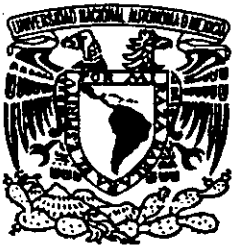

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

ESCUELA NACIONAL DE ESTUDIOS PROFESIONALES

CAMPUS ARAGÓN

"SISTEMA SUPERVISOR DE CLAVES Y ACCESOS EN UNIX (SCAU)"

T E S I S

QUE PARA OBTENER EL TÍTULO DE

INGENIERO EN COMPUTACIÓN

P R E S E N T A N :

CERVANTES SANTIAGO BERENICE

VELASCO SEGURA JAVIER

ASESOR :

ING. DONACIANO JIMENEZ VAZQUEZ

MEXICO. 2000

Agradecimientos:

A mis padres, por todo el apoyo que me han dado siempre.

A mis hermanos Rubí, Vero y Ever, por que siempre están cuando los necesito.

A ti Bere, por todo lo que significas para mi.

JAVIER

A las personas más importantes de mi vida mi madre por ser la luz en mi camino, mi padre por brindarme la inspiración de superarme y a dios por permitirme, lograr una meta más en mi vida.

Gracias

Berenice Cervantes Santiago

AGRADECIMIENTOS

A mis padres , Luisa y Alfonso por todo su amor , paciencia y apoyo.
A Dios, que me a cuidado, enseñado y nunca abandonado.
A Javier por su amor, cariño y comprensión que tuve.
A mi familia y amigos que comparten conmigo tantos recuerdos.
A mis asesores Javier de la Rosa y Donaciano Jiménez por su tiempo y comentarios para el término de esta tesis.
A las autoridades de DGSCA , que con su apoyo y facilidades prestadas han hecho posible la finalización de este proyecto.
A todos gracias.

Fue difícil pero teniendo fe y esperanza todo se logra.

AGRADECIMIENTOS

A mis padres , Luisa y Alfonso por todo su amor , paciencia y apoyo.
A Dios, que me a cuidado, enseñado y nunca abandonado.
A mis manos derechas, Jav y Sele.
A mi familia y amigos que comparten conmigo tantos recuerdos.
A mis asesores Javier de la Rosa y Donaciano Jiménez por su tiempo y comentarios para el término de esta tesis.
A las autoridades de DGSCA , que con su apoyo y facilidades prestadas han hecho posible la finalización de este proyecto.
A todos gracias.

Fue difícil pero teniendo fe y esperanza todo se logra.

INDICE GENERAL

INTRODUCCION VI

CAPITULO 1 ANTECEDENTES DE UNIX 1

1.1 HISTORIA DE UNIX 1
1.2. VERSIONES DE UNIX 3
1.3. CARACTERISTICAS DEL SISTEMA OPERATIVO UNIX . . 5
1.4. ESTRUCTURA DEL SISTEMA OPERATIVO UNIX 5
1.4.1. El kernel 6
1.4.2. El shell 7
1.5. MODELO DEL SISTEMA OPERATIVO UNIX 8
1.6. PERSPECTIVA DEL USUARIO 9
1.6.1. El sistema de archivos 9
1.6.2. El ambiente de procesamiento 11
1.6.3. Manejo de memoria 12
1.6.4. Manejo de entradas y salidas 13
1.7. MODELOS DE COMUNICACION EN UNIX 14
1.7.1. Antecedentes. Redes de computadoras 14
1.7.1.1. Tipos de redes 15
1.7.1.2. Protocolos 16
1.7.2. Modelo Cliente/Servidor 20
1.7.2.1. El servidor 21
1.7.2.2. El cliente 21
1.7.2.3. Ventajas e inconvenientes 21
1.7.3. Comunicación entre procesos 22
1.7.4. Sockets 22
1.7.5. Puertos 24

CAPITULO 2 ACCESANDO AL SISTEMA UNIX 25

2.1. EL INICIO DE SESION 25
2.2. EL CIFRADO DEL PASSWORD 27
2.2.1. Métodos de cifrado 27
2.3. LA ASIGNACION DE LA TERMINAL 29
2.4. LA SEGURIDAD DE LA CUENTA 30
2.4.1. Sistema de permisos 30
2.5. POLITICAS DE SEGURIDAD 32

CAPITULO 3 SISTEMA DE REGISTRO DE ACCESOS EN UNIX 33

3.1 REGISTRANDO LOS ACCESOS EN UNIX 33
3.2 ARCHIVO LASTLOG 33
3.3 ARCHIVOS UTMP Y WTMP 34
3.4 ARCHIVOS UTMPX Y WTMPX 37
3.5 ARCHIVO ACCT 38
3.6 COMANDOS UTILES EN EL MONITOREO DE USUARIOS . 39
3.6.1. Comando Finger 40
3.6.2. Comando Who 42
3.6.3. Comando W 43
3.6.4. Comando Last 43

CAPITULO 4 SISTEMA SUPERVISOR DE CLAVES Y ACCESOS EN UNIX (SCAU) 45

4.1 BREVE DESCRIPCION DEL SISTEMA SUPERVISOR DE CLAVES Y ACCESOS EN UNIX (SCAU) 45
4.2 ANTECEDENTES 45
4.2.1. Problemática 46
4.2.2. Alternativa de solución. 46
4.3 DESCRIPCION GENERAL 47
4.3.1 Servidor-SCAU 47
4.3.2 Cliente-SCAU 49
4.4 VENTAJAS DE SCAU 49

CAPITULO 5 DISEÑO E IMPLEMENTACIÓN DEL SCAU 51

5.1 DISEÑO ORIENTADO A FLUJO DE DATOS 51
5.1.1 Diagrama de flujo de datos 51
5.2 ELECCION DEL LENGUAJE DE PROGRAMACION 56
5.2.1 Características del lenguaje C 56
5.2.2 Características de Perl 57
5.3 MODULOS DE COMUNICACION SCAU 57
5.3.1 Módulo controlador del servidor 57
5.3.2 Módulos controladores del cliente 59
5.4 INTERFAZ GRAFICA. 60
5.4.1 PERL / TK. 60
5.4.2 Ayuda 61

5.5 INSTRUMENTACION DE SCAU. 62
5.5.1 Máquinas involucradas. 62
5.5.2 Sistemas operativos. 63
5.5.3 Configuración del servidor. 63
5.5.4 Configuración del cliente. 65

CONCLUSIONES. 67

APENDICE 69

1. PROGRAMA SERVIDOR DE SCAU: serv_SCAU.pl 69
2. PROGRAMA CLIENTE DE SCAU: ent_SCAU_cl.pl 70
3. PROGRAMA CLIENTE DE SCAU: lee_wtmp.c 72
4. PROGRAMA CLIENTE DE SCAU: sal_SCAU_cl.pl 74
5. PROGRAMA SCAU: crea_interfaz.pl 76
6. PROGRAMA SCAU: ayuda.pl 110
7. PROGRAMA SCAU: ayuda2.pl 114

REFERENCIAS BIBLIOGRAFICAS. 116

INDICE DE FIGURAS

1.1 Versiones de UNIX 4
1.2 Ejemplo del sistema de archivos en UNIX 6
1.3 Estructura jerárquica de los componentes de UNIX 10
1.4 Estructura del modelo OSI 17
1.5 Relación del modelo TCP/IP con el modelo OSI 18
1.6 Modelo de capas de TCP/IP en relación con OSI 20

4.1 Solicitud del cliente al servidor 47
4.2 Atención a más de un cliente al mismo tiempo 48
4.3 Cuarta generación de interfaces gráficas 49

5.1 Diagrama de Flujo de Datos Nivel 0. 52
5.2 Diagrama de Flujo de Datos Nivel 1. 53
5.3 Diagrama de Flujo de Datos Nivel 2. 54
5.4 Diagrama de Flujo de Datos Nivel 3. 55
5.5 Forks del servidor. 58
5.6 Forks del cliente. 59
5.7 Interfaz del sistema SCAU. 61
5.8 Ventana del Ayuda de SCAU. 62

INDICE DE TABLAS

Tabla 3.1: Archivos de registro de los procesos 38
Tabla 3.2: Comandos para habilitar y deshabilitar la contabilidad. 39

INTRODUCCION

A través del tiempo las computadoras se han convertido en una herramienta principal de casi cualquier rama de la ciencia, de la tecnología y de casi todas las labores cotidianas del hombre, donde investigadores, compañías y gobiernos centran su trabajo.

En los distintos centros de cómputo existe una o varias personas que se encargan de la administración del sistema. Las funciones básicas que tiene un administrador son dos: administrar el sistema y cuidar la seguridad del mismo. Estas tareas se complican cada vez más conforme aumenta el tamaño del sistema , el número de usuarios o la complejidad del mismo.

En particular, UNIX no fue diseñado en un principio con la idea de ser un sistema totalmente seguro, pero cuenta con los elementos necesarios para hacerlo seguro. Esta seguridad del sistema se relaciona con los usuarios del mismo y la información contenida dentro de él. Al dejar que la información de cualquiera de los usuarios la conozca otra gente puede ser la causa de muchos problemas como pérdidas de grandes cantidades de dinero, retrasos en proyectos, la reinstalación del sistema operativo inclusive pérdidas de vidas humanas entre otras cosas. Es por eso que se debe tener un control de las claves existentes (cuentas), así como de los accesos de los usuarios al sistema y sus conexiones a otros lugares, para poder tomar precauciones antes de tener algún problema de seguridad y si el número de máquinas aumenta el control es mas difícil.

Como una solución a lo anterior se creó el "Sistema Supervisor de Claves y Accesos en UNIX (SCAU)" que permite supervisar los accesos, salidas de los usuarios y tener una constante revisión del estado de las cuentas en el sistema. Los resultados obtenidos del sistema son representados en forma gráfica remplazando la tarea de analizar la información en archivos de texto. Además, el sistema permite monitorear varias computadoras al mismo tiempo, se creo una comunicación entre ellas que solo se dedicara exclusivamente a atender solicitudes de los registros de accesos de las máquinas involucradas. La comunicación esta basada en el modelo cliente/servidor conteniendo además una interfaz gráfica en Perl/TK, que facilita al administrador el uso de este sistema.

Este sistema ayudará a supervisar los accesos y a la revisión del estado de las cuentas de cada una de las máquinas del Laboratorio de Visualización y del Departamento de Supercómputo de la Dirección Ceneral de Servicios de Cómputo Académico (DGSCA) de la Universidad Nacional Autónoma de México (UNAM).

OBJETIVOS

El objetivo de la tesis es diseñar un programa que permita supervisar de una forma automatizada los accesos, salidas y las claves de los usuarios.

Los objetivos específicos son:

- Terminar SCAU en su primera versión.
- Instalarlo y usarlo como parte de las herramientas desarrolladas en el Departamento de Administración de la DGSCA, UNAM.

HIPOTESIS

Si se instala SCAU en las máquinas del Departamento de Supercómputo y Laboratorio de Visualización, entonces se incrementará el control de los accesos, salidas del sistema y la revisión del estado de las claves de los usuarios, con la ventaja de interpretar de una manera más rápida el conjunto de datos proporcionados por los resultados generados del sistema.

PROLOGO

La tesis consta de cinco capítulos y comienza con la introducción, donde se da una breve razón de por qué es importante contar con herramientas que ayuden a la administración de los sistemas. En el capítulo I se describirá la historia del sistema operativo UNIX, su estructura, así como las características generales más importantes de este sistema que lo hacen portable a una gran variedad de máquinas que van desde computadoras personales hasta supercomputadoras, además se hablará del tipo de comunicación entre las máquinas con sistema UNIX en específico del modelo cliente/servidor. En el capítulo II se describe el inicio de sesión en el sistema UNIX y las políticas de seguridad. En el capítulo III se describe el sistema de registro de accesos en UNIX y algunos comandos que nos ayudan al monitoreo de los usuarios. En el capítulo IV, se describirán las necesidades que cubrirá este sistema además de sus ventajas como un sistema que pueda proporcionar información de los accesos, salidas y el estado de las claves de varias máquinas a través de la red. En al capítulo V se describe el diseño y la implementación de SCAU, utilizando las herramientas de programación adecuadas, como los lenguajes PERL/TK y C que nos proporcionan librerías y funciones, para crear los programas e interfaces gráficas adecuadas en las estaciones de trabajo, la implementación y uso del sistema se describirán es este capitulo, como una parte fundamental del desarrollo del trabajo para tener un punto de vista objetivo sobre las conclusiones generales del mismo.

Este trabajo esta dirigido preferentemente para aquellas personas que tienen un conocimiento de computación con nivel de estudios superiores y para aquellas personas que deseen introducirse a la supervisión de las claves y accesos en los sistemas UNIX.

CAPITULO 1

ANTECEDENTES DEL SISTEMA OPERATIVO UNIX

Este capítulo menciona algunas características importantes del sistema operativo UNIX, que ponen de manifiesto el ambiente de trabajo de este sistema. Además se hablará del tipo de comunicación entre las máquinas con sistema UNIX en específico del modelo cliente/servidor.

1.1 HISTORIA DE UNIX

El sistema operativo UNIX se considera uno de los mayores avances en el progreso de las computadoras, desde las actividades diarias de las personas hasta los grandes procesos de cómputo intensivo ha destacado por su facilidad para hacer algunas tareas difíciles de realizar en otros sistemas operativos.

Como cualquier otro sistema operativo, UNIX permite ejecutar programas, proporciona una útil y conveniente interfaz para la amplia variedad de dispositivos periféricos(impresoras, discos, terminales, etc.) que se encuentran conectados a la mayoría de cualquier computadora, y facilita un sistema de archivos para el almacenamiento de la información.

UNIX nació en los laboratorios Bell de la prestigiada división de investigación de AT&T (American Telephone and Telegraph Company). En los años 60's las compañías AT&T, Honeywell, General Electric y MIT inician un proyecto de información llamado Multics, éste era un sistema interactivo multiusuario que empleaba una gran computadora General Electric dentro de los laboratorios. En 1969, los laboratorios Bell deciden abandonar el proyecto Multics, pero Ken Thompson y Dennis Ritchie de esta compañía continuaron desarrollando algunas ideas. El nombre de UNIX es un juego de palabras sobre la palabra Multics, existe una diferencia notable entre estos dos sistemas: el sistema operativo UNIX es relativamente simple en comparación con Multics que era extremadamente complejo.

Ken Thompson que aparentemente pretendía crear un sistema operativo que pudiera mantener los esfuerzos coordinados de un equipo de programadores en un entorno de investigación de programación más barato y manejable, fue quien comenzó a desarrollar el sistema UNIX. Éste trabajo tuvo suficiente éxito como para atraer la atención de Dennis Ritchie y otros técnicos de los laboratorios Bell que continuaron el proceso de creación de un entorno útil. Una primera versión del sistema fue entregada a la oficina de patentes de Bell Labs en 1971.

Al principio de los años 70, UNIX funcionaba únicamente en las computadoras fabricadas por Digital Equipment y estaba escrito en ensamblador[1], primero sobre la computadora PDP-7, a continuación en el PDP-11/40, /45 y por último en el que sería su consagración obteniendo una amplia aceptación en los laboratorios: el PDP-11/70. Ken Thompson decidió reescribir el sistema operativo en un lenguaje de nivel mayor que denominó lenguaje B. En 1973, Dennis Ritchie escribe el lenguaje de programación C basado en el lenguaje B de Thompson y reescribió UNIX.

[1] Lenguaje de bajo nivel

El lenguaje de programación C hizo que UNIX sea fácilmente transportable a otros sistemas informáticos (solo una muy pequeña parte de UNIX está escrita en lenguaje ensamblador). Al estar escrito en lenguaje C hace que el sistema sea adaptable y fácil hacer modificaciones, esto hace que UNIX siga evolucionando de acuerdo a las necesidades de los diferentes usuarios, no solo de los usuarios que se dedican a hacer software sofisticado, sino que también permite a los usuarios esporádicos modificar las órdenes del sistema.

La transportación de UNIX hacia otros sistemas se hizo posible, el primer paso hacia un tipo diferente de computadora fue realizado por Ritchie y Stephen Johnson en 1976 y UNIX fue probado en un Interdata 8/32. Desde entonces se ha trasladado prácticamente a cualquier máquina, que van desde las de un solo microprocesador hasta las máquinas trabajando en paralelo con varios procesadores.

A medida que UNIX ganaba la aceptación de los programadores, durante el principio de los 70, comenzó a ser usado en todas las áreas de la organización Bell. UNIX extendió su fama y ganó el interés de varias y prestigiosas instituciones académicas. Los derechos eran nominales para las instituciones académicas, animándolas a usar y posteriormente desarrollar el sistema. De éstas, fue la Universidad de Berkeley California la que más ha participado en el desarrollo de novedosas aportaciones, incluso creo su propia versión de UNIX que denominó BSD (Berkeley Software Distribution).

