexiones mas
                         a un puerto especifico ya creado */

   accept(sock,(struct sockaddr *)0,(int *)0); /*Accept permite al socket
                              que oiga peticiones para crear conexiones a
                             un nuevo socket y asignarle un número*/

   do{
      msgsock = accept(sock,(struct sockaddr *)0, (int *)0);
      if(msgsock==-1)
        perror("Acepta la conexión");
      else
      do{
        memset(buf,0,sizeof(buf));
        if((rval=recv(msgsock,buf,80,MSG_PEEK))==-1)
          perror("Error al leer el mensaje");
        if(rval==0)
          printf("Finalizando conexion\n");
        else
         printf("-->%s\n",buf);
    closesocket(msgsock);
      }while(rval != 0);
    }while(strcmp(buf,"fin"));
    exit(0);
}
```

6.2.2 Programa para la recepción de los mensajes utilizando una tarjeta de sonido

Programa hecho en Visual "C++" para habilitar voz en la tarjeta de sonido por medio de las funciones encontradas en la biblioteca FB_SPCH.DLL.

Las funciones que necesitamos de esta biblioteca son: openSpeech() que habilita el puerto de la tarjeta de sonido, say() función que se encarga de la conversión del texto a voz, cabe mencionar que el texto que se convertirá a voz es el enviado por el cliente UNIX para avisar al operador que el sistema tiene algún problema y closeSpeech() que cierra el puerto de la tarjeta.

```
long FAR PASCAL  WindowProc(HWND hWnd, UINT messg, WPARAM wParam, LPARAM
lParam)
{
long ISCB;
ISCB = OpenSpeech(0,0,NULL); /*Openspeech función que inicializa
                                    /*la tarjeta */
Say(hWnd, "Hello World.");        /*Función que se encarga de la
conversión de texto a*/
Say(hWnd, "Hello Again.");      /* voz*/
CloseSpeech(hWnd);             /*Deshabilita  la tarjeta de sonido*/
}
```

6.3 Funcionamiento del Cliente.

Éste realiza peticiones al servidor, para que 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 el destino no se encuentre en el estado 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.

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

6.3.1 Programación de Funciones para el cliente en UNIX

El programa cliente esta hecho en lenguaje ANSI "C" para UNIX, que va a realizar la comunicación con otro equipo de cómputo, en este caso una computadora personal (PC), en la cual ya deberá estar corriendo el programa servidor que recibirá sus peticiones.

El cliente usa las bibliotecas de comunicación propias del Sistema Operativo que son: socket.h, in.h y netdb.h, las cuales definen funciones para configurar y habilitar los puertos de E/S, definir estructuras de las direcciones IP, así como la obtención de información de estado de las estaciones de trabajo.

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

A continuación daremos el código fuente para crear un socket en UNIX.

```
/*Programa que hace la función del cliente, envía peticiones de conexión
hacia el servidor. */

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

#define BUFSIZ 1024

main(int argc, char *argv[])
{
  int sock;
  char DATA[80];
  struct sockaddr_in server;
  struct hostent *hp, *gethostbyname();
  char buf[BUFSIZ+1];

  if((sock=socket(AF_INET,SOCK_STREAM,0))<0)
  {
    perror("No se puede crear el socket");
    exit(1);
  }

  server.sin_family=AF_INET;
  hp=gethostbyname(argv[1]);

  if(hp==(struct hostent *)0)
  {
    fprintf(stderr,"%s: Host desconocido \n",argv[1]);
    exit(2);
  }

  memcpy((char *)&server.sin_addr,(char *) hp->h_addr,hp->h_length);
  server.sin_port=htons(atoi(argv[2]));

  if(connect(sock,(struct sockaddr *)&server,sizeof server)<0)
  {
    perror("error en la conexion");
    exit(1);
  }


  gets(DATA);
  if(write(sock,DATA,strlen(DATA))==-1)
  {
     perror("Error de escritura");
     exit(1);
  }

  close(sock);
  exit(0);
```

}

6.3.2 Programa para obtener los mensajes de error críticos en el sistema

Este programa forma parte del software cliente, tiene como finalidad conseguir los errores del sistema que están registrados en un archivo de bitácora(messages).

La manera de obtener los mensajes de error es revisando periódicamente el archivo de bitácora, identificando con parámetros de búsqueda, palabras clave que indiquen fallas en el sistema, al ocurrir lo anterior, analizará éstas para enviar la información correspondiente al servidor.

La revisión de la bitácora se realiza cada minuto, para saber que un posible mensaje de error ha sido registrado en el sistema, verificamos el tamaño de la bitácora. Si existe una modificación en esta, .tomamos los mensajes generados en ese período, a continuación extraemos los mensajes que coincidan con las palabra clave.

Si en el procedimiento anterior obtenemos algún mensaje de error, hacemos una depuración de éste para analizarlo, utilizando una base de datos de errores.

La base de datos contiene una lista de los errores más frecuentes del sistema, a los cuales les asociamos el mensaje de alerta que enviará al servidor.

#Programa en C-Shell para detectar modificaciones en el archivo messages

```
#/bin/csh
#Programa para obtener mensajes de error

set bandera=1   #condicion para el while, siempre verdadera
set ban2=0      #para verificar la primera ocurrencia del programa
while ("$bandera" == 1)
set fecha="`date +%b%n%e%n%H:%M:%S`"    #fecha con segs para archivo messages
set fecha1="`date +%b%n%e%n%H:%M`"      #fecha para verificar arch. messages
set fecha2="`ls -la messages|cut -c42-53`"     #fecha del archivo messages
echo $fecha1
#echo $fecha2
if ($ban2 == 0) then
  set tamano="`ls -la messages|cut -c34-40`"    #inicializacion de variables
  set tamano1="`ls -la messages|cut -c34-40`"
else
  echo "entre al else"
  set tamano1="`ls -la messages|cut -c34-40`"   #asigna el tama%o actual de messages
endif
```

```
if ($tamano != $tamano1) then

   grep "`echo $fecha1`" < messages >> salida #si el archivo message cambia, ejecuta
verificacion
   grep "[Ee]rror" < salida|cut -d: -f4,5,6,7,8,9|cut -c3-18> salida1
   grep "`echo $fecha1`" < salida1 >> sal3
endif
@ tamano=$tamano1      #guarda el tama%o del archivo messages
set ban2=1
sleep 10
end
```

Programa que hace la comparación del mensaje de error detectado en el servidor SPARC con la base de datos, para posteriormente hacer la conexión al servidor por medio del módulo que programa el socket y enviar esta cadena de error encontrada en la base de datos.