UNIX ha sido pionero en algunas ideas importantes. Una de las innovaciones más importantes del sistema es la tubería (*pipe*)[1], con ésta, se ha logrado que algunas complicadas funciones puedan estar como un conjunto de programas trabajando juntos. Las conexiones en tubería permiten usar tantos programas como sean necesarios. Otra de las ideas que se ha desarrollado en UNIX más que en otros sistemas es la noción de herramienta de software. Muchas de estas herramientas son pequeños programas coherentes que realizan bien una sola tarea y que pueden cooperar con otras herramientas para ejecutar tareas más complicadas.

En la actualidad, el resultado neto de los avances en tecnología sobre hardware es que UNIX se puede ejecutar ahora en casi cualquier sistema informático basados en microprocesadores. Se puede utilizar UNIX en cualquier tipo de computadora, desde las personales hasta las grandes supercomputadoras.

[1] Es una conexión directa que toma la salida de un proceso y la envía como entrada de otro sin necesidad de utilizar un archivo temporal como intermediario.

1.2 VERSIONES DE UNIX

En 1974, Thompson y Ritchie publicaron un informe para la ACM (Association for Computing Machinery) llamado "El sistema UNIX de tiempo compartido"). Esta publicación tomó un amplio interés por el sistema cuando se enteraron de que la versión que se describía en el informe, podría ser adquirida por un mínimo costo y con el código fuente completo.

Muchas de las universidades del mundo recibieron en 1976 la versión 6 del sistema UNIX. La versión 6 fue la base para que se desarrollaran las diferentes variantes del UNIX. Esta versión poseía un *shell*[1] primitivo y un conjunto de casi cien utilidades. Las características de programación del shell de la versión 6 eran rudimentarias pero obtuvo la distinción de ser el primer sistema UNIX que se llevó a una firma comercial, cuando Whitesmiths Inc produjo un sistema parecido a la versión 6 llamado IDRIS.

La versión 7 del sistema UNIX sale al mercado por parte de laboratorios Bell en 1978. Esta versión es importante por varias razones: contiene la primera aparición del Bourne shell (llamado así por el apellido del autor Steve Bourne), el primer shell que combina un potente lenguaje de programación con prestaciones para la entrada de comandos interactivos y fue la base para el sistema UNIX 32V de la computadora VAX de Digital Equipment.

Existen cuatro fuerzas mayores y muchas pequeñas empresas en el campo UNIX, las más importantes son: AT&T, la Universidad de Berkeley en California, Sun Microsystems y Microsoft Corporation. El interés de AT&T en UNIX es obvio, y ahora que se permite competir en el campo del proceso de datos, su mayor fuerza está en el sistema UNIX, la Universidad de Berkeley tiene en varias áreas la versión de UNIX más avanzada técnicamente, Sun Microsystems fábrica estaciones de trabajo que funcionana con este sistema operativo y son responsables de algunos grandes avances técnicos como la primera estación de trabajo sin discos y el sistema de archivos en red NFS, Microsoft Corporation es el responsable de XENIX, una de las versiones comercial más popular de UNIX.

En UNIX existen dos variantes mayores: system V y BSD (Berkeley Software Distribution), más algunas docenas de variaciones menores de estas dos.

UNIX System V

La compañía AT&T anunció en enero de 1983 que por primera vez iba a preparar una versión estándar del UNIX para el mercado comercial OEM (compradores de equipos que a su vez son fabricantes). Esta versión recibe el nombre de System V, y está basado en el UNIX utilizado internamente por AT&T. Cuando se lanza la versión 4.2 de System V, esta cuenta ya con una interfaz gráfica con el usuario o GUI., con esto, el sistema V se convierte en un sistema más amigable.

[1] Interprete de comandos en linea

UNIX BSD

La Universidad de Berkeley California libera por primera vez la versión de UNIX BSD (Berkeley Software Distribution), basada en la versión 7 de UNIX de AT&T, como es conocido por toda la industria, contiene mejoras desarrolladas por la comunidad académica que Berkeley designó para hacer de UNIX un sistema más amigable.

Las mejoras amigables para el usuario en BDS UNIX eran un esfuerzo por hacer UNIX algo en común, gente ordinaria así como programadores avanzados mezclaron sus demandas conformando un sistema flexible para todos. A pesar de ser menos del 100 por ciento compatible con el original UNIX de AT&T, BSD logró sus metas; los rasgos agregados incitaron a los usuarios comunes para usar UNIX. BSD se ha vuelto un estándar académico.

En la actualidad existen muchas versiones de UNIX, fundamentan sus bases ya sea en el Sistema V, en BSD o una combinación de los dos (híbridos). A continuación se muestra un diagrama donde se ven algunos sistemas y sus orígenes (figura 1.1).

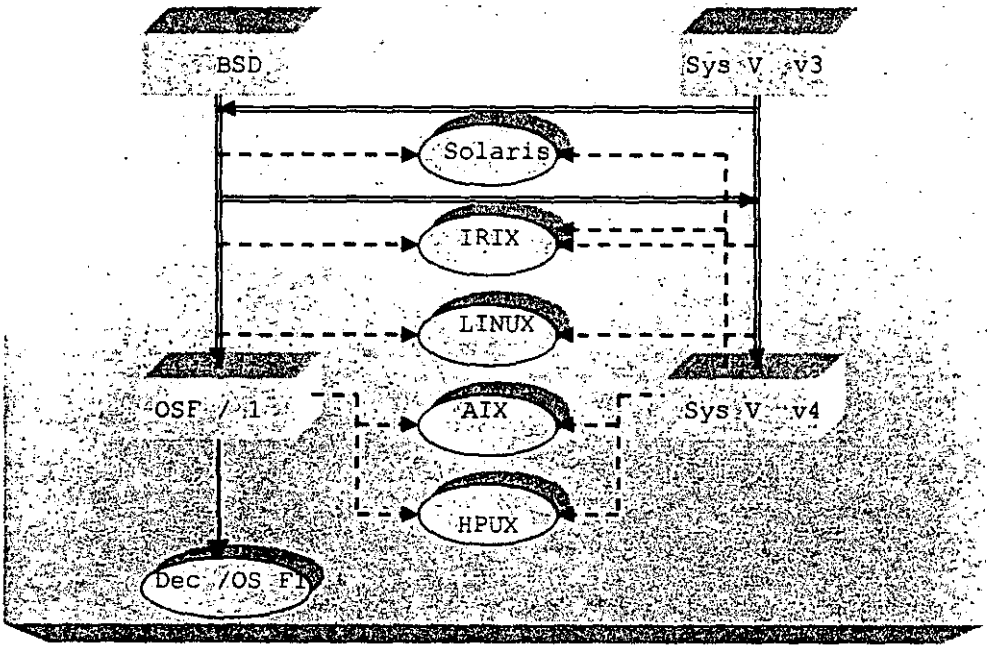

Figura 1.1 Versiones de UNIX

1.3 CARACTERISTICAS DEL SISTEMA OPERATIVO UNIX

El sistema operativo UNIX es un programa de control para computadoras, pero también es toda una familia de programas de utilidad bien diseñados y un conjunto de instrumentos que permiten al usuario conectar y utilizar esas utilidades para construir sistemas y aplicaciones.

UNIX posee las siguientes características singulares:

- Es un sistema operativo *multiusuario*[1], además de poder trabajar con varios procesadores.
- Está escrito en un lenguaje de alto nivel, el cual no depende de la arquitectura de la máquina (lenguaje C).
- Dispone de un lenguaje de control programable (shell).
- Ofrece facilidades para la creación de programas y sistemas y el ambiente adecuado para las tareas de diseños de software.
- Tiene capacidad de interconexión de *procesos*[2].
- Emplea un sistema jerárquico de archivos, con facilidades de protección de archivos, cuentas y procesos.
- Tiene facilidad para redireccionamiento de Entradas/Salidas (E/S).

1.4 ESTRUCTURA DEL SISTEMA OPERATIVO UNIX

Se puede dividir en varios componentes perfectamente diferenciados:

- Núcleo o Kernel: Comprende un 5-10% del código total.
- Caparazón o Shell: Actúa como intérprete de comandos.
- Programas de utilidad.

Los diversos componentes del sistema operativo pueden verse gráficamente de la manera mostrada en la figura 1. 2.

[1] Permite que varios usuarios trabajen al mismo tiempo
[2] Son instancias de un programa en ejecución

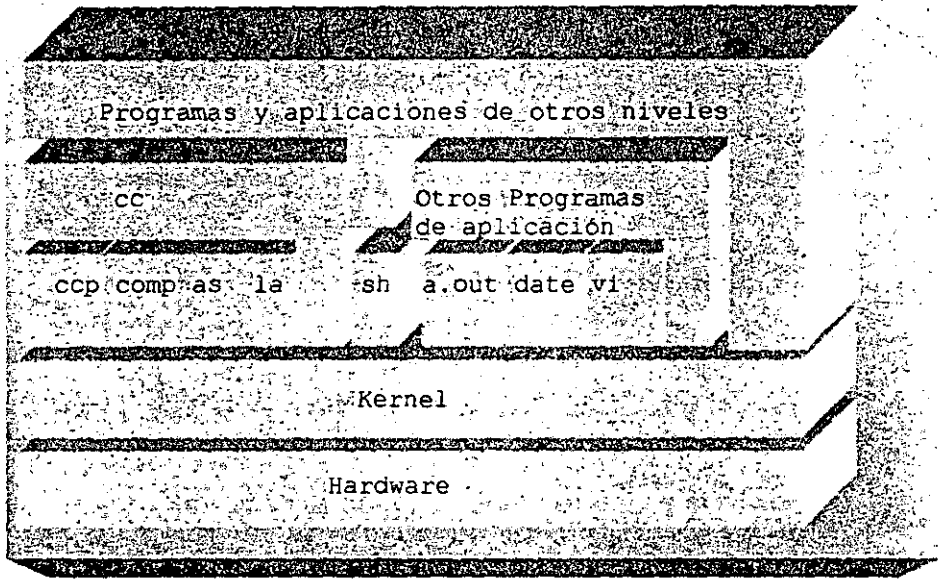

Figura 1.2 Estructura jerárquica de los componentes de UNIX

1.4.1 EL KERNEL

El núcleo del sistema operativo UNIX (*Kernel*) es un programa escrito casi en su totalidad en lenguaje C, con excepción de una parte del manejo de interrupciones, expresada en el lenguaje ensamblador del procesador en el que opera.

Las funciones del núcleo son permitir la existencia de un ambiente en el que sea posible atender a varios usuarios y múltiples tareas en forma conjunta, repartiendo al procesador entre todos ellos, e intentando mantener en grado óptimo la atención individual.

El Kernel opera como asignador de recursos para cualquier proceso que necesite hacer uso de las facilidades de cómputo. Es el componente central de UNIX y tiene las siguientes funciones:

- Creación de procesos, asignación de tiempos de atención y sincronización.
- Asignación de la atención del procesador a los procesos que lo requieren.
- Administración de espacio en el sistema de archivos, que incluye: acceso, protección y administración de usuarios; comunicación entre usuarios y entre procesos, manipulación de E/S y administración de periféricos.
- Supervisión de la transmisión de datos entre la memoria principal y los dispositivos periféricos.

El Kernel reside siempre en la memoria central y tiene el control sobre la computadora, por lo que ningún otro proceso puede interrumpirlo; sólo pueden llamarlo para que proporcione algún servicio de los ya mencionados. Un proceso llama al Kernel mediante módulos especiales conocidos como *llamadas al sistema*[1].

El Kernel consta de dos subsistemas principales: la sección de control de procesos y la de control de dispositivos. La primera asigna recursos, programas, procesos y atiende sus requerimientos de servicio; la segunda, supervisa la transferencia de datos entre la memoria principal y los dispositivos periféricos. En términos generales, cada vez que algún usuario oprime una tecla de una terminal, o que se debe leer o escribir información del disco magnético, se interrumpe al procesador central y el núcleo se encarga de efectuar la operación de transferencia.

Un Kernel típico puede constar de unas 20,000 líneas de código de las cuales un 70-80% está escrito en C y el resto depende de máquina.

1.4.2 EL SHELL

La comunicación con el sistema UNIX se da mediante el programa de control llamado shell. Este es un lenguaje de control, un intérprete y un lenguaje de programación, cuyas características lo hacen sumamente flexible para las tareas de un centro de cómputo. Como lenguaje de programación abarca los siguientes aspectos:

- Ofrece las estructuras de control normales: secuenciación, iteración condicional, selección y otras.
- Paso de parámetros.
- Sustitución textual de variables y cadenas.
- Comunicación bidireccional entre las órdenes.

El shell permite modificar en forma dinámica las características con que se ejecutan los programas en UNIX:

- El sistema de E/S puede ser redireccionado o redirigido hacia archivos, procesos y dispositivos.
- Es posible interconectar procesos entre sí.
- Diferentes usuarios pueden "ver" versiones distintas del sistema operativo debido a la capacidad del shell para configurar diversos ambientes de ejecución.

Desde el punto de vista del usuario, el shell actúa como un intérprete de comandos. Es un programa que siempre está en ejecución. El shell lee las órdenes suministradas, las decodifica y lo comunica al núcleo para realizar la acción especificada.

[1] Comunicaciones directas con el Kernel

Prácticamente, todas las órdenes son programas ejecutables que el shell busca en el sistema de archivos, siguiendo el orden especificado en la variable global *PATH*[1].

Existen varios tipos de shells que dependen principalmente de la versión de UNIX utilizada:

- Bourne shell (System V, Xenix).
- C shell (Berkeley).
- Korn shell (Ambos).

1.5 MODELO DEL SISTEMA OPERATIVO UNIX

Los principales componentes del Kernel son el subsistema de archivos y el subsistema de control de procesos. Por lo que los dos conceptos importantes en el modelo del sistema operativo UNIX son: los archivos y los procesos.

El subsistema de archivos se entiende como aquella parte del sistema responsable de la administración de los datos en dispositivos de almacenamiento secundario (discos duros). El subsistema de archivos accesa a los periféricos usando un mecanismo llamado *buffering*[2]. Este mecanismo regula el flujo de los datos entre el Kernel y dispositivos secundarios de almacenamiento. Dicho mecanismo interactua con los manejadores de dispositivos de E/S de bloques, para inicializar transferencias de datos con el Kernel.

El subsistema de control de procesos es responsable de sincronizar los procesos, de la comunicación entre procesos del manejo de memoria y el tiempo que asigna el *scheduler* (despachador) a cada proceso. El módulo scheduler distribuye el tiempo de CPU para todos los procesos, los cuales se ejecutan uno a la vez, hasta que dejan de consumir CPU mientras esperan un recurso, o cuando exceden el tiempo determinado para ejecutarse. El scheduler elige otro proceso que tenga la prioridad mayor entre los procesos que están listos para ejecutarse, el proceso interrumpido se ejecutará nuevamente cuando este disponible para hacerlo y su prioridad lo permita.

El módulo de memoria maneja y controla la distribución de la memoria, si por alguna razón la memoria física no es suficiente para todos los procesos, el Kernel intercambia estos entre la memoria principal (memoria RAM) y la secundaria (discos duros), con la finalidad de que todos los procesos tengan el tiempo razonable para ejecutarse. Este tipo de memoria se le conoce como memoria *swap*.

[1] Variable de ambiente que contiene las rutas directas de programas ejecutables
[2] Almacenamiento momentáneo

1.6 PERSPECTIVA DEL USUARIO

1.6.1 EL SISTEMA DE ARCHIVOS

Una de las funciones más importantes de un sistema operativo es proporcionar un sistema de archivos para manejo de información.

Entre las características más relevantes del sistema de archivos de UNIX podemos citar los siguientes:

- Los usuarios tienen la posibilidad de crear, modificar y borrar archivos.
- Cada archivo tiene tres tipos de acceso diferentes: acceso de lectura [r], acceso de escritura [w] y acceso de ejecución [x]. A su vez, esos tres tipos de acceso pueden extenderse a la persona propietaria del archivo, al grupo al cual está adscrita dicha persona y el resto de los usuarios del sistema. Eso permite que los archivos puedan ser compartidos de forma controlada.
- Cada usuario puede estructurar sus archivos como desee, el núcleo de UNIX no impone ninguna restricción, solo el uso correcto de persmisos evita problemas.
- UNIX también proporciona varias utilerías para hacer copias de seguridad de todos y cada uno de los archivos para prevenir la pérdida de forma accidental o maliciosa de la información.
- Proporciona la posibilidad de cifrado y descifrado de información. Eso se puede hacer para que los datos solo sean útiles para las personas que conozcan la clave de descifrado.
- El usuario tiene una visión lógica de los datos, es el sistema el encargado de manipular correctamente los dispositivos y darle el soporte físico deseado a la información. El usuario no tiene que preocuparse por los dispositivos físicos, es el sistema el que se preocupa de la forma que toman los datos en los dispositivos y de los medios físicos de transferencia de datos desde y hacia los dispositivos.

El sistema de archivos mantiene una organización conocida con el nombre estructura en árbol invertido (figura 1.6).

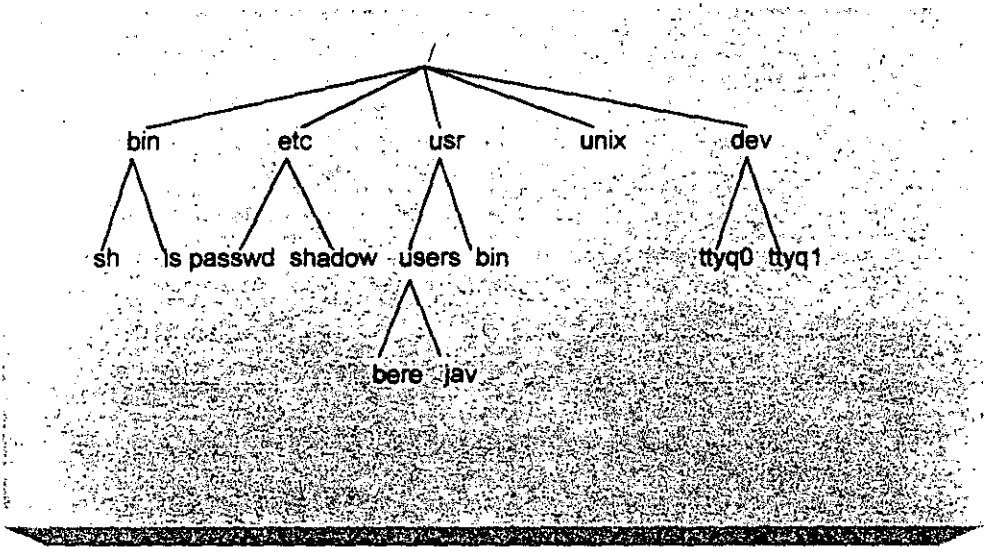

Figura 1.6 Ejemplo del sistema de archivos en UNIX

El árbol contiene un nodo llamado root (/) que es el directorio inicial y apartir de éste se encuentran otros nodos que pueden ser tanto archivos como directorios y estos últimos pueden contener a su vez subdirectorios y archivos.

Existen tres tipos de archivos en UNIX:

Ordinarios. Son cadenas de bytes terminadas con <ctrl>D (este código significa fin de archivo). Pueden ser texto, objetos, ejecutables, librerías de módulos, etc.

Directorios. Contienen nombres de archivos y su dirección física. Puede pensarse en ellos como carpetas que contienen archivos y directorios. Un directorio dentro de otro directorio se denomina subdirectorio.

Especiales. Asociados a dispositivos E/S. Contienen referencias a los *drivers* (programas que manejan directamente los dispositivos y que forman parte del núcleo), pueden ser de tipo *bloque* (apuntan a dispositivos tipo disco) y *carácter* (apuntan a dispositivos como terminales, impresoras, etc). Por convenio, residen en el directorio /dev.

Cada archivo en el sistema UNIX tiene un *inodo* único, que básicamente es un registro por medio del cual el sistema de archivos identifica a todos los archivos existentes. El inodo contiene la información necesaria para que un proceso pueda accesar al archivo, permisos de acceso, tamaño del archivo, y la localización de los datos en el sistema de archivos. Los procesos accesan a los archivos por un grupo bien definido de llamadas al sistema. El nombre de un archivo está dado por una dirección llamada trayectoria, ésta describe en donde se encuentra localizado el archivo dentro de la jerarquía del sistema de archivos. Una trayectoria es una secuencia de nombres separados por un carácter slash (/) . Cada nombre únicamente especifica a un archivo, y el kernel convierte el nombre de la trayectoria en el inodo del archivo.

El sistema operativo UNIX accesa a los periféricos de igual forma como lo hace con los archivos regulares. Esto es debido a que los nombres de los periféricos y los nombres de los archivos regulares son similares además las condiciones de trabajo también son similares, muchos programas no tienen por que conocer internamente el tipo de archivo a periférico que ellos manipulan.

1.6.2 AMBIENTE DE PROCESAMIENTO

En UNIX se ejecutan programas en un medio llamado "proceso de usuario". Cuando se requiere una función del Kernel, el proceso de usuario hace una llamada especial al sistema y entonces el control pasa temporalmente al núcleo. Para esto se requiere de un conjunto de elementos de uso interno, que se mencionan a continuación.

Se conoce como *imagen* a una especie de fotografía del ambiente de ejecución de un proceso, que incluye una descripción de la memoria, valores de registros generales, estado de archivos abiertos, el directorio actual, etc. Una imagen es el estado actual de una computadora virtual, dedicada a un proceso en particular.

Un proceso se define como la ejecución de una imagen, mientras el procesador ejecuta un proceso, la imagen debe residir en la memoria principal; durante la ejecución de otros procesos permanece primero en la memoria principal a menos que la aparición de un proceso activo de mayor prioridad la obligue a ser copiada al disco como ya se dijo.

Un proceso puede encontrarse en uno de varios estados: en ejecución; listo para ejecutar, o en espera.

Cuando se invoca una función del sistema, el proceso de usuario llama al Kernel como subrutina. Hay un cambio de ambientes y, como resultado, se tiene un proceso del sistema. Estos dos procesos son dos fases del mismo original, que nunca se ejecutan en forma simultánea. Existe una tabla de procesos que contiene una entrada por cada uno de ellos con los datos que requiere el sistema: identificación, direcciones de los segmentos que emplea en la memoria, información que necesita el scheduler y otros. la entrada de la tabla de procesos se asigna cuando se crea el proceso y se libera cuando éste termina.

Entre las diferentes llamadas al sistema para el manejo de procesos que existen en UNIX están las siguientes, algunas de las cuales ya han sido mencionadas: *fork* (crear una copia a un proceso); *exec* (cambiar la identidad de un proceso); *kill* (enviar una señal a un proceso); *signal* (especificar la acción por ejecutar cuando se recibe una señal de otro proceso), y *exit* (terminar un proceso).

Dentro de las tareas del manejo del procesador destaca la asignación dinámica (scheduling), que en UNIX resuelve el scheduler mediante un mecanismo de prioridades. Cada proceso tiene asignada una prioridad; las prioridades de los procesos de usuario son menores que la más pequeña de un proceso del sistema.

El "motor" que mantiene en movimiento un esquema de multiprogramación es, por un lado, el conjunto de interrupciones que genera el desempeño de los procesos y, por otro, los constantes recordatorios que hace el reloj del procesador para indicar que se terminó la fracción de tiempo dedicada a cada proceso.

En el sistema UNIX, las interrupciones son causadas por lo que se conoce como eventos, entre los cuales se consideran: la ejecución de una tarea de E/S; la terminación de los procesos dependientes de otro; la terminación de la fracción de tiempo asignada a un proceso, y la recepción de una señal desde otro proceso.

En un sistema de tiempo compartido se divide el tiempo en un determinado número de intervalos o fracciones y se asigna cada una de ellas a un proceso. Además UNIX toma en consideración que hay procesos en espera de una operación de E/S y que ya no pueden aprovechar su fracción. Para asegurar una distribución adecuada del procesador entre los procesos se calculan dinámicamente las prioridades de estos últimos, con el fin de determinar cuál será el proceso que se ejecutará cuando se suspenda el proceso activo actual.

1.6.3 MANEJO DE MEMORIA

Dependiendo de la computadora en la que se ejecute, UNIX utiliza dos técnicas de manejo de memoria: swapping y memoria virtual.

Lo estándar en UNIX es un sistema de intercambio de segmentos de un proceso entre memoria principal y memoria secundaria, llamado swapping lo que significa que se debe mover la imagen de un proceso al disco si éste excede la capacidad de la memoria principal, y copiar el proceso completo a memoria secundaria. Es decir, durante su ejecución, los procesos son cambiados de y hacia memoria secundaria conforme se requiera.

Si un proceso necesita crecer, pide más memoria al sistema operativo y se le da una nueva sección, lo suficientemente grande para acomodarlo. Entonces, se copia el contenido de la sección usada al área nueva, se libera la sección antigua y se actualizan las tablas de descriptores de procesos. Si no hay suficiente memoria en el momento de la expansión, el proceso se bloquea temporalmente y se le asigna espacio en memoria secundaria. Se copia a disco y, posteriormente, cuando se tiene el espacio adecuado -lo cual sucede normalmente en algunos segundos- se devuelve a memoria principal.

Está claro que el proceso que se encarga de los intercambios entre memoria y disco (llamado swapper) debe ser especial y jamás podrá perder su posición privilegiada en la memoria central. El Kernel se encarga de que nadie intente siquiera interrumpir este proceso, del cual dependen todos los demás. Este es el proceso 0 mencionado antes. Cuando se decide traer a la memoria principal un proceso en estado de "listo para ejecutar", se le asigna memoria y se copian allí sus segmentos. Entonces, el proceso cargado compite por el procesador con todos los demás procesos cargados. Si no hay suficiente memoria, el proceso de intercambio examine la tabla de procesos para determinar cuál puede ser interrumpido y llevado al disco.

Cuando UNIX opera en máquinas más grandes, suele disponer de manejo de memoria de paginación por demanda. En algunos sistemas el tamaño de la página en UNIX es de 512 bytes; en otros, de 1024. Para reemplazo se usa un algoritmo que mantiene en memoria las páginas empleadas más recientemente.

Un sistema de paginación por demanda ofrece muchas ventajas en cuanto a flexibilidad y agilidad en la atención concurrente de múltiples procesos y proporciona además, memoria virtual, es decir, la capacidad de trabajar con procesos mayores que el de la memoria central. Estos esquemas son bastante complejos y requieren del apoyo de hardware especializado.

1.6.4 MANEJO DE ENTRADAS Y SALIDAS

El sistema de E/S se divide en dos sistemas complementarios: el estructurado por bloques y el estructurado por caracteres. El primero se usa para manejar cintas y discos magnéticos, y emplea bloques de tamaño fijo (512 o 1024 bytes) para leer o escribir. El segundo se utiliza para atender a las terminales, líneas de comunicación e impresoras, y funciona byte por byte.

En general, el sistema UNIX emplea programas especiales (escritos en C) conocidos como manejadores (drivers) para atender a cada familia de dispositivos de E/S. Los procesos se comunican con los dispositivos mediante llamadas a su manejador. Además, desde el punto de vista de los procesos, los manejadores aparecen como si fueran archivos en los que se lee o escribe; con esto se logra gran homogeneidad y elegancia en el diseño.

Cada dispositivo se estructura internamente mediante descriptores llamados número mayor, número menor y su clase (de bloque o de caracteres). Para cada clase hay un conjunto de entradas en una tabla que aporta a los manejadores de los dispositivos. El número mayor se usa para asignar manejador correspondiente a una familia de dispositivos; el menor pasa al manejador como un argumento y éste lo emplea para tener acceso a uno de varios dispositivos físicos semejantes.

Las rutinas que el sistema emplea para ejecutar operaciones de E/S están diseñadas para eliminar las diferencias entre los dispositivos y los tipos de acceso. No existe distinción entre acceso aleatorio y secuencial, ni hay un tamaño de registro lógico impuesto por el sistema. El tamaño de un archivo ordinario está determinado por el número de bytes escritos en él; no es necesario predeterminar el tamaño de un archivo.

El sistema mantiene una lista de áreas de almacenamiento temporal (*buffers*), asignadas a los dispositivos de bloques. El Kernel usa estos buffers con el objeto de reducir el tráfico de E/S. Cuando un programa solicita una transferencia, se busca primero en los buffers internos para ver si el bloque que se requiere ya se encuentra en la memoria principal (como resultado de una operación de lectura anterior). Si es así, entonces no será necesario realizar la operación física de entrada o salida.

Existe todo un mecanismo de manipulación interna de buffers (y otro de manejo de listas de bytes), necesario para controlar el flujo de datos entre los dispositivos de bloques (y de caracteres) y los programas que los requieren.

Por último, y debido a que los manejadores de los dispositivos son programas escritos en lenguaje C, es relativamente fácil reconfigurar el sistema para ampliar o eliminar dispositivos de E/S en la computadora, así como para incluir tipos nuevos.

1.7 MODELOS DE COMUNICACION EN UNIX

1.7.1 ANTECEDENTES. REDES DE COMPUTADORAS

Actualmente el volumen de información a procesar se ha incrementado considerablemente y los sistemas de información tienden a ser más complejos. Esto ha dado la pauta para que los trabajos que antes realizaba una sola computadora, se distribuyan ahora entre varias computadoras que deben ser capaces de comunicarse entre sí y trabajar de manera conjunta para satisfacer los actuales requerimientos de información.

Esta comunicación puede darse entre computadoras que estén físicamente cercanas como en un mismo edificio o geográficamente tan distantes como en diferentes continentes.

Hace poco menos de 30 años surgieron las primeras redes de computadoras y desde entonces se han aportado elementos valiosos a las redes que hoy conocemos. A continuación se citan algunas fechas importantes para las redes de computadoras:

- En 1960 se inicia la historia de las redes con el establecimiento de la conmutación de paquetes, que es un método de fragmentación de mensajes en subpartes llamadas paquetes (packets), estos paquetes son dirigidos hacia un receptor, una vez ahí, todas las partes se unen en forma correcta para recuperar la información original.
- En 1968 opera la primer periferia para la conmutación de paquetes en las Laboratorios de Física Nacional de UK. Después la Sociedad Internacional de Telecomunicaciones y Aeronáutica hace experimentos con esto.
- En diciembre de 1969, surge la primera red experimental llamada ARPANET. Desarrollada por la Agencia de Proyectos e Investigaciones Avanzados (ARPA) del Departamento de Defensa de los Estados Unidos de América, esta red contaba con cuatro nodos y conectaba hasta cien computadoras ubicadas en varios estados del país. El propósito de el diseño de dicha red fue el de comunicar todos los centros militares que sobrevivieran a una posible catástrofe nuclear.
- En 1972 se comenzó a considerar a INTERNET como una red de redes. Internet surgió del Ministerio de los Estados Unidos y de esta manera los distintos centros de Investigación instalaron sus propios nodos.
- En 1973, la compañía XEROX desarrolla una red de gestión de archivos en base a sus equipos instalados en EUA. Esta red fue la pionera de las redes Ethernet.
- En 1974, comienza a funcionar la red pública TRANSPAC de Francia, la cual conecta cientos de equipos en todo el país, ésta fue una de las primeras redes públicas.
- En 1981, México pone en marcha su red pública TELEPAC para ofrecer servicios de transmisión de datos en todo el país.
- En ese mismo año, aparecen las PC's marcando un cambio definitivo en la informática y comienzan a desarrollarse las primeras redes de microcomputadoras.

Ante estos continuos avances de la informática y las telecomunicaciones, la Organización Internacional de Normas (ISO) y la Unión Internacional de Telecomunicaciones (UIT), a través del Comité Consultivo de Telefonía y Telegrafía (CCITT), deciden establecer las primeras Normas de conectividad de equipos en redes de computadoras, con lo que quedan establecidas las bases fundamentales de las redes de cómputo.

1.7.1.1 TIPOS DE REDES

Una red de computadoras es un conjunto de computadoras y periféricos conectadas entre si, a través de medios de comunicación (líneas telefónicas, cable coaxial, fibra óptica y microondas) con objetivos primarios específicos: compartir información, comunicar usuarios, compartir dispositivos y tener flexibilidad en el manejo de la información. A pesar de las características mencionadas, existen ciertas particularidades que diferencian unas redes de otras. Las redes se clasifican de la siguiente manera:

Por el tipo de propietarios:

- Redes Privadas. Su característica es de que sólo un grupo reducido de personas tienen acceso a la red (los propietarios, socios, empleados o estudiantes).
- Redes Comerciales. Rentan sus servicios a personas interesadas en tener acceso a la información de la red.
- Redes Públicas. Son administradas generalmente por el gobierno en países subdesarrollados y por grandes consorcios en países capitalistas.

Por su extensión geográfica:

- Redes de Area Amplia (Wide Area Network). La extensión geográfica que abarca una red WAN puede ser desde una pequeña ciudad, cubrir en su totalidad el territorio de un país, o unir países enteros.
- Redes de Area Metropolitada (Metropolitan Area Network).. Se diferencian de las Wan en que los equipos de comunicación no son tan complejos pues no se transmite a distancias muy grandes.
- Redes de Area Local (Local Area Network). Están confinadas a un espacio físico restringido y comparten periféricos de costo elevado, entre computadoras que comparten la red.