```
/*Programa que hace la comparación del mensaje de error detectado en el servidor
SPARC*/

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

/* int  substr(char *,char*); */

void main()
{
 FILE *fp,*tempo,*salida;
 char arreg2[400]={'\0'},arreg4[400]={'\0'},arreg3[400]={'\0'};
 char ch,ch1,*p;
 int i,m=0,flag1,dosp,j,k,l,comp,pos_nu;
unsigned int length1=0;
unsigned int length2=0;
 flag1=dosp=j=i=0;

 if((fp=fopen("sal3","r"))==NULL)
   {
   printf("No puedo abrir el archivo error.dat  \n");
   exit(1);
   }

 if((tempo=fopen("base.dat","r"))==NULL)
   {
```

```
            arreg2[l]='\0';
            j=0;
        }
      else
        {
         arreg2[j]=ch;
         j++;
        }
    }while(ch!=EOF);
  fclose(fp);
  fclose(salida);
}
```

6.4 Integración del programa servidor.

Programa fuente del servidor, integrando los programas de comunicación y text-to – speech, que informan errores del host UNIX.

```
#include <windows.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include  <dos.h>
#include  <conio.h>
#include  <memory.h>
#include "c:\msvc\progra~1\winsock.h"

long FAR PASCAL WinProc(HWND hWnd, UINT messg, WPARAM wParam,LPARAM
lParam);
void FAR PASCAL CloseSpeech(long SCB);
long FAR PASCAL OpenSpeech(HWND hwnd, WORD mode, LPSTR voiceType);
int  FAR PASCAL Say(long SCB, LPSTR lpText);
long ISCB;
char szProgName[]="ProgName";
char buf[80];

int PASCAL WinMain(HINSTANCE hInst,HINSTANCE hPreInst,LPSTR lpszCmdLine,
int nCmdShow)
{

HWND hWnd;
  MSG  lpMsg;
  WNDCLASS  wcApp;
  if(!hPreInst)
  {
    wcApp.lpszClassName=szProgName;
    wcApp.hInstance=hInst;
    wcApp.lpfnWndProc=WinProc;
    wcApp.hCursor=LoadCursor(NULL,IDC_ARROW);
    wcApp.lpszMenuName=NULL;
    wcApp.hIcon=NULL;
```

```
    wcApp.hbrBackground=GetStockObject(WHITE_BRUSH);
    wcApp.style=CS_HREDRAW|CS_VREDRAW;
    wcApp.cbClsExtra=0;
    wcApp.cbWndExtra=0;
    if(!RegisterClass(&wcApp))
      return FALSE;
  }

hWnd=CreateWindow(szProgName,"crea un DLL",
WS_OVERLAPPEDWINDOW,CW_USEDEFAULT,CW_USEDEFAULT,CW_USEDEFAULT,CW_USEDEFAU
LT,(HWND)NULL,(HWND)NULL,(HANDLE)hInst,(LPSTR)NULL);
  ShowWindow(hWnd,nCmdShow);

  UpdateWindow(hWnd);
  while (GetMessage(&lpMsg,(HWND)NULL,0,0)){
      TranslateMessage(&lpMsg);
      DispatchMessage(&lpMsg);
  }

  return(lpMsg.wParam);

}


long FAR PASCAL  WinProc(HWND hWnd, UINT messg, WPARAM wParam, LPARAM
lParam)
{

struct sockaddr_in server;
int   sock,length;
int   msgsock;

//char buf[80];

int   rval;
WSADATA wsadata;

      ISCB = OpenSpeech(hWnd,0,NULL);


   if(WSAStartup(0x101,&wsadata))
   {
     printf("No puede inicializar el winsock \n");
     exit(1);
   }

   printf("Pasamos la primera prueba");


   if ((sock = socket(AF_INET,SOCK_STREAM,0)) < 0)
   {
     perror("Error no puede crearse el socket");
     exit(1);
   }
```

```
    server.sin_family = AF_INET;
    server.sin_port = 0;
    server.sin_addr.s_addr = INADDR_ANY;

    if (bind(sock,(struct sockaddr *)&server,sizeof(server)) < 0)
    {
      perror("Error no puede asignarse nombre en la funcion bind");

     exit(1);
    }
    length=sizeof(server);
    if (getsockname(sock,(struct sockaddr *)&server,&length) == -1)
    {
      perror("Error al asignar el nombre al socket");
      exit(0);
    }
    printf("\nEl # de socket es:  %d",htons(server.sin_port));

    listen(sock,5);



    do{
       msgsock = accept(sock,(struct sockaddr *)0, (int *)0);
       if(msgsock==-1)
         perror("Acepta la conexion");
       else
       do{
         memset(buf,0,sizeof(buf));
         if((rval=recv(msgsock,buf,80,MSG_PEEK))==-1)
           perror("Error al leer el mensaje");
         if(rval==0)
           printf("Finalizando conexion\n");
         else
         {
           printf("-->%s\n",buf);
           rval=0;
            Say(ISCB,buf);
         }
       }while(rval != 0);

     }while(strcmp(buf,"fin"));

      closesocket(msgsock);
      CloseSpeech(ISCB);
       printf("cierra sockets");
     exit(0);
}
```

6.5 Integración del programa cliente

Programa fuente del cliente, donde en este programa se integra la obtención del archivo de errores para el envió del mensaje a el servidor.