1.7.1.2 PROTOCOLOS

Para que las computadoras de una red se puedan comunicar no solamente hay que conectarlas físicamente, si no que es necesario tener un medio llamado protocolo, por el cual se especifican las reglas de la comunicación en la red. Un protocolo se puede pensar como un lenguaje, una manera de comunicarse usando un conjunto común de los términos que son entendidos por cada uno que los emplea. Los protocolos usados por las redes no son sino reglas estrictas para el intercambio de mensajes entre dos o más nodos, los principales conjuntos de protocolos son OSI y TCI/IP.

OSI

El Modelo de Referencia de Interconexión de Sistemas Abiertos, conocido mundialmente como *Modelo OSI (Open System Interconnection)*, fue creado por la ISO (Organización Estándar Internacional), con el fin de poner orden entre todos los sistemas y componentes requeridos en la transmisión de datos, además de simplificar la interrelación entre fabricantes. Así, todo dispositivo de cómputo y telecomunicaciones podrá ser referenciado al modelo y por ende concebido como parte de un sistemas interdependiente con características muy precisas en cada nivel. Esta idea da la pauta para comprender que el modelo OSI existe potencialmente en todo sistema de cómputo y telecomunicaciones, pero que solo cobra importancia al momento de concebir o llevar a cabo la transmisión de datos. El Modelo OSI cuenta con 7 capas o niveles (figura 1.3), que resumen la manera en la cual se pueden comunicar las máquinas.

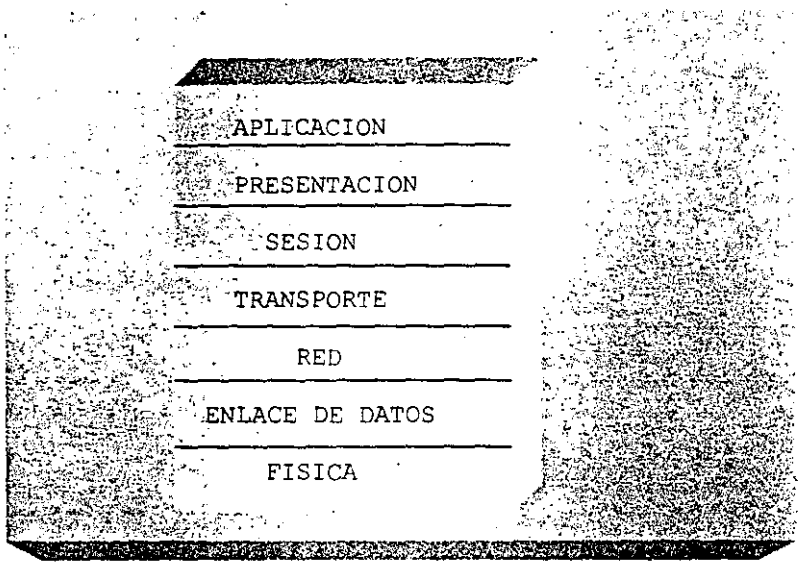

Figura 1.4 Estructura del modelo OSI

CAPAS DEL MODELO OSI

- Capa física. Define las características físicas y eléctricas que existen en el hardware de los sistemas conectados. Por ejemplo, el cableado, las tarjetas, conectores, etc.
- Capa de enlace de datos. Se encarga de enviar los bits a través del nivel físico, detecta errores y los corrige, por requerimiento de retransmisión de paquetes o mensajes mal transmitidos. Se divide en dos subcapas: El control de acceso al medio (MAC) y el control de enlace lógico (LLC). La subcapa MAC tiene la responsabilidad del acceso de red. La subcapa LLC opera a través de la MAC enviando y recibiendo los datos.
- Capa de red. Se encarga que los protocolos de enrutamiento como *RIP (Routing Information Protocol), OSPF (Open Shortest Path First), IGRP (Interior Gateway Routing Protocol)*, entre otros que viven en este nivel deciden cual es la mejor ruta para el paquete, es decir por donde se va .
- Capa de transporte. Se encarga del flujo de datos del transmisor al receptor verificando la integridad de los mismos por medio de algoritmos de detección y corrección de errores, organiza secuencialmente los paquetes de mensajes, así como también detecta y descarta aquellos que son duplicados, y regula el flujo de tráfico.
- Capa de sesión. Este nivel es el encargado de proveer servicios de conexión entre las aplicaciones, tales como iniciar, mantener y finalizar una sesión.

- Capa de presentación. Provee la representación de datos, es decir, mantiene la integridad y valor de los datos independientemente de la representación.
- Capa de aplicación. Es el nivel más cercano al usuario y a diferencia de los demás niveles, por ser el más alto o el último, no proporciona un servicio a ningún otro nivel.

TCP/IP

Para que este conjunto de protocolos pueda funcionar debe existir comunicación entre las distintas máquinas conectadas a la red, deben usar el mismo protocolo de comunicaciones y deben fraccionar la información que se transmite en los que se insertarán las direcciones de las computadoras origen y destino, asegurándose de que la información transmitida llegue intacta a su destino. Para ello elige las rutas más convenientes hasta el receptor y tras hacer comprobaciones para que la información original llegue a su destino de forma que ésta quede como inicialmente se envío. Al momento de enfrentarse con un problema, TCP/IP utiliza el sistema de dividir el problema en pequeñas porciones y dar solución a la porción en problema. Por otro lado puede funcionar en máquinas de cualquier tamaño y soporta múltiples tecnologías.

Existe una fuerte relación entre TCP/IP y OSI la cual se observa en la figura 1.4

<table>
<tr><td>OSI</td><td colspan="6"></td></tr>
<tr><td>Aplicación</td><td colspan="6"></td></tr>
<tr><td>Presentación</td><td>TELNET</td><td>FTP</td><td>SNP</td><td>SMTP</td><td>DNS</td><td>HTTP</td></tr>
<tr><td>Sesión</td><td colspan="6"></td></tr>
<tr><td>Transporte</td><td colspan="6">TCP</td></tr>
<tr><td>Red</td><td colspan="6">IP</td></tr>
<tr><td rowspan="2">Liga de Datos</td><td colspan="4">802.5</td><td>X.25</td><td>LLC/SNAP</td></tr>
<tr><td>802.3</td><td colspan="2">802.5</td><td>LAPB</td><td colspan="2">ATM</td></tr>
<tr><td>Física</td><td>Ethernet</td><td>Token Ring</td><td>FDDI</td><td>Línea Síncrona</td><td>WAN</td><td>SONET</td></tr>
</table>

Figura 1.5 Relación del modelo TCP/IP con el modelo OSI

Las capas de TCP/IP (Ver figura 1.5) se explican a continuación:

- Capa de aplicación. Corresponde a las aplicaciones que están disponibles para los usuarios, como TELNET, FTP, SNMP, etc. No transporta ni envía ningún dato a cualquier lugar excepto a la capa de transporte en una cara y a las aplicaciones en la otra. Proporciona rutinas para la presentación de los datos. Por ultimo se comunica con la capa de aplicación en la otra máquina.
- Capa de transporte. Provee comunicación extremo a extremo desde un programa de aplicación a otro. Puede proveer un transporte confiable asegurándose de que los datos lleguen sin errores y en la secuencia correcta. Coordina a múltiples aplicaciones que se encuentren interactuando con la red simultáneamente de tal manera que los datos que envíe una aplicación sean recibidos correctamente por la aplicación remota. En esta capa se encuentran los protocolos UDP y TCP.

    1. El protocolo UDP (Protocolo de Datagramas de Usuario). Proporciona aplicaciones con un tipo de servicio de datagramas orientado a transacciones. No es fiable y no esta orientado a la conexión. El UDP es simple, eficiente e ideal para aplicaciones como el TFTP y el DNS.
    2. El protocolo TCP(Protocolo de Control de Transmisión). Proporciona un servicio de comunicación que forma un circuito. Los programas que utilizan el TCP tienen un servicio de conexión entre los programas llamados y los que llaman, un chequeo de errores, control de flujo y capacidad de interrupción.
- Capa de red (IP, Protocolo de Internet).Controla la comunicación entre un equipo y otro. Conforma los paquetes IP que serán enviados por la capa inferior. Desencapsula los paquetes recibidos pasando a la capa superior la información dirigida a una aplicación. Transmite los datos por la red después de determinar la mejor ruta disponible.
- Capa física. Este nivel corresponde al hardware. En este nivel están los protocolos ARP y RARP.

    1. El protocolo ARP (Address Resolution Protocol), es el encargado de convertir las direcciones IP en direcciones de la red física.
    2. El protocolo RARP (Reverse Address Resolution Protocol) es el encargado de asignar una dirección IP a una dirección física.

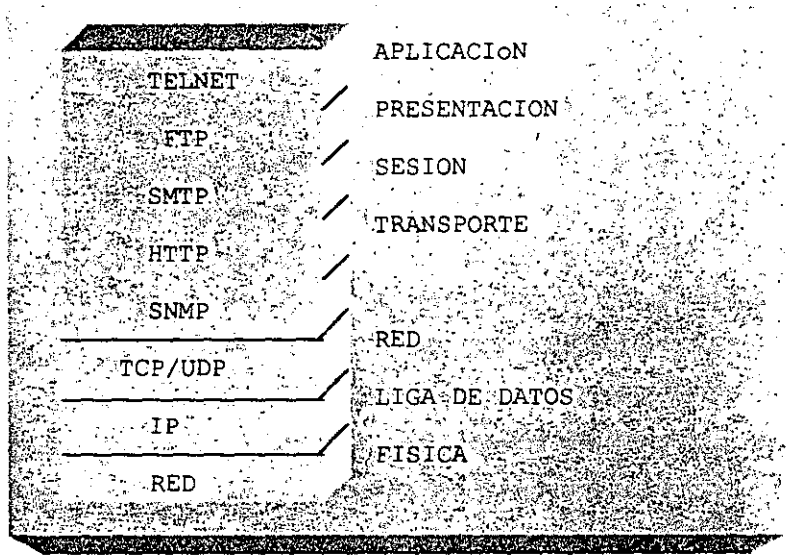

Figura 1.6 Modelo de capas de TCP/IP en relación con OSI

1.7.2 EL MODELO CLIENTE/SERVIDOR

Las tecnologías computacionales modernas buscan responder a las necesidades de los usuarios propias de la época, para ello se plantean nuevas formas trabajar con las computadoras. Algunas de estas tecnologías son el uso de la red y modelos de software que lo permitan.

Uno de los modelos de software más utilizados para redes es el denominado modelo cliente/servidor. Este esquema es un modelo de computación en el que el procesamiento requerido para ejecutar una aplicación o conjunto de aplicaciones relacionadas se divide en dos o más procesos que cooperan entre sí. La clave para comprender el concepto de cliente/servidor es entender la relación lógica entre una entidad que demanda un servicio (esto es un cliente) a otra entidad que responde la petición a favor del cliente (el servidor). Usualmente la mayoría del trabajo pesado se hace en el proceso llamado servidor y el (los) proceso(s) cliente(s) sólo se ocupan de la interacción con el usuario (aunque esto puede variar).

Como su nombre lo indica, el proceso servidor provee los servicios al proceso cliente, normalmente por una vía de proceso específico que solo ellos pueden entender. El proceso cliente se libra de la complejidad, solo envía su petición y puede realizar otro trabajo útil.

1.7.2.1 EL SERVIDOR

Los servidores proporcionan un servicio al cliente y devuelven los resultados. En algunos casos existen procesos auxiliares que se encargan de recibir las solicitudes del cliente, verificar la protección, activar un proceso servidor para satisfacer el pedido, recibir su respuesta y enviarla al cliente. Además deben manejar los interbloqueos, la recuperación ante fallas y otros aspectos afines. Por las razones anteriores la plataforma computacional asociada con los servidores es más poderosa que la de los clientes. Por esta razón se utilizan PC's poderosas, estaciones de trabajo, minicomputadoras o sistemas de cómputo grandes. Por otra parte deben manejar servicios como administración de la red, mensajes, control y administración de la entrada al sistema (login), auditoría y recuperación y contabilidad. Usualmente en los servidores existe algún tipo de servicio de base de datos.

Para que los clientes y los servidores puedan comunicarse se requiere una infraestructura de comunicaciones, la cual proporciona los mecanismos básicos de direccionamiento y transporte. La mayoría de los sistemas Cliente/Servidor actuales se basan en redes locales y por lo tanto utilizan protocolos no orientados a conexión, lo cual implica que las aplicaciones deben hacer las verificaciones. La red debe tener características adecuadas de desempeño, confiabilidad, transparencia y administración.

1.7.2.2 EL CLIENTE

Los clientes interactúan con el usuario, usualmente en forma gráfica. Frecuentemente se comunican con procesos auxiliares que se encargan de establecer conexión con el servidor, enviar el pedido, recibir la respuesta, manejar las fallas y realizar actividades de sincronización y de seguridad.

Cualquier estación de trabajo puede usarse como cliente, si ésta puede compartir simultáneamente memoria a varios usuarios, entonces también puede usarse como servidor. Cuando una estación de trabajo cliente se encuentra conectada a una red de área local, ésta puede accesar a los servicios proporcionados por el sistema operativo de red, adicionalmente a aquellos proporcionados por el servidor.

1.7.2.3 VENTAJAS E INCONVENIENTES

El modelo cliente/servidor facilita la integración entre sistemas diferentes y compartir información, permitiendo, por ejemplo que las máquinas ya existentes puedan ser utilizadas con interfaces más amigables al usuario. De esta manera, se pueden integrar PC's con sistemas medianos y grandes, sin que las máquinas tengan que utilizar el mismo sistema.

Al favorecer el uso de interfaces gráficas interactivas, los sistemas construidos con este modelo tienen una interacción más intuitiva con el usuario. Si se utilizan interfaces gráficas para interactuar con el usuario, el modelo cliente/servidor presenta la ventaja, con respecto a uno centralizado, de que no es siempre necesario transmitir información gráfica por la red pues ésta puede residir en el cliente, lo que quiere decir que el servidor solo se envía los datos y es el cliente el encargado de darle una salida gráfica evitando dando a la red solo el trabajo necesario y no saturarla de información no necesaria.

Es muy importante que los clientes y servidores utilicen el mismo mecanismo (por ejemplo sockets o RPC que más adelante se explican), lo cual implica que se deben utilizar mecanismos generales que existan en diferentes plataformas. Además hay que tener estrategias para el manejo de errores y para mantener la consistencia de los datos.

La seguridad es otro factor importante, se deben hacer verificaciones en el cliente y en el servidor, también se puede recurrir a técnicas como el encriptamiento de datos.

1.7.3 COMUNICACION ENTRE PROCESOS

La comunicación entre procesos (IPC del inglés Interprocess Comunication) permite a los sistemas UNIX la comunicación de dos o más procesos con algún otro.

En el modelo cliente/servidor siempre existe la necesidad de la comunicación entre procesos. Los clientes y servidores siempre deben comunicarse con peticiones y respuestas. En realidad, una gran parte de aplicaciones cliente/servidor están implicadas con el IPC.

El uso de comunicación entre los procesos es inherente en ambientes de sistemas multitarea. Las tareas activas operan independientemente, reciben requisiciones y envían respuestas vía los IPC.

Para una implementación efectiva de cualquier aplicación cliente / servidor, las IPC's son usadas en aquellas operaciones que implican comunicación entre los procesos en una simple máquina o a través de diferentes máquinas de una red.

Algunos de los servicios que proporcionan las IPC son los siguientes:

- Protocolos para coordinar cuando se envían y reciben los datos entre los procesos.
- Mecanismos de espera que permiten a los procesos ejecutarse asíncronamente.
- Soporte para intercambios de datos de muchos a uno (un servidor tratando con muchos clientes).

1.7.4 SOCKETS

Los *sockets* son mecanismos de comunicación entre procesos que permiten que un proceso "hable" (emita o reciba información) con otro proceso, incluso estando estos procesos en distintas máquinas. Esta característica de interconectividad entre máquinas hace que el concepto de socket nos sirva de gran utilidad. Esta interfaz de comunicaciones es una de las atribuciones de Berkeley al sistema UNIX, implementándose las utilidades de interconectividad de este Sistema Operativo (rlogin, telnet, ftp,...) usando sockets.

El programador ve el *socket* como un lugar en el que puede escribir y del que puede leer, como si de un archivo se tratase (salvando lógicamente las diferencias de tratamiento que no pueden ser escondidas). Por supuesto, un socket deberá estar asociado a un número de puerto y a un protocolo específico del nivel de transporte.