```
/* Programa para el cliente */
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

#define BUFSIZ 1024

main(int argc, char *argv[])
{
  int sock;
  char DATA[80]={'\0'};
  struct sockaddr_in server;
  struct hostent *hp, *gethostbyname();
  char buf[BUFSIZ+1];
  FILE *fp;
        int i,k;
        char ch;

  if((sock=socket(AF_INET,SOCK_STREAM,0))<0)
  {
    perror("No se puede crear el socket");
    exit(1);
  }

  server.sin_family=AF_INET;
  hp=gethostbyname(argv[1]);

  if(hp==(struct hostent *)0)
  {
    fprintf(stderr,"%s: Host desconocido \n",argv[1]);
    exit(2);
  }
}

  memcpy((char *)&server.sin_addr,(char *) hp->h_addr,hp->h_length);
  server.sin_port=htons(atoi(argv[2]));

  if(connect(sock,(struct sockaddr *)&server,sizeof server)<0)
  {
    perror("error en la conexion");
    exit(1);
  }
  if((fp=fopen("envia.dat","r"))==NULL)
    {
       printf("No puedo abrir el archivo envia.dat  \n");
       exit(1);
```

```
    }
    i=0;
  do{
      ch=getc(fp);
      if(ch=='\n')
        {
          DATA[i++]='\n';
          if(write(sock,DATA,strlen(DATA))==-1)
            {
               DATA[i]='\0';
               perror("Error de escritura");
               exit(1);
            }
          for(k=0;k<=i;k++)
         DATA[k]='\0';
         i=0;
        }
        else
        {
         DATA[i]=ch;
         i++;
        }
  }while(ch!=EOF);
fclose(fp);
close(sock);
exit(0);
}
```

6.6 Pruebas del Proyecto

6.6.1 Módulos usados en el equipo SPARC SUN

Programación del cliente y servidor.

El desarrollo del proyecto fue realizado en varias etapas, con la finalidad de entender cada uno de los conceptos y si existía un error fuera más fácil detectarlo.

Lo primero que hicimos fue programar las rutinas de un socket cliente y un servidor en lenguaje "C, usando el set de instrucciones ya implícito por el S.O. UNIX, el objetivo de esta práctica ya estableciendo la comunicación entre el cliente y servidor fue enviar una cadena tomada desde el teclado del cliente al servidor y desplegarla en el monitor.

La comunicación establecida se daba de la siguiente manera:

Al ejecutar el programa del servidor, verificamos que la conexión no tuviera problema, programando la rutina para que automáticamente el sistema nos asignará un número de puerto para establecer la comunicación con el cliente.

Para la ejecución del cliente incluimos la dirección IP del host y número de puerto asignado por el servidor para enviar una petición de conexión.

Para el módulo que obtiene el mensajes de error del sistema

Al establecerse la conexión entre el cliente y el servidor sin errores, lo que procedía era teclear cualquier palabra o mensaje y enviarlo al servidor; el servidor aceptaba el envío y lo visualizábamos desplegándolo en pantalla.

En esta parte uno de los problemas presentados fue que no definíamos correctamente las estructuras que se manejan para realizar la programación de los sockets.

La siguiente parte consistió en la obtención de los mensajes de error de la bitácora del sistema. Este punto tuvo algunos problemas ya que no analizamos en primer instancia como se manejaría la información.

Inicialmente verificaríamos todo el archivo de bitácora de principio a fin para encontrar algún mensaje que indicara error o falla en el sistema.

Buscábamos la palabra error en alguna línea de la bitácora, al encontrarla procedíamos a tomar todo el mensaje de error generado en ese instante. Este método funcionó en la pruebas que realizamos con una bitácora de prueba, solo que cuando utilizamos bitácoras de un sistema más complejo, el programa presentaba fallas, debido a que en un instante el sistema generaba en ocasiones mensajes de hasta 170 líneas que se tomaban y guardaban en memoria, provocando que se agotará la memoria.

Al no solucionar de manera aceptable este problema, buscamos otras alternativas, y reiniciamos el análisis de manera más detallada, este nos dio un nuevo enfoque y surgieron puntos que no habíamos tomado en cuenta, como son:

- No era necesario recorrer de principio a fin el archivo de bitácora.
- Lo importante era la línea donde se encontraba la línea "error".
- No era necesario tener todo el mensaje generado en un instante de tiempo (N líneas).
- El mensaje de error obtenido tenia que ser manipulado de tal forma que el aviso generado fuera entendible y diera una idea del problema presentado.
- Había que establecer un periodo de tiempo en el que se realizara un monitoreo del archivo de bitácora.

Tomando en cuenta los puntos mencionados anteriormente procedimos a buscar la forma de implementarlos. El lenguaje que utilizamos fué el C-shell de UNIX, por permitirnos manejar comandos propios del intérprete.

Para verificar el archivo messages tomamos el tamaño de este y lo asignamos a una variable, cada minuto se verifica si existe algún cambio y de existir se procede a obtener el mensaje que se genera en ese momento, a continuación buscamos la palabra clave "error" y

tomamos la línea que contiene ese parámetro, seleccionando la parte principal del mensaje de error.

Este mensaje es enviado tanto a una bitácora como a un archivo que servirá de entrada para el siguiente módulo

El siguiente módulo es el de comparación del mensaje de error con una base de datos de errores más comunes del sistema, en esta parte se toma la información del archivo generado anteriormente y se asocia con la información que envía al servidor.

6.6.2 Módulos usados en Windows

Investigamos la existencia de las bibliotecas dinámicas para comunicación existentes para Windows y encontramos que existía la biblioteca dinámica winsock dll versión 3.1 para Windows 3.1 en la cual basamos el desarrollo de este módulo y elegimos el lenguaje de Visual "C++" para programar los sockets ya que este nos permitiría el uso de las DLL's e instruccones estandares del mismo lenguaje.

En la elección del lenguaje tuvo que ver el conocimiento que teníamos de los lenguajes de programación y la estandarización con ANSI C

Para la programación de estas rutinas casi se tomo por completo como referencia la programación hecha en la primera práctica que fue la programación del servidor, el lenguaje ANSI C para UNIX, se le agrego a este la parte de configuración de la ventana en Windows y el módulo de manejo de mensajes.

En la programación del módulo para la conversión de texto a voz también se investigó el software que proporcionara una DLL que permitiera este cambio.

También usamos funciones de la biblioteca llamada FB_SPCH.DLL por medio de Visual C++ programando un módulo que convirtiera texto a voz.

Al final ya que las dos funciones entregaban resultados correctos se procedió a unirlas en un solo módulo llamado servidor

Conclusiones

El objetivo del proyecto fue cubierto en varias etapas que son: planteamiento del problema, definición de conceptos, desarrollo e implementación y análisis de resultados.

Planteamiento del problema: Hacemos un análisis de las necesidades que se tienen en el área de cómputo respecto al óptimo funcionamiento de los equipos.

Las alternativas que observamos viables dentro del análisis fueron:

1. *Un operador humano* dedicado exclusivamente a esta labor.
2. Obtención de algún software comercial que realice la administración sobre los errores del sistema(desventajas: alto costo, posible mayor requerimiento de hardware y software).
3. Nuestra propuesta que utiliza recursos existentes, *no requiere algún dispositivo* especial de hardware o software salvo en el equipo que se utiliza como servidor, el cual requiere una tarjeta de sonido y bocinas cuyo costo no es equiparable con el costo del punto 2.

Desarrollo e implementación:

Durante el desarrollo de este trabajo adquirimos conocimientos importantes en cuanto software de desarrollo para Windows y S.O Unix. El desglose de los capítulos 1 al 5 son los conocimientos básicos necesarios para el desarrollo de este trabajo, realizada en ambiente Windows e interactuando con servidores SPARC UNIX por medio de Sockets. De acuerdo con esto para nosotros es importante destacar las siguientes conclusiones sobre el desarrollo de sistema.

En esta parte comprendimos la diferencia entre un *socket* y un puerto y como hacer una conexión de red en ambiente UNIX. Con esto creamos los programas correspondientes al cliente y servidor utilizando lenguaje ANSI C de UNIX y las funciones para el manejo del conjunto de protocolos TCP/IP.

La programación de sockets en UNIX es similar a la que se realiza en Windows, por el lenguaje que utilizamos en estas dos interfaces que fue "C" que como sabemos existe un estándar entre estas dos plataformas.

En Windows ya que no existe como estándar la comunicación de redes usando el protocolo TCP/IP utilizamos las bibliotecas dinámicas *.DLL que nos ayudaron a resolver este problema porque ya que se creo un estándar para PC's llamado winsock.dll que contiene las funciones para la comunicación de red y compatible con la plataforma de Windows.

Consideramos que 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.

La integración Visual C++ y Winsockets, es altamente recomendada por nosotros, debido a su sencillez en el manejo y entendimiento, que nos permite crear una aplicación amigable

Usando una DLL para programar la tarjeta de sonido solo utilizamos 3 funciones con sus parámetros correspondientes y tuvimos como resultado lo que se llama texto-a-voz, donde las interpretaciones dadas por la tarjeta son bastante aceptables.

Análisis de resultados

Posibles aplicaciones del proyecto

1. Aplicarlo a cada uno de los servidores existentes de la DGB.
2. Apoyo a administradores de equipos de cómputo en las bibliotecas que integran el Sistema Bibliotecario de la UNAM.
3. Uso para administrar otras aplicaciones existentes, por ejemplo en la biblioteca donde tenemos que obtener las actualizaciones semanales de los acervos para las dependencias, enviarlas y realizar el proceso de actualización, verificando espacio de disco y bitácoras de los procesos lanzados, tareas del sistema y tareas del WEB que se encuentren habilitadas. Este proyecto haría automáticamente estas tareas y además sus bitácora la tendría como reporte hablado, mientras estuviera realizando cualquier otra actividad de trabajo.

Posibles mejoras del proyecto

Ampliarlo para aplicarse a sistemas donde se necesite comunicación abriendo puertos para enviar información no solo de texto sino ahora con el avance de la tecnología sonido e imágenes manipulando estos como sea necesario.

Hacemos hincapié en que este trabajo es una aplicación básica, la cuál tiene la finalidad de mostrar las herramientas tan poderosas como los sockets y DLL's y que se pueden hacer desarrollos tan extensos como la imaginación de los programadores lo permita.

Apéndice A

Características de Enterprise SunTM 5500

El contar con un equipo de estas dimensiones en la DGB conlleva a desarrollar a futuro varios proyectos, por lo mismo es conveniente conocer sus características para aprovecharlas al máximo, en este apéndice daremos las características del servidor **Enterprise SunTM 5500** ya que será la parte más importante en el desarrollo de todos los proyectos para la DGB, en cuanto al manejo de las bases de datos y digitalización de documentos de su acervo para la consulta vía Internet.

El servidor **Enterprise SunTM 5500** es un sistema confiable y poderoso con almacenamiento integrado, el cual con su storage, tiene más capacidad interna. Cuenta además con reconfiguración dinámica, sistema automático de recuperación y hardware proactivo de predicción a fallas. El manejo es simplificado por el software Sun Enterprise SyMONTM.

Últimos Avances

El sistema tiene la habilidad para reconfigurarse y reparar componentes del sistema mientras el servidor permanece en línea. Cuenta con una capacidad de mas de ½ TB y un alto desempeño de almacenamiento en un solo gabinete central, alto desempeño en las aplicaciones con sus 14 procesadores, componentes modulares, que simplifican los upgrades, y si su capacidad aumenta a 30 procesadores se tendrá un sistema Enterprise 6500.

Especificaciones Técnicas

Procesador

Número de procesadores	De 1 a 14
Arquitectura	Superescalar SPARC Versión 9,UltraSPARC
Caché por procesador	Primario: 16 KB de instrucciones, y 16 KB de datos por chip Secundario: caché externa de 4 MB
Interfaz del CPU	de 1 a 14, Ultra Puertos de 128-bits Slots de arquitectura (UPA)
Interconexión del Sistema	Gigaplane, 2.68 GB/seg (a 84 MHz), 3.2 GB/seg (a 100 MHz)

System Boards

Número de tarjetas	Ocho tarjetas máximo por sistema Configuración mínima requerida 1 CPU/Tarjeta de memoria y una tarjeta de E/S
Tarjeta de CPU/Memoria	Espacio para 2 procesadores y 6 SIMMS de memoria
Tarjeta de SBus E/S	Ofrece 2 canales SBUS, 3 slots SBus, SunFastEthernet, fast/wide SCSI-2, 2 sockets FC-AL
Tarjeta gráfica de E/S	Ofrece un canal SBUS, 2 slots SBUS un slot UPA para gráficos Creator y Creator3D, SunFastEthernet, fast/wide SCSI-2, 2 sockets FC-AL.
Tarjeta E/S PCI	Ofrece 4 canales PCI, 2 slots cortos PCI, SunFastEthernet, fast/wide SCSI-2

Memoria Principal

256 MB a 14 GB de capacidad de memoria por sistema 256 MB y 1 GB opción de expansión de memoria (cada grupo de 8 SIMMs)

Interfaces Estándar

Serial	2 Puertos RS-232/423
Sbus	Ancho del bus de datos de 64 bits, 25 MHz
PCI	Ancho del bus de 64 bits de datos, 66 MHz
Puerto para teclado y mouse	Uno por sistema
Ethernet	Par trenzado estándar de 10/100 MB/seg(10-BaseT y 100-BaseT) y/o MII transceiver por tarjeta de E/S
SCSI	Una tarjeta SCSI-2 de 20-MB/seg fast/wide (síncrona) por tarjeta de E/S
Canal de fibra	2 sockets en tarjeta(100 MB/seg full duplex) por SBus y tarjeta gráfica de E/S.

Mass Storage

Disco interno	Más ½ TB de almacenaje puede montarse en su gabinete
Cinta interna	DLT, 8mm, 4mm, y opción a cintas de 25-pulgadas
CD-ROM	SunCD® 32 drive estándar

Storage externo

Soporta almacenamiento arriba de 6 TB

Opciones de consola

Monitor	Monitor de color de 17 o 20
Frame buffer	TurboGX®, TurboGXplus®, Creator, o Creator3D frame buffer

Opciones de SBus

SunFastEthernet, SunATM® 155 y 622, Quad FastEthernet, SCSI-2 (DWIS/S) inteligente diferencial Fast/Wide SCSI-2 (SWIS/S) inteligente Single-Ended Fast/Wide Intelligent, interface serial de alta velocidad (HSI/S), Interface Token Ring (TRI/S), FDDI dual y simple, controlador serial paralelo (SPC/S), SCSI 100BaseT Fast/Wide, ISDN, Fast Differential SCSI-2 Buffered Ethernet (DSBE/S), Fast SCSI-2 Buffered Ethernet (FSBE/S), Sun Gigabit Ethernet, FC-AL

Opciones PCI

UltraSCSI de 10/100BaseT Fast/Wide, FastEthernet 10/100BaseT, Quad FastEthernet, Single y FDDI Dual-Attach, Token Ring, Interfaz serial de alta velocidad, Interfaz serial asíncrona, Red Ethernet, SunATM 155 y 622

Energía

Una fuente de poder periférica estándar de 184 watts. A actualizar por un módulo de energía de 300 watts (PCM) con redundancia, ventiladores de velocidad dual. Un PCM para dos tarjetas. Energía completamente redundante y temperatura baja dentro del sistema.

Características disponibles

Sistema Automático de recuperación
Reconfiguración dinámica
Ruteo alternativo
Módulos Hot-swap Power/Cooling
Hardware de predicción a fallas
Componentes modulares y más

Software

Sistema Operativo	Solaris 2.5.1 o posteriores
Lenguajes	C, C++, Pascal, FORTRAN, Java
Red	ONC, NFS, TCP/IP, SunNet® OSI, MHS, X.25, DCE, Netware
Sistema Windows	CDE, OpenWindows opcional Versión 3
Sistema de Monitoreo	Enterprise SyMON
Sistema y manejador de red	Solaris Web Start, ®Solstice AdminSuite, Solstice Domain Manager, ®Solstice Enterprise Manager, ®Solstice DiskSuite, ®Solstice Backup, File System y más

Ambiente

Energía AC	200-240 VAC, 47-63 Hz, 24 A
Operación	5° C a 35° C (41° F a 95° F) 20% a 80% humedad relativa, no condensada
No opera	-20° C a 60° C (-4° F a 140° F) 5% a 93% humedad relativa, no condensada

Regulación

Reúne o excede los siguientes requerimientos

Seguridad	UL 1950, CSA 950, TUV EN60950, Esquema CB(Desviación Nórdica)
RFI/EMI	FCC Clase A, DOC Clase A, EN55022 Clase A, VCCI Clase 1
Inmunidad	EN50082-1
Armónicas	EN61000-3-2

Dimensiones y Peso

Altura: 173 cm (68.3 in.)
Ancho: 77 cm (30 in.)
Profundidad: 99 cm (39 in.)
Peso, gabinete principal: 1 450 kg (1000 lb.) aproximadamente
Cable de corriente de 4.6 m (15 ft.)

Actualizaciones

Actualizaciones del sistema completo son válidas desde la SPARC-center 2000/2000E, SPARC-server 1000/1000E, SPARC-server 6X0,4X0, y sistemas 3X0.
Actualizaciones del chasis son válidas desde los servidores Enterprise 3000 y Enterprise 3500.

Apéndice B

Componentes de Solaris 2.6

Para las últimas noticias de Sun sobre actualizaciones y características de Solaris 2.6, vea la página Web Sun's New Solaris 2.6

Componentes:

- SunOS 5.6
- OpenWindows 3.6
- CDE 1.2

Releases:

- Release beta: Dic. 1996
- Release inicial: 18 de Agosto de 1997
- Nombre SunSolve Release: s297

Plataformas de hardware soportado:

- SPARC: sun4c, sun4m, sun4d, sun4u
- Intel 486, Pentium, Pentium Pro

Hardware soportado nuevamente:

- Dispositivos *SCSI-3*

Hardware antiguo no soportado:

- Intel 386 para PCs (no Sun 386i's -- No soportan cualquier Solaris)
- SPARCserver 630MP, 670MP, 690MP
- PowerPC
- Impresoras usando el software NeWSprint
- Dispositivos gráficos GS & GT
- Subsistemas de expansión SBus

Cambios respecto a Solaris 2.5.1:

- Nuevo software de impresión basado en el SunSoft Print Client de Solaris Migration CD, incluyendo soporte para configuración de impresión distribuida vía datos NIS o NIS+
- WebPrint - Administración de impresión basado en web y conectividad con Windows NT
- Manejo integrado de energía dentro el SO, soporte adicionado para x86
- Sincronización de archivos bajo máquinas móviles (Como las instalaciones breves de Win95)
- Mejorada la instalación/actualización del SO – puede reasignar espacio cuando se esta actualizando
- WebStart –Una Nueva herramienta de instalación para visualizador web basado en Java "fácil" instalación de software de múltiples CD's
- Remueve cliente dataless para mejorar el AutoClient
- Remueve Direct Xlib para mejorar la *X* Shared Memory (vea Early Notifier #13632 o 2.6 Release Notes)
- Características adicionales vold: tolerancia para remover dispositivos SCSI más que cdroms (por ejemplo Zip drives, unidades de cinta syquest, etc.)) (rfe/bug 1226629), Habilidad para accesar WABI & SunPC con drives vold-controlled (rfe/bug 1214434), comando volrmmount proporciona control al usuario sobre como montar un vold(rfe/bug 1214453)
- El visualizador del libro de respuestas tendrá la funcionalidad de un navegador Web adicionado para soportar el nuevo formato SGML – muchos de sus documentos tomados con el SO serán convertidos al nuevo formato, pero ambos formatos serán soportados por muchos productos que no están convertidos al momento de la actualización (Contrario a informes anteriores, las páginas man estarán en formato troff). Todos los libros de respuestas serán provistos en un CD.
- Máquina virtual Java 1.1.2 que incluye un navegador WWW HotJava
- Software cliente y servidor DHCP
- CDE 1.2 – incluido dentro SO-CD, dtlogin es el default en la consola en lugar del comando en línea login
- Dtmail de CDE soporta IMAP v4
- OpenWindows basado en X11R6(release original, no R6.1, R6.2 o R6.3)
- Soporta fuentes TrueType
- Mejor cliente NFS para tratar con fallas del servidor a través de clientes failover y mejoramiento de cachefs para permitir el uso de datos en cache cuando el servidor se apaga
- Adiciona WebNFS
- Adiciona Sun WebServer
- Mejoramiento de NIS+: backup/restore, mejor soporte sobre WANs, el nis_cachemgr hecho para multi-conexiones y proporciona funcionalidad como el ypbind para NIS+
- BIND DNS cliente/servidor actualizado para 4.9.5
- Interface del file system a 64 bits (archivos individuales >2 gigs) basado en el acuerdo API sobre el archivo Large File Summit, e incluido en la especificación única de UNIX, Versión 2 estándar (UNIX 98)
- Especificación UNIX, Versión 1 (aka SPEC 1170) conforme a – la división UNIX 95, parcial pero no completa.
- Especificación UNIX, Versión 2 (UNIX 98) conforme a libw y libintl unidas sobre libc, acompañada con las llamadas basename, dirname, regcmp, y regex de libgen (rfe/bug id 1258035)
- snprintf/vsnprintf anexado – actualmente definido por libc BSD 4.4/GNU, versiones futuras se completarán con la especificación UNIX 98 y el estándar revisado ISO C9x actualmente bajo desarrollo (rfe/bug id's 1220849,1266418)
- POSIX estándar adicional : soporta 1003.1 b en tiempo real, E/S asíncronas E/S.
- Soporta kerberos 5 cliente y servidor
- GSS-API (Generic Security Service Application Program Interface, RFC's 1508 & 1509) permitido por RSA y la funcionalidad Kerberos 5 en RPC
- Las puertas API ya son válidas para el público, tal como usadas en el sistema(semejante a un nuevo transporte el cual es pretendido para mostrar un gran desempeño extendido sobre el uso de la interfaz *looopback*)
- Keyserv es multi-enlazada y usa una puerta nueva de transporte RPC
- Nueva tarjeta de interface para servicios PCMCIA adicionada a DDI (ref/bug id 1254731), booteo de dispositivos soportados PCMCIA.

- x86: Nuevo sistema de booteo soportado por EISA y dispositivos Plug-and-Play, driver de instalación puede ser hecho por solo inserción de floppies con un nuevo driver (como Win95), compatibilidad binaria con SCO Unix
- nombre de dispositivos continuo de acuerdo a la reconfiguración inicial
- X/Open Federated Naming (XFN) 2.0
- Sincronización de reloj de red vía NTP (Network Time Protocol. rfc 1305l)
- velocidad general en una variedad de áreas, incluyendo mejoramiento para la optimización de las actualizaciones para los compiladores SPARC versión 4.2 mejor registro de localidades y algoritmos de predicción dividida
- UFS traído de Solstice DiskSuite 4.1 e integrado sobre el SO principal
- Soporte adicionado para la Variable Length Subnet Masking (VLSM)
- Mejoramiento de memoria virtual: El espacio total de direcciones es de 4 gig, tamaño de páginas >4 Kb
- MP mejorado: habilidad para unir un proceso a un conjunto de procesadores en vez de uno solo
- Mejoramiento de TCP/IP: Chequeo de hardware Zero-copy sobre ATM, tamaño máximo para escuchar las colas levantadas de 1024 (bug id 1265336)
- Los drivers SCSI (esp, isp, fas) permiten ahora establecer opciones a nivel por dispositivo
- "E/S directas": con la misma distancia entre UFS y particiones raw de bases de datos
- *Conforme al año 2000*: Todas las partes de SO probadas con la fecha del 2000 en adelante (Notificado pronto en #13631))

Apéndice C

Ejemplo de como crear una DLL

Los pasos para crear un DLL con Visual C++ son similares a los ejecutados para escribir un programa en C. Escribimos un archivo de encabezados .h, los archivos con la definición de las funciones (archivos .c o .cpp) el archivo de definiciones de módulos (archivos .def) y el archivo .mak para automatizar la construcción de la DLL.

Como ejemplo desarrollaremos un DLL llamada operac.dll que incluye las cuatro funciones denominadas suma, resta, multip y divis. Estas funciones tienen dos parámetros de tipo entero u regresarán un resultado también de tipo entero a excepción de la división que regresará un valor de tipo float.

Archivo de Encabezados (.h)

Los archivos de encabezado contienen declaraciones y definiciones que el preprocesador de C incluye en el archivo fuente justo antes de la compilación. Para este ejemplo, escribiremos un archivo *misOperac.h* que contiene una declaración de una variable hIsntDLL para almacenar el handle de la DLL por si fuera necesario, y las declaraciones de las funciones.