La comunicación entre procesos a través de *sockets* se basa en la filosofía cliente/servidor: un proceso en esta comunicación actuará de proceso servidor creando un socket cuyo nombre conocerá el proceso cliente, el cual podrá "hablar" con el proceso servidor a través de la conexión con dicho socket nombrado. El proceso crea un *socket* sin nombre cuyo valor de vuelta es un *descriptor* sobre el que se leerá o escribirá, permitiéndose una comunicación bidireccional, característica propia de los sockets y que los diferencia de los pipes, o canales de comunicación unidireccional entre procesos de una misma máquina. El mecanismo de comunicación vía sockets tiene los siguientes pasos:

1°) El proceso servidor crea un socket con nombre y espera la conexión.
2°) El proceso cliente crea un socket sin nombre.
3°) El proceso cliente realiza una petición de conexión al socket servidor.
4°) El cliente realiza la conexión a través de su socket mientras el proceso servidor mantiene el socket servidor original con nombre.

Es muy común en este tipo de comunicación crear un proceso una vez realizada la conexión, que se ocupe del intercambio de información con el proceso cliente mientras otro proceso servidor sigue aceptando conexiones. Para eliminar esta característica se cerrará el descriptor del socket servidor con nombre en cuanto realice una conexión con un proceso socket cliente.

Todo socket viene definido por dos características fundamentales:

- El tipo del socket, que indica la naturaleza del mismo, el tipo de comunicación que puede generarse entre los sockets.
- El dominio del socket especifica el conjunto de sockets que pueden establecer una comunicación con el mismo.

Los tipos de sockets disponibles son los siguientes:

SOCK_STREAM. Establece un socket sobre protocolo TCP.

SOCK_DGRAM. Establece un socket sobre protocolo UDP.

SOCK_RAW. Establece un socket sobre protocolo IP.

Una vez establecido el canal de comunicación entre el cliente y el servidor, hay cinco llamados posibles a primitivas del sistema operacional para enviar datos a través de dicho canal : send, sendto, sendmsg, write y writev. Las primitivas send, write y writev solo pueden ser utilizadas si el socket esta asociado con una dirección destino (en términos de TCP/IP sería si el socket es orientado a conexión). Las diferencias entre estas tres primitivas son muy pocas.

A continuación se describe la primitiva write.

```
write(socket,buffer,longitud).
```

Esta primitiva es la misma que se utiliza para el manejo de archivos en Unix. Es decir, desde el punto de vista del proceso usuario no hay ninguna diferencia entre enviar datos por un canal de comunicaciones y una escritura a un archivo.

1.7.5 PUERTOS

Los puertos pueden verse como puntos de anclaje para conexiones de red. Todas las computadoras que utilizan el protocolo TCP/IP permiten a los programas conectarse a los diferentes puertos de otra computadora; cada puerto es la entrada de algún servicio de red como mail, finger, ftp, etc. Por ejemplo, si una aplicación quiere ofrecer cierto servicio a través de la red, se le asigna un determinado puerto en donde espera las peticiones de los clientes (a esto también se le denomina "escuchar en un puerto"). Un cliente que quiere usar este servicio consigue un puerto libre en su nodo local y se conecta al puerto del servidor en el nodo remoto.

Los puertos tienen una propiedad importante: una vez que se ha establecido una conexión entre el cliente y el servidor, otra copia del servidor tiene la posibilidad de responder en el mismo puerto a peticiones de otros clientes. TCP es capaz de distinguir unas conexiones de otras, ya que todas ellas provienen de diferentes puertos o nodos.

El mecanismo de puerto se basa en la asignación para cada uno de los protocolos del nivel de transporte (TCP o UDP) de un conjunto de puertos de E/S identificados mediante un número entero. Así TCP y UDP multiplexarán las conexiones por medio de los números de los puertos. Existen una serie de puertos reservados a aplicaciones estándares Internet (echo - puerto 7, ftp - puerto 21, telnet - puerto 23). El archivo /etc/services contiene la lista de los puertos estándar. En UNIX están reservados los números de puerto inferiores a 1024. El resto pueden ser utilizados, cualidad fundamental que aprovechan los programas definidos por el usuario para el establecimiento de comunicaciones entre *hosts* (computadoras remotas).

CAPITULO 2 ACCESANDO AL SISTEMA UNIX

En este capítulo se describe el inicio de sesión en el sistema UNIX, las políticas de seguridad y los métodos de cifrado.

2.1. EL INICIO DE SESION

El administrador del sistema es el encargado de crear lo que se denomina una cuenta en el sistema. Tener una cuenta para cierto sistema significa que un usuario puede tener acceso a los recursos de ese equipo, claro que existen restricciones para cada usuario dependiendo su posición dentro de la empresa u organización. En UNIX se puede dar una mayor prioridad a los procesos de un usuario si éste requiere de una pronta atención por parte del sistema. El administrador pide un nombre al usuario con el cual será reconocido y una palabra secreta que será la clave para poder entrar al sistema, el nombre puede ser el nombre real de la persona, las iniciales de su nombre, un alias, etc., generalmente son nombres cortos, en minúsculas y fáciles de reconocer, la contraseña debe ser una palabra fácil de recordar para el usuario, pero difícil de saber por otros usuarios o personas. Al crear una cuenta se establecen las condiciones en que ese usuario va a trabajar una vez dentro del sistema.

Una vez que se tiene asignada una cuenta estamos listos para comenzar una sesión. Para comenzar una sesión en ambiente UNIX, tenemos que dejar al sistema dos tareas: que verifique el derecho de acceso a un usuario y que se defina el entorno de trabajo de ese usuario. En el monitor aparecerá por principio un mensaje de "*login:*", significa que el sistema está listo para recibir el nombre de un usuario, se debe teclear el mismo nombre que el administrador empleó para crear la cuenta. Una vez introducido el nombre, aparecerá el mensaje "*password:*", pidiendo al usuario la palabra clave para que el sistema verifique que ese usuario es quien dice ser. Si se tecleó por alguna razón algún carácter o la palabra incorrecta, el sistema volverá a pedir el nombre de usuario con el mensaje "*login:*" y se repite el proceso.

Ya dentro del sistema, podrán aparecer ciertos mensajes o avisos puestos por el administrador o simplemente aparecerá indicador de sistema (*prompt*) indicando que está listo para comenzar a trabajar. Si se está trabajando el la consola del sistema, regularmente se cuenta con un sistema de ventanas o interfaz gráfica de usuario para establecer una comunicación más amigable del sistema con el usuario.

Cuando el administrador crea una cuenta para algún usuario, lo que hace es crear una nueva línea en archivo /etc/passwd. Este es el archivo de registro de los usuarios válidos en el sistema. La información que contiene es la siguiente:

1. Nombre de la cuenta de usuario.	Es el nombre de la cuenta.
2. Password encriptado.	Es la contraseña encriptada.
3. Identificador de Usuario (UID).	Número con el que el sistema reconoce al usuario.
4. Identificador de grupo (GID).	Número con el que el sistema reconoce el grupo al que pertenece el usuario.

5. Información del usuario.	Información particular del usuario (nombre, dirección, etc.).
6. Directorio de HOME.	Es la ruta del directorio que tendrá por defaul el usuario al entrar al sistema.
7. Shell de inicio.	Indica el shell que se ejecutará cuando el usuario entre a sesión (por ejemplo csh, ksh, sh, etc).

Es muy importante saber que para el sistema, el usuario no es más que un número, es decir, el nombre solo es conveniencia para las personas y distingan un usuario de otro, pero el sistema realmente maneja a un usuario como un número conocido como UID.

Al igual que se edita el /etc/passwd, también se debe editar el archivo /etc/group que contiene el registro de los grupos válidos en el sistema. Este archivo tiene los siguientes registros:

1. Nombre del grupo.	Es el nombre que identifica al grupo. Cada nombre puede ser representativo para las tareas que realice el grupo.
2. Password.	Facilita la compatibilidad de las versiones UNIX. Este campo anteriormente se utilizaba y ahora solo contiene un asterisco.
3. Identificador del grupo (GID).	Es el número con el que el sistema reconoce el grupo
4. Usuarios adicionales.	Este campo tiene una lista de los usuarios y grupos secundarios que podrían tener acceso a los archivos de grupo, adicionalmente a aquellos inicializados en el grupo en virtud del archivo /etc/passwd. Los nombres de los usuarios en la lista deben estar separados por una (,).

El grupo es también manejado como un número por el sistema conocido como GID, el nombre es solo para reconocer más fácilmente un grupo entre los usuarios.

Cada usuario que entra al sistema tendrá definidas las variables de ambiente, éstas son parte del entorno de UNIX, las más importantes son:

PATH	Define las trayectorias de búsqueda de los comandos de los usuarios.
HOME	Contiene el nombre del directorio de trabajo del usuario, dentro del csh el carácter (~) también hace referencia a este directorio.
TERM	Contiene el nombre del tipo de terminal que se esta utilizando.

Cuando un usuario entra a sesión es recomendable que configure su medio ambiente, para ello existen algunos archivos que llevan a cabo esta tarea y deben estar en el directorio HOME del usuario. Algunos de estos archivos son:

.cshrc	Permite configurar un ambiente de csh.
.login	Permite configurar el medio ambiente de inicio de sesión cuando se usa csh.
.profile	Tiene la misma función que .login, pero cuando se trata de bourne shell o kshell.

2.2 EL CIFRADO DEL PASSWORD

Al momento que UNIX solicita la contraseña, necesita un método para determinar que se está tecleando la contraseña correcta. UNIX mantiene cifrada la contraseña de cada usuario en el sistema dentro del archivo /etc/passwd en el segundo campo (algunos sistemas manejan el también el archivo */etc/shadow* por razones de seguridad. El algoritmo que UNIX utiliza para cifrar la contraseña es la función *crypt()* que está basado en el sistema de cifrado DES (Estándar de Cifrado de Datos).

La función *crypt()* desarrollada por Robert Morris y Ken Thompson toma la contraseña del usuario como llave de cifrado y la usa para cifrar bloques de 64 bits de ceros. El bloque resultante es cifrado nuevamente con la contraseña del usuario, este proceso es repetido un total de 25 veces. El resultado final es transformado en una cadena de 11 caracteres imprimibles y almacenada en la tabla de /etc/passwd o en el archivo /etc/shadow.

2.2.1 METODOS DE CIFRADO

Como cifrado entenderemos a la transformación de un mensaje (conocido como texto puro) a otro (llamado texto cifrado), usando un algoritmo o función matemática y una contraseña cifrada conocida como la llave. Las llaves de cifrado son similares a las contraseñas de acceso que se utilizan en otras computadoras, es decir, es necesario poseer la llave para poder acceder a la información almacenada en un archivo cifrado, pero a diferencia de éstas, en un programa cifrado la llave no se utiliza para compararla con la que se cifró el archivo (u permitir el acceso a este si son iguales); en vez de eso, el programa de cifrado utiliza la llave para la obtención del texto puro. Si la llave es válida se obtendrá el mensaje original, si no lo es, solo se obtendrá información no legible de ese archivo.

El cifrado nos sirve para tratar de garantizar la integridad de la información que manejan los usuarios de una computadora, es decir, protegerla de todas las personas que no haya sido explícitamente autorizada para verla, particularmente en un sistema multiusuario como UNIX.

Existen actualmente dos tipos de sistemas de cifrado:

- Cifrado de llave privada: Es aquella en la cual, la misma llave que se usa para cifrar es utilizada para descifrar el mensaje.
- Cifrado de llave pública: Este método utiliza dos tipos de llave, una para cifrar (llave pública) y otra para descifrar (llave privada). El nombre de llave pública viene del hecho de que es posible darla a conocer sin comprometer la confidencialidad del mensaje o las características de la otra llave.

Ejemplos de sistemas con llave privada:

- ROT13 (ROTATE13)
- CRIPT
- DES (Estándar de Cifrado de Datos)

Ejemplo de sistema con llave pública:

- RSA

El más importante por ser ampliamente utilizado hoy en día, es el sistema de llave privada DES (Estándar de Cifrado de Datos) desarrollado por IBA en los 70's. El DES funciona básicamente a partir de funciones de permutaciones, sustituciones y recombinaciones de bits, ejecutadas en bloques de 64 bits de datos y 56 de llave (cada una de ocho por 7 bits)

De entre los sistemas de llave pública, el más popular es RSA, que a diferencia de los sistemas de llave privada que utilizan una sola llave para cifrar y descifrar, este en particular utiliza dos llaves: una llamada llave pública para cifrar y otra privada para descifrar. La fuerza de RSA está basada en las propiedades numéricas teóricas de aritmética modular e integrales. Una de estas funciones es la función de "Totien" de Euler, la cual se define como el número menor de integrales para que ese número sea relativamente primo.

Actualmente existen muchas aplicaciones las cuales cuentan con un sistema de cifrado, las más populares por el momento son:

- PGP (Pretty Good Privacy). Es un programa de cifrado que se utiliza para proteger la privacidad del correo electrónico y los archivos que se encuentran almacenados en la computadora, además, PGP permite firmar digitalmente un texto. El cifrado que utiliza PGP al utilizar la llave pública se realiza en dos partes debido a la gran cantidad de operaciones matemáticas que se necesitan. El primer paso es realizado por medio de un algoritmo de llave privada conocido como IDEA desarrollado en Suiza por James Messey y Xueejia Lai, que utiliza una llave conocida como llave de sesión que como su nombre lo indica, se genera una diferente para cada sesión. Para el segundo paso se utiliza el algoritmo llamado RSA.

- S/KEY. Es una herramienta de seguridad para evitar los ataques pasivos (ataques por medio del monitoreo de la red). S/KEY utiliza el método de "contraseñas de una sola vez", este método consiste en cambiar la contraseña cada vez que se termina una sesión. El usuario solo tiene que crear una sola contraseña "fuerte" y S/KEY la va cambiando automáticamente después de cada sesión.

- PEM (Privacy Enhanced Mail). Se puede definir como una serie de reglas de cifrado (o estándar propio de los servicios de correo y no un producto como comúnmente se llama) para la autentificación de la información que viaja a través de los sistemas de E-mail.

2.3 LA ASIGNACION DE LA TERMINAL

El sistema UNIX tiene, además de archivos ordinario y directorios, otro tipo de archivos, los archivos de dispositivo; éste reside en la estructura de archivo del sistema operativo UNIX, por lo general en el directorio /dev, y representa un dispositivo periférico como una terminal, una impresora o unidad de disco.

Al entrar un usuario en sesión, UNIX asigna una terminal a cada usuario, esto es, ya que UNIX maneja todo como si fuesen archivos, incluyendo las terminales, se asigna un nombre de archivo al usuario para que éste sea el lugar donde el shell dirija la salida estándar. El shell dirige la salida estándar de los programas al archivo de dispositivo que representa a la terminal de usuario, al direccionar la salida de esta forma, se hace que aparezca en la pantalla de la terminal. El shell también dirige la entrada estándar para que provenga del mismo archivo, así lo digitado en el teclado de la terminal será ingresado en el programa.

Un nombre común dado a los archivos que representan una terminal es:

/dev/ttyNN donde NN es un número asignado por el sistema.

En UNIX existe un comando para ver el nombre de la terminal usada por el usuario. Si se teclea *% tty* se puede obtener como respuesta algo como esto: *% /dev/tty09* , lo que significa que se está utilizando la terminal número 9 y que esta terminal está siendo tratada como un archivo del directorio */dev* (de *dev*ice).

2.4 LA SEGURIDAD DE LA CUENTA

El sistema UNIX proporciona la posibilidad de proteger una cuenta mediante el uso de identificadores y permisos tanto para los archivos como para los procesos.