```
//-----------------------------------------------------------------------
// Nombre del archivo:MISOPERAC.H
//
// Este archivo de encabezado contiene las funciones
// prototipo para las funciones exportables por la DLL
// llamada misoperac
//-----------------------------------------------------------------------
```

```
// Variables globales
// Para almacenar el handle del ejemplar de la DLL

extern HANDLE hInstDLL


// Funciones prototipo
#ifdef _cplusplus // si C++ ..
  extern "C" {
#endif

int FAR PASCAL suma(int x, int y)
int FAR PASCAL resta( int x, int y)
float FAR PASCAL divis(int x, int y)
int FAR PASCAL multip(int x, int y)

#ifdef _cplusplus // si C++ ..

#endif
```

La palabra clave **FAR** especifica que la función puede ser llamada no solamente por las funciones que están en el mismo segmento de código, sino también por las funciones residentes en cualquier otro lugar de la memoria.

La palabra clave **PASCAL** dice al compilador use el convenio de compilación *pascal* . Todas la funciones que deban ser enlazadas desde fuera de la DLL deberán ser declaradas como **FAR PASCAL** o como **CALLBACK** .

Como se ha declarado los parámetros formales de l as funciones C esperan un valor cuando las funciones sean invocadas (no una dirección o referencia).

Archivo Fuente (.c o .cpp)

El archivo fuente contiene la definición de las funciones. Para nuestro ejemplo, escribiremos un archivo operac.c que contendrá los archivos de encabezado *windows.h* y *misoperac.h*, la función de entrada a la DLL *Libmain*, la función para retornar a Windows WEP (Windows Exit Procedure) y las funciones que deseamos incluir en la librería, suma,resta, multiplicación y división.

```
//----------------------------------------------------------------------------
// Nombre del archivo: operac.c
//
// Este es el archivo fuente principal de la DLL
// el punto de entrada de la DLL
//
//----------------------------------------------------------------------------

#include <windows.h>
#include "misoperac.h"

// Handle para almacenar un ejemplar de la DLL
HANDLE hInsDLL;

//----------------------------------------------------------------------------
// Función LibMain
//
// Esta función es llamada por Windows. Nadie la tiene que
// llamar desde las funciones
//
// Parámetros
//       hInst          - el handle para un ejemplar de la DLL
//       wDataSeg       - Apuntador al segmento de datos de la DLL
//       wHeapSize      - Tamaño de la memoria local de la DLL
//       loszCmdLine    - Línea de órdenes pasada a la DLL por Windows
//                        (raramente utilizado)
// Valor no regresado:
//       1 – indicando que la DLL se ha inicializado satisfactoriamente
//
//----------------------------------------------------------------------------

int PASCAL LibMain(HINSTANCE hInst, WORD wDataSeg, WORD wHeapSize, LPSTR loszCmdLine)
{
  if(wHeapSize!=0)
    UnlockData(0);              // Hacer que el segmento de datos de la DLL sea
  return(1);                    // movible (MOVEABLE)

}

//----------------------------------------------------------------------------
//
// Función WEP
//
// Llamada por Windows antes de que Windows descargue la DLL. Cada
// DLL: Cada DLL debe tener un WEP definida como está aquí. Visual C++
// incluye esta función automáticamente.
//
// Parámetros
// nParam – es una imitación de parámetros (raramente utilizado).
//
// Valor regresado
//  1 – Indicando que siempre podremos salir de la DLL.
//
//----------------------------------------------------------------------------
```

```
// Para implementar su propia WEP, añada la siguiente función a su
// aplicación (en el archivo .def no es necesaria una referencia a WEP).
// El código extern "C" sólo es necesario si se trata de un módulo C++
//
// extern "C" int FAR PASCAL _WEP(int Param)
{
        /* Escriba el código de su WEP aquí*/
  return 1
}

//---------------------------------------------------------------------------
//
// Funciónes
//
// Parámetros
//        x – valor entero
//        y – valor entero
//
// valor regresado
// valor entero: x+y
// valor entero: x-y
// valor entero: x*y
// valor flotante: x/y
//---------------------------------------------------------------------------

int PASCAL suma(int x, int y)
{
  int resul;
  resul=x+y;
  return(resul);
}

int PASCAL resta( int x, int y)
{

  int resul;
  resul=x-y;
  return(resul);
}

int PASCAL multip(int x, int y)
{
  int resul;
  resul=x*y;
  return(resul);
}

float PASCAL divis(int x, int y)
{
  int resul;
  resul=x/y;
  return(resul);
}
```

Archivo de definición de módulos (.def)

El archivo de definición de módulos informa al ligador (link) sobre como crear el archivo ejecutable. Para la DLL, puede ser el siguiente.

```
//-------------------------------------------------------------------
//
// Nombre del archivo: operac.def
// Módulos de definición del archivo
//-------------------------------------------------------------------

LIBRARY         operac
DESCRIPTION 'operaciones basicas DLL'
EXETYPE         WINDOWS
CODE            PRELOAD MOVEABLE DISCARDABLE
DATA            PRELOAD MOVEABLE
HEAPSIZE        4096;initial heap size
EXPORTS         suma
                resta
                multip
                divis
```

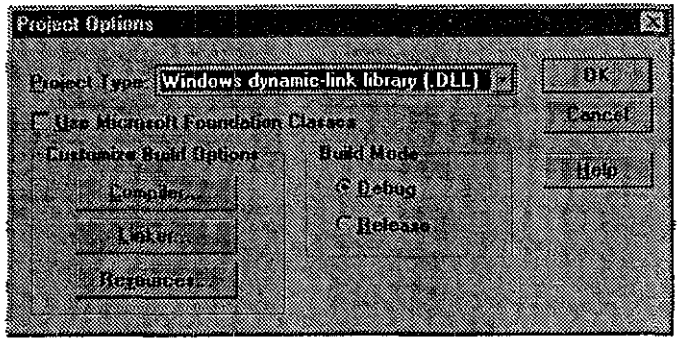
Donde **LIBRARY** especifica el nombre de la librería, **EXPORTS** define los nombres de los atributos de las funciones puestas a disposición de aplicaciones DLL; el valor ordinal a continuación del nombre de la función define la localización del nombre de la función en la tabla de nombres de la aplicación. **RESIDENTNAME** instruye para que el nombre de la función sea colocado en la tabla residente de nombres (por defecto se coloca en la tabla no residente).

Archivo .mak

Este archivo automatiza el proceso de construcción del programa. Este archivo no lo tenemos que escribir, ya que lo creamos utilizando el menú del compilador (en Visual C++ este menú es **Project**)

Compilación y enlace

Para compilar la DLL tiene que elegir el tipo de proyecto *Windows dynamic-link library (.DLL)*. Para ello, ejecute la orden **Project** del menú de **Options** del entorno de desarrollo VC++ y se visualizará la ventana de diálogo siguiente:



Ejecutado el proceso de compilación y enlace podemos observar que VC++ a generado dos archivos: operac.dll y operac.lib, donde operac.lib es la librería que informará al programa que utilice la librería y cuales son los puntos de entrada de las funciones que se encuentran en la DLL; esto es, sirve para realizar lo que denominaremos enlace estático.

Llamando a las funciones de la DLL

A continuación vamos a implementar una aplicación sencilla en .C que llame a las funciones DLL. Este programa contiene dos funciones, la primera llamada WinMain crea la ventana para el despliegue de las operaciones de la segunda función WinProc usando las funciones suma y resta creadas en una DLL.
//