2.4.1 SISTEMA DE PERMISOS

El número identificador de usuario (UID) es un entero que se encuentra en el archivo /etc/passwd y que está asociado con el nombre del login del usuario. Cuando un usuario inicia una sesión, la orden */bin/login* convierte este número identificador en el número de usuario asociado al primer proceso creado, el intérprete de órdenes. Los procesos que vayan ejecutándose a partir de ese momento llevarán adherido este número de identificación. Los procesos también están organizados en grupos, los cuales poseen un número de identificación llamado número de identificación de grupo (GID), que también se encuentra en el archivo /etc/passwd, que se convertirá en el número de identificación adherido al shell. Estos grupos están definidos en el archivo /etc/group. Estos dos números de identificación son denominados reales, porque son representativos del usuario real, o sea, el que ejecutó el proceso login. Existen otros dos números de identificación que también estarán asociados a cada proceso: el número de identificación de usuario efectivo y el número de identificación de grupo efectivo. Éstos suelen ser iguales a los reales, pero pueden ser distintos, y se usan para determinar los permisos, mientras que los reales se usan para saber la identidad verdadera del usuario. Cada archivo (ya sea ordinario, directorio o especial) contiene en su inodo el UID de su propietario y el GID de su grupo propietario, el conjunto de permisos de lectura, escritura y ejecución para el propietario, grupo y otros, además de otros datos concernientes al archivo. Este conjunto de permisos determina cuándo un proceso puede ejecutar una acción (lectura, escritura o ejecución) en un archivo dado. En los archivos ordinarios, estas tres acciones son obvias. En directorios, la acción de escritura significa poder modificar el directorio añadiendo o borrando un enlace, mientras que la acción de ejecución significa que puede ser incluido en un PATH. En los archivos especiales, las acciones de lectura y escritura significan la posibilidad de poder utilizar las llamadas al sistema *read*[9] *y write*[10]. Este sistema de permisos funciona de la siguiente forma:

- Si el número de identificación de usuario efectivo es 0 (root), entonces, se dan los permisos como propietario.
- Si el número de identificación de usuario efectivo coincide con el número de identificación del usuario propietario del archivo marcado en su inodo, entonces se dan los permisos de propietario establecidos.
- Si el número de identificación de grupo efectivo coincide con el número de identificación del grupo propietario del archivo marcado en su inodo, entonces se dan los permisos de grupo.
- Si no se da ninguna de las tres anteriores suposiciones se darán los permisos establecidos para otros.

[9] Esta llamada al sistema abre archivos y lee de ellos directamente.
[10] Esta llamada al sistema abre archivos y escribe en ellos directamente.

Los derechos asociados a un archivo se representan por nueve bits (r,w,x). Un bit activo (a uno) indica el derecho correspondiente a estar activo y un bit a cero indica lo contrario. En función de estos derechos, se determina lo que cada usuario puede hacer con el archivo. Estos derechos se extienden como ya se dijo anteriormente al propietario, grupo y al resto. A su vez, estos grupos pueden tener diferentes posibilidades de acceso al archivo. Una [r] indica el derecho de lectura, una [w] de escritura, y la [x] de ejecución. El guión indica que el derecho correspondiente está desactivado. Esta secuencia de caracteres se agrupa de tres en tres. De izquierda a derecha tenemos lo siguiente: los tres primeros caracteres se corresponden con los derechos del propietario (*user*), los tres siguientes con los del grupo (*group*) y los tres últimos con el resto (*others*).

Además de estos nueve bits de derechos asociados a cada archivo, podemos considerar tres más, conocidos como bit pegajoso (*sticky bit*) bit de *set-gid* y de *set-uid*, respectivamente.

El bit de *set-uid* permite solucionar problemas relacionados con la seguridad (en ocaciones se puede modificar algún archivo donde el dueño es root, pero solo se pueden hacer ciertos cambios, y para poder hacerlo se necesita tomar la identidad del dueño por un momento). El hecho de que un programa tenga este bit activo, indica que cuando ejecutemos dicho programa, éste tomará como identificador de usuario el identificador del propietario. Si el propietario fuese root, entonces el programa se ejecutaría como si lo hubiese lanzado root. El bit de *set-uid* está activado cuando en la máscara de derechos del programa, en el campo de ejecución para el propietario tiene una *s* en lugar de una *x*.

El bit de *set-gid* es completamente similar al del bit de *set-uid* pero en este caso se aplica al grupo. El bit de *set-gid* está activado cuando en la máscara de derechos del programa, en el campo de ejecución para el grupo tiene activa una *s* en lugar de una *x*.

Por último, el *sticky-bit* indica al núcleo de UNIX que ese archivo es un programa con capacidad para que varios procesos compartan su código, y que este código se debe mantener en memoria aunque alguno de los procesos que lo ejecuta deje de ejecutarse, esto permite ahorrar memoria en el caso de trabajar con programas muy utilizados como editores o compiladores. Este bit está activo cuando en la máscara de derechos del archivo en cuestión, en el campo de ejecución del resto de usuarios, aparece una *t* en lugar de una *x*.

Existen comandos que permiten modificar los permisos de un archivo (*chmod*), cambiar un archivo o un grupo de archivos de propietario (*chown*), y cambiar un archivo de un grupo a otro (*chgrp*).

2.5 POLITICAS DE SEGURIDAD

Los recursos informáticos están sujetos a amenazas generadas por diferentes tipos de riesgos a los que están expuestos dependiendo de sus componentes y sus propósitos, por lo tanto, es necesario implementar controles para prevenir, detectar o corregir los ataques de las amenazas y para disminuir los riesgos.

La seguridad al 100% no se logra nunca, los controles deben ser proporcionales a los riesgos; la falta de protección de los recursos informáticos es muy peligrosa, pero el exceso de ésta es muy costosa. Por ello es necesario identificar dónde podrían correr riesgos y qué implicaciones originaría para el área o entidad afectada, así como también lo que costaría eliminar el riesgo o disminuir la probabilidad de ocurrencia.

Solamente así, se pueden tomar decisiones en cuanto al establecimiento de las previsiones requeridas o de los riesgos que se asumen.

La seguridad en informática involucra el establecimiento de un sistema de control para proporcionar confidencialidad, integridad y disponibilidad de la información así como ética del recurso humano. A continuación se enuncian las políticas que se pueden seguir para lograr un ambiente informático seguro.

- Estimular la creación de una cultura de seguridad en informática, concientizando a los usuarios del verdadero valor de la información del sistema.
- Garantizar la adecuada protección de los recursos informáticos.
- Identificar el nivel de sensibilidad de la información y establecer responsabilidades por su manejo.
- Informar a los empleados de los requerimientos de seguridad de la entidad.
- Definir para cada proyecto en informática los requerimientos mínimos de seguridad, de acuerdo con la sensibilidad de la información que se maneja.
- Definir atributos de seguridad para la información, los cuales se deben preservar cuando se comparta la información con otras entidades.
- Garantizar la continuidad de las operaciones de las entidades, ante una situación crítica de suspensión del servicio informático.

CAPITULO 3

SISTEMA DE REGISTRO DE ACCESOS EN UNIX

En este capítulo se describe el sistema de registro de accesos en UNIX y algunos comandos que nos ayudan a supervisar los accesos y las claves o cuentas.

3.1 REGISTRANDO LOS ACCESOS EN UNIX

Debido al constante crecimiento de las redes, y a que UNIX es muy utilizado en este ambiente, existe la posibilidad de tener varios usuarios conectados de diferentes lugares al mismo tiempo y por ello es necesario llevar un continuo y exhaustivo control de los accesos al sistema (de dónde y hacia dónde).

En general, UNIX proporciona comandos para visualizar los accesos de los usuarios al sistema. Estos comandos recopilan la información necesaria de los archivos especiales de contabilidad (utmp, wtmp, utmpx, wtmpx, lastlog y acct). Estos archivos especiales o logs almacenan los principales datos de identificación del usuario y desde donde se conecta.

Dichas bitácoras están diseñadas para llevar un registro de los usuarios conectados al sistema y tiene una aplicación práctica principalmente para los administradores, los cuales están encargados de garantizar la seguridad de la información de la máquina.

Una vez conociendo qué usuarios están accesando o no al sistema, se puede comenzar a desarrollar estrategias al respecto, por ejemplo:

- Si el usuario nunca ha accesado a su cuenta, se puede sugerir la deshabilitación de la misma.
- Si el usuario accesa de hosts remotos desconocidos, en horas poco comunes, o de diferentes lugares al mismo tiempo, se pueden tomar medidas de prevención.
- Si el usuario hace conexiones hacia varios lugares remotos y desconocidos se puede comunicarse con él y tratar de averiguar si en verdad es él quién hace esas conexiones y porqué.

3.2 ARCHIVO LASTLOG

El archivo lastlog almacena la hora del más último acceso del usuario al sistema y desde donde se lleva a cabo el acceso. En algunas versiones de System V se muestra el último acceso fallido de entrar al sistema. El comando que lo utiliza es *finger* y *login*. Se encuentra en el directorio */var/adm*, dicho archivo está en formato binario y solo lo puedes visualizar utilizando la estructura definida por el fabricante.

Para poder entender las estructuras de los archivos, debemos antes entender que una estructura es un conjunto de tipos de datos y que hacen parecer a la estructura como otro tipo de dato.

La estructura más comercial es la siguiente:

```
struct lastlog {
   time_t  ll_time;
   char    ll_line[8];
   char    ll_host[16];
};
```

En donde:

ll_time: La fecha del último acceso.
ll_line: Nombre del dispositivo, terminal por donde accedió.
ll_host: La máquina desde donde se conectó.

3.3 ARCHIVOS UTMP y WTMP

El archivo *utmp* guarda un registro (log) de los usuarios que están utilizando actualmente el sistema mientras están conectados a él. Este archivo se actualiza cada que el usuario entra y sale del sistema, lo utilizan los comandos *who, w, users* y *finger*. Se encuentra en el directorio */var/adm* o en /etc dependiendo de la configuración del sistema.

El archivo *wtmp* guarda un registro(log) cada vez que un usuario se introduce en el sistema o sale de él , también guarda algunas acciones como los *reboots*, algunos programas pueden añadir su propia línea al *wtmp*, este archivo puede llegar a ocupar bastante espacio si no se vuelve a poner en cero bits con el tiempo. Lo utilizan los comandos *who, login y last* principalmente. Se encuentra en el directorio */var/adm* o */etc* dependiendo de la configuración del sistema.

Tanto *utmp* y *wtmp* no son archivos de texto por lo cual para poderlos visualizar es necesario implementar un programa el cual contenga su estructura la cual se muestra a continuación:

```
struct utmp {
            char  ut_user[8];
            char  ut_id[4];
            char  ut_line[12];
            short ut_pid;
            short ut_type;
            struct exit_status {
                 short e_termination;
                 short e_exit;
            } ut_time;

            unsigned short ut_reserved1;
            char ut_host[16];
          unsigned long ut_addr;};
```

Donde:
ut_user: Nombre del usuario.
ut_id: Identificador de "/etc/inittab".
ut_line: Nombre del archivo de dispositivo asociado("console,ttyXX,...").
Ut_pid: Identificador del proceso.
ut_type: El tipo de entrada.
e_termination: Estado de terminación del proceso.
e_exit: Estado de salida del proceso.
ut_time: Se aplica a las entradas cuyo tipo es DEAD_PROCESS.
ut_reserved1: Reservada para usos futuros.
ut_host: Nombre del nodo.
Ut_addr: Dirección Internet del nodo.

Existen diez tipos de entrada las cuales son:

```
EMPTY          0
RUN_LVL        1
BOOT_TIME      2
OLD_TIME       3
NEW_TIME       4
INIT_PROCESS   5
LOGIN_PROCESS  6
USER_PROCESS   7
DEAD_PROCESS   8
ACCOUNTING     9
```

Cuando veamos un registro en el archivo *utmp/wtmp* con el tipo de entrada 2 querrá decir que por ejemplo se ha producido un *reboot* (si es un *reboot* o un *shutdown* el nombre de la terminal en el *wtmp* será "~" y el *username* será *reboot* o *shutdown*). Cuando se llama a la función *getty*(), ésta hace una entrada en el archivo *utmp/wtmp* poniendo el tipo de entrada a 6 , cambia los campos *ut_time* y *ut_line* y espera a que se establezca una conexión. Una vez que un usuario ha entrado en el sistema, *login*() pone el campo tipo de entrada a 7 y rellena el resto de los campos, cuando *init*() detecta que uno de los procesos ha finalizado añade otra entrada con el campo tipo de entrada a 8 y borra *ut_user, ut_host* y *ut_time,* por tanto cuando veamos un registro con el nombre de usuario vacio y de tipo de entrada 8 indicará un *logout* de esa terminal.También cuando veamos en el *wtmp* en la terminal usada los caracteres"|" y "}" quiere decir que se ha cambiado la fecha y graba en el *wtmp* la fecha antigua y nueva.

Cada registro tanto del *utmp* como del *wtmp* contiene los mismos campos, pero sin embargo hay que distingir entre el *utmp/wtmp* de los sistemas UNIX basados en BSD y los basados en System V.Cabe mencionar que en BSD no tienen los campos tipo y ID que son de bastante utilidad, System V no tiene el campo de host, aunque System V guarda un valor de retorno cuando muere un proceso por lo tanto la estructura cambia

En sistemas BSD la estructura *utmp* seria :

```
struct utmp {
 char ut_line[UT_LINESIZE];
 char ut_name[UT_NAMESIZE];
 char ut_host[UT_HOSTSIZE];
 long ut_time; };
```

Finalmente en sistemas System V la estructura utmp quedaria asi :

```
 struct utmp {
 char ut_user[8];
 char ut_id[4];
 char ut_line[12];
 short ut_pid;
 short ut_type;
 struct exit_status ut_exit;
 time_t ut_time;
 };
```

Existen llamadas al sistema y funciones para poder leer el archivo *utmp/wtmp* las cuales son:

void utmpname(const char *file);	Sirve para indicar el archivo con estructura utmp, al que accederán después el resto de las funciones, si no se especifica ninguno se toma el que hay por defecto que suele ser */var/adm/utmp*.
void setutent(void);	Posiciona el puntero al principio del archivo.
void endutent(void);	Cierra el archivo.
struct utmp *getutent(void);	Leé un registro del archivo a partir de la posición del puntero, devuelve un puntero a una estructura *utmp* donde está almacenado el registro.
void pututline(struct utmp *ut);	Escribe un registro en el archivo a partir de la posición del puntero, se le pasa como parámetro un puntero a una estructura *utmp* que contiene los datos a escribir.
struct utmp *getutline(struct utmp *ut);	Recorre el archivo desde la posición actual buscando registros cuyo tipo sea *Login_process* o *User_process* y devuelve el primero en el que coincidan el campo ut_line de la estructura que se pasa como parámetro y del registro que esta leyendo.

struct utmp *getutid(struct utmp *ut);

Es otra función de búsqueda que recorre el archivo desde la posición actual y busca según los siguientes patrones : Si el campo tipo de entrada de la estructura que se le pasa es RUN_LVL,BOOT_TIME, NEW_TIME o OLD_TIME devuelve el primer registro que encuentre con el tipo solicitado. Pero si el campo tipo de entrada es INIT_PROCESS,LOGIN_PROCESS,USER_PROCESS o DEAD_PROCESS delvuelve el primer registro que encuentre en el que coincida el campo id.

3.4 ARCHIVOS UTMPX y WTMPX

A estos archivos se les denomina utmp/wtmp extendidos, se usan en sistemas como Solaris, IRIX y casi todos los SVR4 actuales, los registros de estos archivos son como los de *utmp/wtmp* pero con más campos extra, vienen a compensar la falta de algunos campos bastante importantes en el *utmp/wtmp* de los System V ahora mismo conviven los dos *utmp/wtmp* y *utmpx/wtmpx*, pero solo por razones de compatibilidad, el *utmp/wtmp* acabará desapareciendo. Se encuentran en el directorio */var/adm* o */etc* al igual que *utmp* y *wtmp* estos archivos están en formato binario y es necesario saber su estructura la cual se muestra a continuación:

```
struct utmpx  {
      char    ut_user[32];
      char    ut_id[4];
      char    ut_line[32];
      pid_t   ut_pid;
      short   ut_type;
      struct  exit_status ut_exit;
      struct  timeval ut_tv;
      long    ut_session;
      long    pad[5];
      short   ut_syslen;
      char    ut_host[257];
      };
```

ut_user: Nombre del usuario.
ut_id: Identificador de "/etc/inittab".
ut_line: Nombre del archivo de dispositivo asociado("console,ttyXX,...").
ut_pid: Identificador del proceso.
ut_type: Tipo de entrada.
ut_exit: Estado de salida de un proceso.
ut_tv: El tiempo de la entrada.

ut_session: Identificador de sesión.
pad: Reservada para usos futuros.
ut_syslen: Longitud de ut_host
ut_host: Nombre del host remoto desde donde se realizó la conexión.

El tipo de entradas son iguales a las de *utmp* y *wtmp* respectivamente.

Estos archivos extendidos proporcionan a diferencia de *utmp* y *wtmp*, un nombre de usuario de 32 caracteres en vez de 8, el tipo de dispositivo asociado de 32 caracteres y el host remoto de hasta 256 caracteres.

Las llamadas al sistema y funciones que gestionan los archivos *utmpx/wtmpx* son idénticas a las de *utmp/wtmp* pero añadiéndole una x al nombre:

```
struct utmpx *getutxent(void);
struct utmpx *getutxid(const struct utmpx *);
struct utmpx *getutxline(const struct utmpx *);
struct utmpx *pututxline(const struct utmpx *);
void setutxent(void);
int utmpxname(const char *);
```

3.5 ARCHIVO ACCT

El archivo acct (qué pertenece a la contabilidad del sistema) registra todos los comandos ejecutados por cada usuario, permite saber qué procesos están siendo ejecutados por el sistema y que usuarios los ejecutan. Se encuentra escrito en formato binario por lo que no se puede visualizar con un simple editor. Uno de los inconvenientes que tiene es su crecimiento en algunas ocasiones exagerado.