```
//--------------------------------------------------------------------------
//
// Nombre del archivo: aplicac.c
//
//
//
//--------------------------------------------------------------------------
#include <windows.h>
#include <stdio.h>

long FAR PASCAL  WindowProc(HWND hWnd, UINT messg, WPARAM wParam,LPARAM lParam);
int FAR PASCAL suma(int x, int y); //Función Implementada como DLL
int FAR PASCAL resta(int x, int y); //Función Implementada como DLL

char szProgName[]="ProgName";
int PASCAL WinMain(HINSTANCE hInst,HINSTANCE hPreInst,LPSTR lpszCmdLine, int nCmdShow)
{
 HWND hWnd;
 MSG  lpMsg;
 WNDCLASS  wcApp;
 if(!hPreInst)
```

```
  {
   wcApp.lpszClassName=szProgName;
   wcApp.hInstance=hInst;
   wcApp.lpfnWndProc=WindowProc;
   wcApp.hCursor=LoadCursor(NULL,IDC_ARROW);
   wcApp.lpszMenuName=NULL;
   wcApp.hIcon=NULL;
   wcApp.hbrBackground=GetStockObject(WHITE_BRUSH);
   wcApp.style=CS_HREDRAW|CS_VREDRAW;
   wcApp.cbClsExtra=0;
   wcApp.cbWndExtra=0;
   if(!RegisterClass(&wcApp))
     return FALSE;
  }


  hWnd=CreateWindow(szProgName,"crea un DLL", WS_OVERLAPPEDWINDOW,CW_USEDEFAULT,

CW_USEDEFAULT,CW_USEDEFAULT,CW_USEDEFAULT,(HWND)NULL,(HWND)NULL,(HANDL
E)hInst,(LPSTR)NULL);
  ShowWindow(hWnd,nCmdShow);

  UpdateWindow(hWnd);
  while (GetMessage(&lpMsg,(HWND)NULL,0,0)){
     TranslateMessage(&lpMsg);
     DispatchMessage(&lpMsg);
  }

  return(lpMsg.wParam);

}

long FAR PASCAL  WindowProc(HWND hWnd, UINT messg, WPARAM wParam,LPARAM lParam)
{

     int resul1, resul2;

     resul1=suma(1,1);
     resul2=resta(10,6);

     printf("suma=%d resta=%d",resul1, resul2);

return (0L);

  }
```

Apéndice D

Puertos Estándares para la Conexión de red

Los cuales están tomados del archivo *services* que se encuentra en el directorio /etc del sistema Solaris Unix.

```
#ident "@(#)services 1.13 95/07/28 SMI" /* SVr4.0 1.8*/

#
# Network services, Internet style
#
tcpmux          1/tcp
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          timserver
time            37/udp          timserver
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         rpcbind
sunrpc          111/tcp         rpcbind
#
# 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
pop3            110/tcp         pop-3             # Post Office dia 8/11/96
uucp-path       117/tcp
nntp            119/tcp         usenet                  # Network News Transfer
ntp             123/tcp                         # Network Time Protocol
ntp             123/udp                         # Network Time Protocol
imap            143/tcp         imap              # Protocolo para correo
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
kerberos        750/udp          kdc            # Kerberos key server
kerberos        750/tcp         kdc             # Kerberos key server
ufsd            1008/tcp        ufsd            # UFS-aware server
ufsd            1008/udp        ufsd
ingreslock      1524/tcp
listen          2766/tcp                          # System V listener port
nfsd            2049/udp        nfs             # NFS server daemon (clts)
nfsd            2049/tcp        nfs             # NFS server daemon (cots)
lockd           4045/udp                        # NFS lock daemon/manager
lockd           4045/tcp
dtspc           6112/tcp                        # CDE subprocess control
fs              7100/tcp                        # Font server
```

BIBLIOGRAFIA

Solaris 2.2 System Configuration and Install Guide
SunSoft
A Sun Microsystems, Inc. Business

TCP/IP Network Administration
Craig Hunt
O'Reilly & Associates, Inc.

Tesis de Licenciatura
Solis Infante, Javier
Monitoreo y utilerías de administración de estaciones de trabajo
UNIX mediante una computadora personal trabajando en ambiente
Windows

Manual de Referencia
Windows Sockets
An Open Interface for Network Programming under Microsoft Windows.
Martin Hall, Mark Towfiq, Geoff Arnold, David Treadwell and Henry Senders.

Redes Locales y TCP/IP
Raya Cabrera, José Luis
Ra-Ma, 1995

Internetworking with TCP/IP Vol III, Client-server Programing
and Applications
Comer, Douglas E.
Prentice Hall, 1997

Aprendiendo TCP/IP en 14 días
Timothy Parker, Ph.D.
SAMS Publishing

TCP/IP Ilustrated Volume 2
The Implementation
Gary Wright, W. Richard Stevens
Addison-Wesley

Aprenda Visual C++
Mark Andrews
Microsoft Press

C The Complete Reference 3 Edition
Herbert Schildt
Mc Graw Hill Osborne

Visual C++
Victor Toth
Sams Publishing.

Inside Visual C++
Microsoft Press Programming Series
David J. Krugliskl.

Visual C++
Aplicaciones para Windows
Fco. Javier Ceballos
Ra-Ma

Libro de la SOUND BLASTER
José Luis Orós Cabello
Addison-Wesley Iberoamericana

Windows Sound Funpack
Tosca Moon Lee
SAM Publishing.

Introducción a los Ordenadores
J. Shelley
Alhambra

Internet
Tischer y Jennrich
Marcombo

Internet Interno Técnica y Programación
Tischer and Jennrich
Marcambo S.A. 1997

http://cslu.cse.ogi.edu/tts/research/index.htm Junio 1998
http://www.adi.uam.es/~adarraga/Lozano/SEVENWEB.HTML Junio 1998
http://www.alek.pucp.edu.pe/~dflores/matos Mayo 1998
http://www.alek.pucp.edu.pe/~dflores/matos/generaci.html Mayo 1998
http://132.248.44.94/lety.sistexp.html Junio 1998
http://liceu.uab.es/~joaquim/publicacions/llisterri_west_1987.htm Junio 1998
http://liceu.uab.es/~joaquim/publicacions/tutorial_sepln_88.html Junio 1998
http://www.sun.com/servers/enterprise/e5500/index.html Agosto 1998
http://www.sce.puc.cl/curso/_dist/cbc/textos/teoria/prospe1.html#quinta Agosto 1998
http://www.ocf.berkley.edu/solaris/versions/solaris/2.6.html Agosto 1998