Dependiendo del sistema operativo el acct se puede llamar *pacct* y se encuentra de acuerdo a la tabla 3.1, donde se puede observar la trayectoria completa de la ubicación del archivo.

VERSIONES		
BSD	BSD	/usr/adm/acct
	SUNOS	/var/adm/acct
	V.3	/usr/adm/pacct
	V.4	/var/adm/pacct
SYSTEM V	AIX 3.1	/usr/adm/pacct
	XENIX	/usr/adm/pacct
	IRIX	/var/adm/pacct

Tabla 3.1 Archivos de registro de los procesos

El registro acct contienen los siguientes datos de los procesos:

- Nombre de la imagen.
- Tiempo de CPU.
- Tiempo transcurrido.
- Tiempo de inicio del proceso.
- Memoria utilizada.
- Número de caracteres leídos y escritos.
- Número de bloques de E/S leídos y escritos en disco.
- Terminal inicial TTY.
- Varias banderas asociadas con los procesos, incluyendo estado de salida.

Esta bitácora solo funciona si la contabilidad está activada debido a que es generado por ella. Para poder activar o desactivar la contabilidad existen varios comandos dependiendo de la versión esto se muestra en la tabla 3.2.

BSD **Por default la contabilidad esta activada**	Se utiliza el comando accton y se especifica un archivo contable como su argumento además es necesario asegurarse que el archivo acct se encuentre ya que el sistema por si solo no lo crea y para eso nos podemos auxiliar de el comando touch el cual crea al archivo. Todo esto se realiza siendo el superusuario o root. # touch /usr/adm/acct # accton /usr/adm/acct Una vez que el comando es ejecutado, la contabilidad se registrará automáticamente sobre el archivo acct.	Para deshabilitarla se utiliza el comando accton sin argumentos y se puede borrar el archivo si así se requiere. # accton
SYSTEM V **La contabilidad es más elaborada y por default no esta activada**	En este caso se utiliza el comando chconfig para poder activar la contabilidad y startup para levantarla. # chconfig acct on # startup	Para poder desactivar la contabilidad se utiliza el comando chconfig y shutacct. # chconfig acct off # usr /usr/lib/acct/shutacct

Tabla 3.2 Comandos para habilitar y deshabilitar la contabilidad.

3.6 COMANDOS UTILES EN EL MONITOREO DE USUARIOS.

El sistema operativo UNIX proporciona una serie de herramientas útiles para la obtención de información de qué, cómo y hacia dónde realizan salidas o conexiones (remotas o locales) los usuarios propios del sistema, algunas de ellas permiten no sólo obtener información de los usuarios de nuestro sistema, sino también de otros sistemas remotos. Las herramientas que UNIX ofrece a los usuarios y al administrador mismo del sistema son muy utilizadas y efectivas, aunque la información obtenida no siempre es fácil de entender por cualquier usuario, y por otro lado no siempre se está monitoreando por lo tedioso que resulta ser esta tarea. A continuación veremos cómo y para qué utilizar las herramientas más comunes.

3.6.1 COMANDO FINGER.

El comando finger es una herramienta útil para obtener información sobre un usuario o persona dentro de una red. Esta búsqueda corre bajo UNIX y se puede buscar una persona o usuario dentro de la misma máquina, dentro de la misma red o en la red de Internet.

Podemos ver un solo usuario de nuestro sistema u otros sistemas de la red, de igual manera podemos ver información sobre todos los usuarios de nuestro sistema o de todos los usuarios de otros sistemas.

La sintáxis de este comando es:

```
finger [-bfhilmpqsw] [username...]

finger [-l]  [username@hostname1[@hostname2...@hostnamen] ...]

finger [-l] [@hostname1[@hostname2...@hostnamen] ...]
```

finger nos sirve para conocer algo más de la persona que está detrás del nombre@dominio, ya que nos da información sobre su nombre de usuario, si esta conectado en ese momento y también si ha leído o no su correo.

Si nosotros queremos saber quién está usando la misma red o máquina que nosotros estamos usando solo es necesario escribir el nombre, es decir, no tenemos que poner todas las palabras que hay en común en nuestras direcciones, si queremos saber sobre todos los usuarios conectados a nuestro sistema de manera local o remota entonces solo se teclea el nombre del comando.

A continuación se muestran algunos ejemplos de cómo se utiliza el comando finger.

Ejemplo 1.
```
%finger
```

Aquí, el comando finger nos muestra a todos los usuarios conectados a nuestro sistema, locales y remotos. Por default, el comando finger despliega múltiples columnas con la información siguiente:

- Nombre de usuario
- Nombre completo del usuario
- Nombre de la terminal asociada

- El tiempo de ocio
- La hora de entrada
- El nombre del host, si es conexión remota

Ejemplo 2.

```
charanda% finger user
```

Nos muestra información sobre un usuario en específico. La información que muestra como default es:

- Nombre de usuario
- Nombre real de usuario
- Directorio home
- El shell asignado
- La fecha y hora en que se conecto
- El nombre de la terminal asociada
- El host name si es conexión remota
- El tiempo de ocio
- La última vez que el usuario recibió correo y lo leyó
- El contenido del archivo $HOME/.plan (plan de usuario)

Finger tiene algunos parámetros cuando se pregunta por un solo usuario, algunas de las más importantes son:

-b No imprime el directorio home y no imprime el shell utilizado.
-s Forza a una salida en formato corto

Ejemplo 3.

```
charanda% finger @mira.labvis.unam.mx
```

Nos muestra información de los usuarios conectados en una determinada máquina. La información que muestra el comando finger al ser utilizado de esta manera es:

- Nombre de usuario
- Nombre real del usuario
- Nombre de la terminal
- Tiempo de ocio
- La hora en que se conectó
- Información adicional del usuario.

3.6.2 COMANDO WHO

El comando *who* es el comando más común para obtener información acerca de que usuarios están conectados al sistema. En UNIX, no es de extrañarse que existan varias personas trabajando en una misma computadora de manera simultánea, incluso, un mismo usuario puede tener varias sesiones abiertas en un determinado momento. Este comando está contenido en todos los sistemas UNIX, es un comando muy simple, se teclea:

```
%who
```

y el sistema da una lista de todos los usuarios que en ese momento están conectados al sistema.

who tiene varias opciones que pueden cambiar drásticamente la manera de observar la salida del comando. Aquí se muestran todas las opciones de este comando.

Opción	Resultado
-b	Imprime la fecha del último rearranque.
-d	Imprime una lista de los procesos muertos que no han sido regenerados.
-H	Imprime las cabeceras de las columnas.
-l	Lista las líneas *tty* en espera de usuario.
-m	Imprime solo información de la terminal actual.
-p	Lista todos los procesos activos recuperados por *init*.
-q	Lista solo los nombres y cuentas de usuarios.
-r	Lista el estado actual de ejecución del sistema.
-s	Lista el *tty* y fecha de entrada del usuario.
-t	Lista la fecha del último cambio de hora del reloj.
-T	Lista el estado de cada terminal (+indica que la línea es escribible, y – que no).
-u	Lista el tiempo de reposo (idle) de cada terminal.

Algunas de estas opciones no están implantadas en LINUX y algunas otras no aparecen en sistemas BSD. Para poder determinar qué opciones son válidas en nuestro sistema, se puede verificar en la página del manual de who.

Existe la opción *am i* para este comando, cuando se ejecuta el comando con este parámetro, se visualizará por pantalla el nombre de conexión (*login*), la terminal asociada y la fecha y hora de inicio de sesión del usuario que mandó a ejecutar el comando.

3.6.3 COMANDO W

El comando *w* nos brinda información de las entradas del sistema, nos informa de lo que están haciendo los usuarios conectados en ese momento al sistema, es decir, sus procesos. Este comando no está presente en todas las versiones UNIX, por lo tanto es necesario comprobar si en nuestro sistema existe consultado la página del manual de *w*.

El comando tiene la siguiente sintaxis:

```
%w [-hsl] [usuario]
```

Cada entrada del listado que suministra el comando *w* tiene siete campos y a continuación se describe cada uno de ellos.

user	Nombre del usuario asociado con el *tty*.
tty	Entrada *tty*.
login@	Fecha de entrada original.
idle	Tiempo transcurrido desde la última entrada del usuario, en minutos.
JCPU	Tiempo total de CPU utilizado por todos los procesos en la terminal.
PCPU	Tiempo total para todos los procesos activos en la terminal.
what	Nombre y argumentos del proceso en curso.

Además, el comando *w* tiene las opciones:

-h	Suprimir encabezados.
-l	Formato largo.
-s	Formato corto.
usuario	Muestra únicamente información relacionada con el usuario indicado.

El formato corto tiene sólo cuatro campos: *user, tty, idle y what.*

3.6.4 COMANDO LAST

El comando *last*, sin argumentos, lista las últimas personas que han entrado al sistema, en orden inverso. Esta lista puede ser muy larga, dependiendo de que tantos usuarios tenga el sistema y que tanto usen los recursos cada uno de ellos.

La sintaxis del comando *last* es:

```
%last [n] [user] [tty]
```

Este comando lista en pantalla la duración de cada sesión, así como la fecha y hora original de cada entrada. Se puede truncar la salida del comando *last* indicándole un número como opción, es decir, solo listará los últimos *n* usuarios que entraron al sistema. Escribiendo un nombre o un *tty*, puede comprobarse la última vez que un usuario entró en el sistema, o quién ha usado un *tty* determinado.

El comando *last* es importante para los administradores de sistemas, ya que con él pueden comprobar quién ha entrado y cuándo. Es importante mencionar que los datos obtenidos por el comando *last* los obtiene del archivo */etc/utmp* que registra las entradas y los rearranques del sistema.

Como se ha mencionado, *last* puede mostrar información acerca de las últimas veces que el sistema ha sido rearrancado, para obtener esta información, se teclea:

```
%last reboot
```

y a continuación se verá una lista de las últimas veces que el sistema ha sido rearrancado.

CAPITULO 4

SISTEMA SUPERVISOR DE CLAVES Y ACCESOS EN UNIX (SCAU)

En este capítulo se describe al Sistema Supervisor de Claves y Accesos en UNIX (SCAU), así como los antecedentes y las necesidades que dieron origen a su desarrollo.

4.1 BREVE DESCRIPCION DEL SISTEMA SUPERVISOR DE CLAVES Y ACCESOS EN UNIX (SCAU)

El sistema Supervisor de claves y accesos en UNIX es una herramienta de administración para revisar los accesos y claves de los usuarios de los sistemas existentes en el departamento de supercómputo. El proyecto se compone de una serie de programas desarrollados en Perl y C, su función general es la de transferir a una máquina central a través de la red todos los accesos de los usuarios, para almacenarlos ,filtrarlos y transformarlos en información gráfica que permita al administrador tener una visión más rápida y general del comportamiento de los usuarios y los accesos generados por ellos.

4.2 ANTECEDENTES

La idea original de este programa nace en el Departamento de Administración de Supercómputo de la Dirección General de Servicios de Cómputo Académico (DGSCA), en la Universidad Nacional Autónoma de México (UNAM).

En este departamento se llevan a cabo labores de administración de varias máquinas con diferentes versiones del UNIX (IRIX, Solaris y OpenBSD), las cuales son utilizadas en proyectos de investigación, además se realiza la administración de los equipos del Laboratorio de Visualización.

El laboratorio de Visualización, cuenta con estaciones de trabajo con excelentes características de manejo de imágenes y graficación, las cuales son necesarias para la obtención visual de los resultados numéricos proporcionados por las supercómputadoras.

Las claves para trabajar en la máquinas del departamento son asignadas dependiendo de las características del usuario; una máquina tiene varios usuarios, que le dan uso continuo. Cada usuario tiene que utilizar su cuenta solo para tareas asignadas, ya que el abusar de los recursos del sistema puede ser superficial al resto de los usuarios, por otro lado algunos usuarios desconocen que debido a un password muy fácil existe el peligro de usuarios extraños al equipo.

Con el análisis de los archivos de accesos (utmp y wtmp) podemos definir un mal uso de la cuenta, pero considerando que solo se almacena los accesos a la máquina y no almacena los accesos a otras máquinas a partir de ella, además de que se tiene un gran número de máquinas surge la necesidad de contar con una herramienta optimizada que ayude al administrador de sistemas a realizar este tipo de tareas.

El Sistema Supervisor de Claves y Accesos en UNIX (SCAU), haciendo uso de las bitácoras de accesos y de otros algoritmos que supervisan las salidas a otras máquinas, facilita la labor y rapidez de analizar los accesos y las claves de un conjunto de máquinas, generando reportes gráficos fáciles de interpretar y tomando automáticamente algunas acciones de prevención.

4.2.1 PROBLEMATICA

Cada máquina genera una bitácora de accesos a ella, la cual se modifica cada vez que un usuario entra al sistema o se conecta a otro lugar , el administrador solo tiene manera de saber los usuarios que accesaron a la máquina pero no tiene la información de sus conexiones apartir de la máquina a otros lugares, el administrador del sistema a su vez realiza la supervisión de los accesos en determinados tiempos y lo hace con comandos los cuales leen las bitácoras de accesos, todo lo anteriormente mencionado genera varios inconvenientes:

- Los comandos que leen la bitácoras, varían su salida de acuerdo al tipo de sistema operativo y en supercómputo se tienen principalmente tres versiones IRIX, Solaris y OpenBSD.
- Las bitácoras pueden estar dañadas y no reportar nada.
- La bitácoras pueden ser modificadas por root pero en la actualidad esto no asegura que la bitácora jamas será modificada por alguien más ya que existen técnicas para convertirse en superusuario y modificarlas.
- El usuario puede conectarse a lugares extraños a partir de la máquina local sin ser monitoreado y hacer mal uso de dicha conexión.
- Las bitácoras no se pueden leer con cualquier editor de texto, se necesita desarrollar un programa que lea el formato del archivo o utilizar los comandos propios del sistema.
- Cada máquina tiene su bitácora, considerando que se cuenta con varias máquinas esto genera una cantidad de ellas que no es posible examinar a detalle, debido a que es una tarea tardada y generalmente el administrador necesita atender lo mas pronto posible los inconvenientes que surjan.

4.2.2 ALTERNATIVA DE SOLUCION

La solución a esta problemática es hacer que se entienda el significado de poder mantener un verdadero control en los accesos y salidas, generando una aplicación capaz de poder supervisar las claves y accesos, desarrollando para esto una representación gráfica sencilla de asimilar, tangible para el administrador y mediante el cual se resuelvan rápidamente situaciones de mal uso de una cuenta, de un conjunto de máquinas.

4.3 DESCRIPCION GENERAL

El SCAU es un sistema que facilita la búsqueda de información relevante de los accesos y claves, resolviendo los problemas antes mencionados. El SCAU está desarrollado con base en el modelo cliente/servidor, donde el cliente solicita al servidor el permiso para ingresar sus accesos y salidas y este atiende la petición dándole permiso y guardando la información en archivos correspondientes a cada cliente (figura 4.1), creando las bitácoras de accesos y de salidas, así como reportes tanto gráficos y de texto.

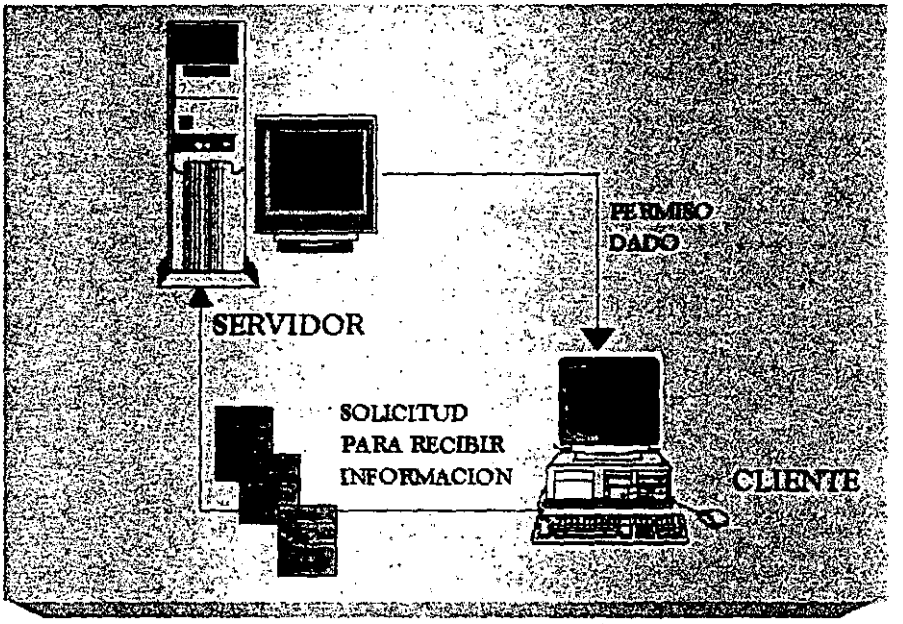

Figura 4.1 Solicitud del cliente al servidor

4.3.1 SERVIDOR-SCAU

El servidor . resuelve peticiones de los clientes. Atento a la petición por el puerto asignado para esta tarea, se encuentra un programa que resuelve si la petición es válida y envia la respuesta de la solucitud. Este programa siempre estará listo para el número de peticiones que lleguen, ya que por cada petición ejecutará un proceso nuevo para atender la misma y al mismo tiempo el programa seguirá atento a que llegue una nueva petición, aprovechando de esta manera las características del multiprocesamiento que tiene UNIX (figura 4.2).

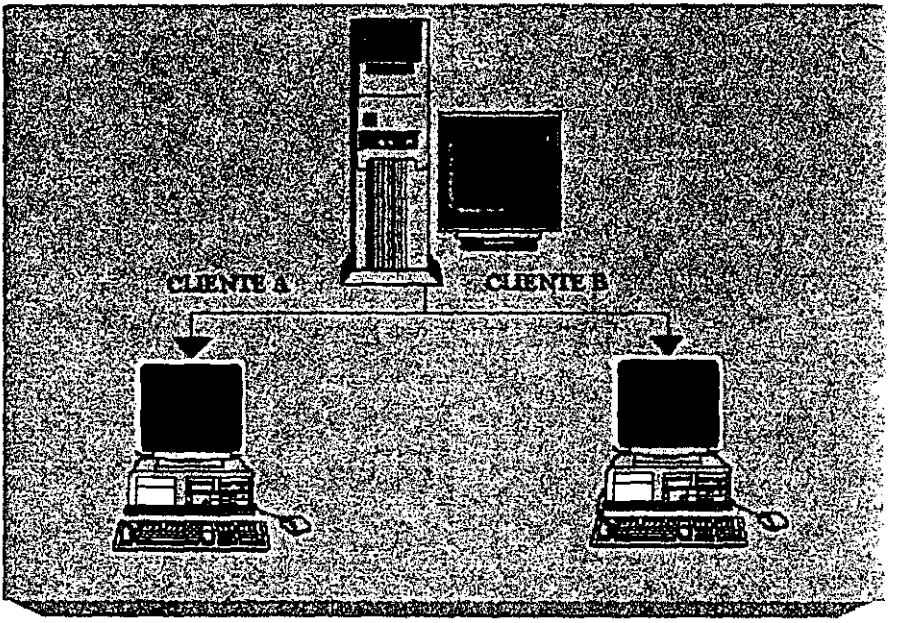

Figura 4.2 Atención a más de un cliente al mismo tiempo

Consta de una interfaz gráfica, de las llamadas interfaces orientadas a ventanas con opción a señalar y elegir[11]. "La interfaz de usuario es el mecanismo a tráves del cual se establece un diálogo entre el programa y el humano"[Pressman 93]. Aquí podemos analizar cada una de las bitácoras de los clientes, realizando un filtrado de información de los mismos para finalmente obtener la información requerida y posteriormente convertirla en imágenes (gráficas resultantes de tipo histograma o barra) que nos representan los accesos y salidas de los sistemas.

La cuarta generación de interfaces gráficas[Pressman 93] es aprovechada para trabajar con el SCAU ya que tiene la habilidad para realizar difentes tareas simultáneas y es posible realizar en otras ventanas otras tareas rutinarias(figura 4.3).

[11] También denominadas "WIMP" (del inglés, ventanas, iconos, menús y dispositivos de señalización)

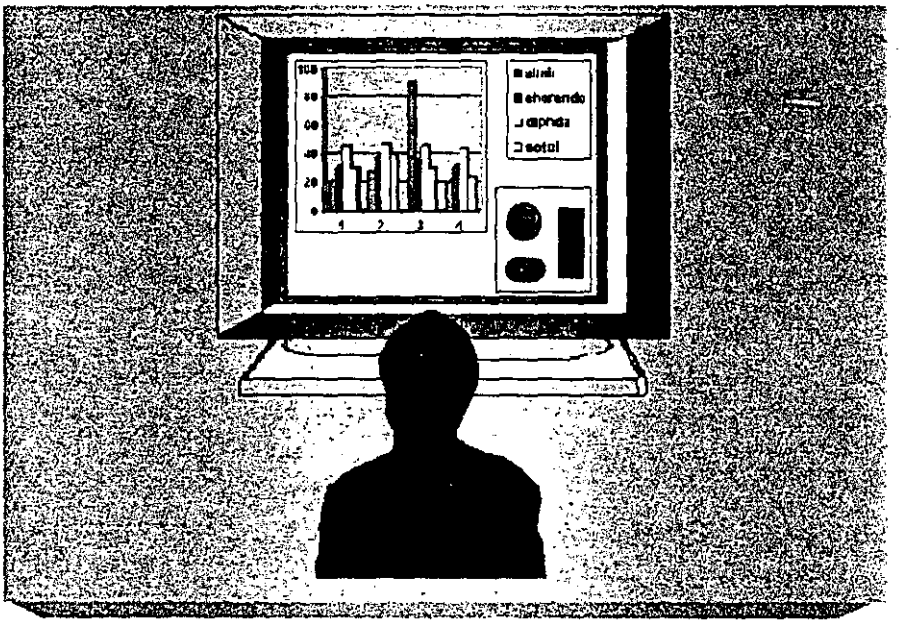

Figura 4.3 Cuarta generación de interfaces gráficas

4.3.2 CLIENTE-SCAU

La parte del cliente del SCAU, es un programa que siempre estará atento a los accesos y salidas de cada sistema. Cada vez que el archivo de accesos del sistema es modificado o el usuario sale a otro sistema a partir de esa máquina (salidas), el cliente lee la información posteriormente hará la petición al servidor, una vez que recibe la respuesta válida mandará su información al servidor y este la guardará en la bitácora correspondiente a cada cliente.

4.4 VENTAJAS DE SCAU

- Se generan bitácoras más fáciles de entender tanto de accesos como de salidas a otras máquinas, por cada sistema involucrado.
- Se tiene una visión más detallada de todas las máquinas y el comportamiento de sus usuarios, no importando la versión de UNIX.
- Desde una sola máquina podemos observar detalladamente a las demás, así como el estado de las claves de cada una de ellas.

- La información de los accesos se toma inmediatamente cuando accesa el usuario, y se manda al servidor, garantizando que la información no pueda ser modificada o dañada.
- Las máquinas proporcionan un servicio exclusivo para transferir los accesos y salidas sin necesidad de tener una conexión directa a estas máquinas, es decir, no necesitamos entrar a estas máquinas para revisar las bitácoras de accesos.
- Al evitar la necesidad de entrar al sistema, evita también que viaje demasiado el password del usuario que esté revisando las bitácoras a través de la red.

CAPITULO 5

DISEÑO E IMPLEMENTACION DE SCAU

En este último capítulo es donde se presenta a detalle la manera en que se diseñó e implementó el Sistema Supervisor de Claves y Accesos en UNIX (SCAU), aquí se mostrarán los programas fuente correspondientes a los módulos de SCAU, se explica cada uno de ellos de manera rápida pero resaltando los puntos más importantes desde nuestro punto de vista.

5.1 DISEÑO ORIENTADO A FLUJO DE DATOS

El diseño como fase del desarrollo de un sistema, es de gran importancia y algunas personas lo definen como: "El proceso de aplicar distintas técnicas y principios con el propósito de definir un dispositivo, proceso o sistema con los suficientes detalles como para permitir su realización física".

Para responder a las preguntas importantes en el desarrollo del sistema como: ¿Qué procesos integran el sistema?, ¿Qué datos se utilizan en cada proceso?, ¿Qué datos se almacenan? y ¿Qué datos entran y salen al sistema?, es necesario realizar un análisis de datos del mismo.

Utilizando un modelo lógico de diagramas de flujo como una herramienta para estructurar el diseño de un sistema desde el comienzo del mismo, significa que se podrán utilizar las características del flujo de la información para generar la estructura del programa. La información puede representarse como un flujo continuo que sufre una serie de transformaciones conforme se va moviendo a través del sistema de la entrada a la salida.

5.1.1 DIAGRAMA DE FLUJO DE DATOS

El análisis del flujo de datos examina el uso de éstos para llevar a cabo procesos específicos dentro del objetivo de una investigación del sistema. Los diagramas de flujos de datos (DFD) muestran su uso en el sistema en forma gráfica, es decir, todos los componentes esenciales del sistema y como se relacionan entre sí.

Existe una notación dentro de los diagramas de flujo de datos que es sencilla y a la vez fácil de manejar.

- Flujo de datos. Los datos cambian en una dirección específica, desde su origen hasta su destino, y para representarlos se utilizan las flechas.
- Procesos. Los datos se transforman a través de procesos, se identifican con un círculo.
- Datos almacenados. De alguna manera los datos deberán ser almacenados para poder usarlos en algún otro proceso, estos se identifican con las líneas dobles.

Para el diseño de SCAU se ha desarrollado el diagrama de flujo en cuatro diferentes niveles de abstracción que a continuación se presentan.

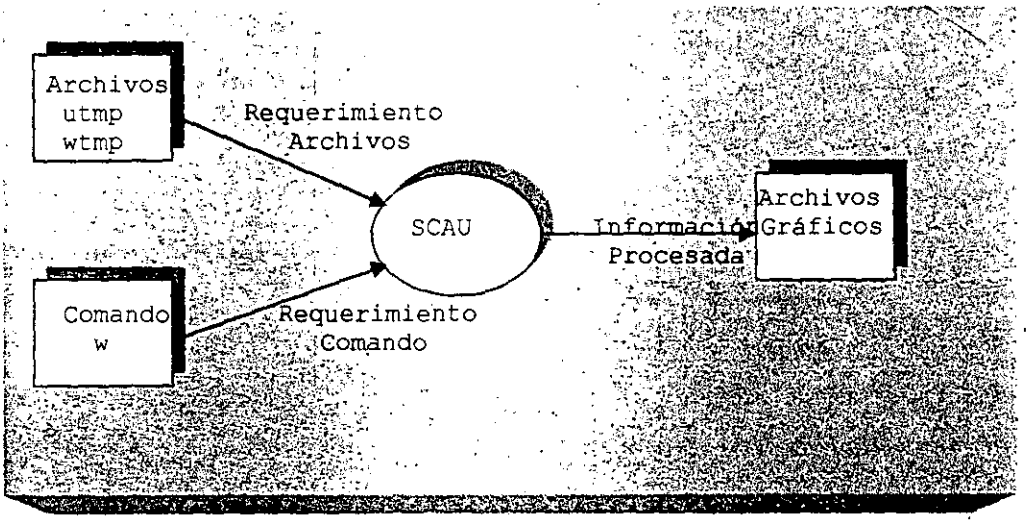

Figura 5.1 Diagrama de Flujo de Datos Nivel 0

La figura 5.1 representa el Diagrama de Flujo de Datos de Nivel 0 siendo éste el modelo fundamental del sistema. Como se puede observar, existen tres entidades que son las fundamentales: los archivos *utmp* y *wtmp*, el comando *w* y los archivos de salida. Los archivos utmp y wtmp producen información acerca de los usuarios que entran al sistema monitoreado por SCAU, el comando w produce información acerca de los usuarios que salen a sitios remotos desde el sistema monitoreado por SCAU y por último tenemos a los archivos gráficos, que es la información de salida que SCAU ha generado a partir de las dos primeras entidades. Las flechas que indican el flujo de los datos nos muestran las órdenes principales del sistema: requerir y transformar.

La figura 5.2 muestra el DFD de Nivel 1, éste es el resultado de la expansión del DFD de nivel 0 para generar más información a partir de este nivel. Ahora como se puede observar, este nivel muestra trece procesos pero se puede observar la continuidad del flujo de la información. En este diagrama podemos observar como se generan los nuevos procesos necesarios para explicar lo que el sistema esta realizando. SCAU trata las dos entradas al sistema de manera similar y esto se puede observar en el diagrama, solo que la información proviene de dos diferentes entidades. Los procesos que en este nivel se han creado muestran donde llegan los datos por los diferentes flujos dependiendo de peticiones o validaciones del sistema.

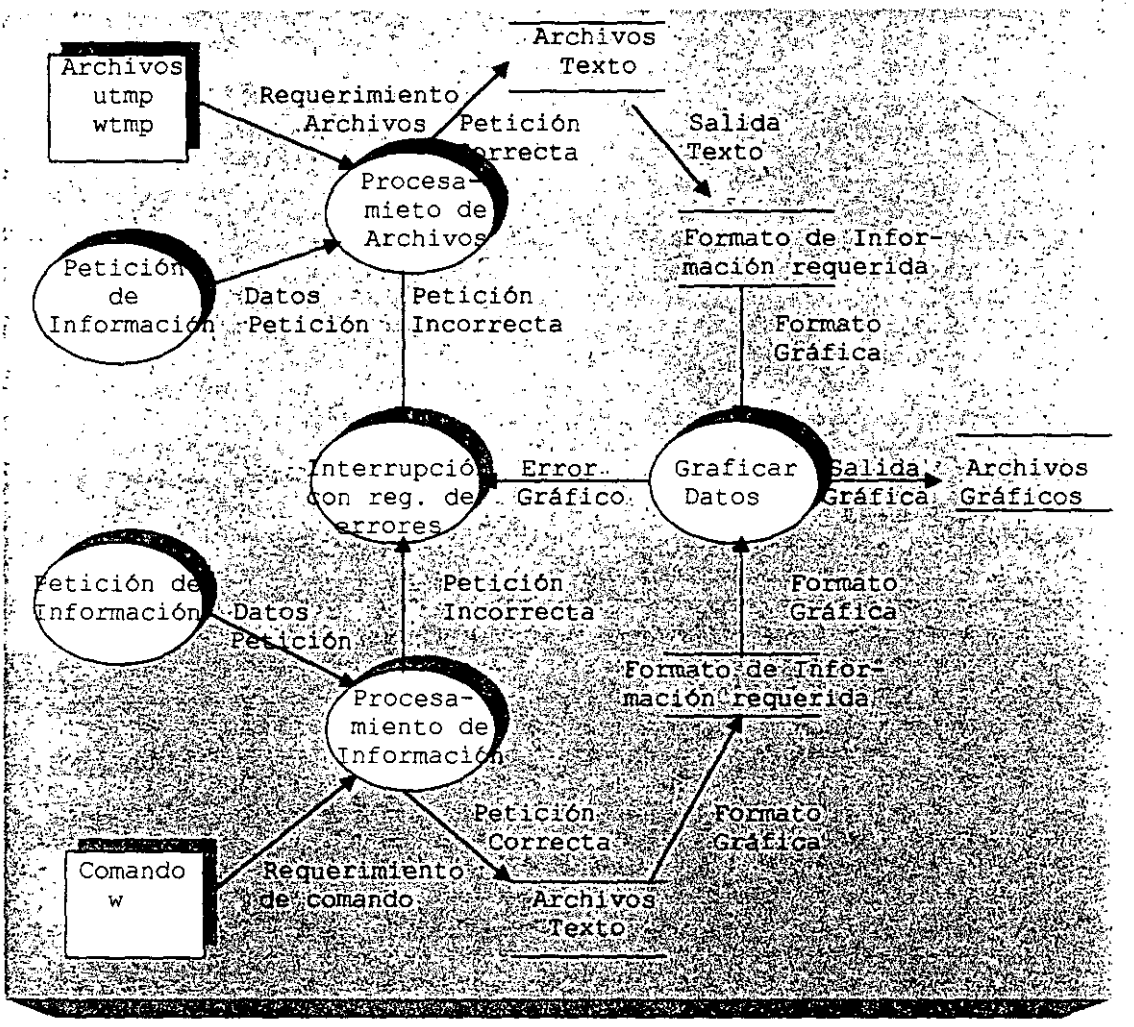

Figura 5.2 Diagrama de Flujo de Datos Nivel 1.

Expandiendo el DFD de nivel 1 a un nivel 2 (figura 5.3), podemos observar como se generan nuevos subprocesos, éstos muestran gran detalle el flujo de datos donde se validan las peticiones, si existen errores en las peticiones se genera un nuevo subproceso que clasifica que cual fue el tipo de error para que el usuario pueda o trate de corregirlo; en caso de que la validación sea correcta se almacenan los datos. Estos procesos son para las dos entradas principales la sistema y se manejan de manera similar.

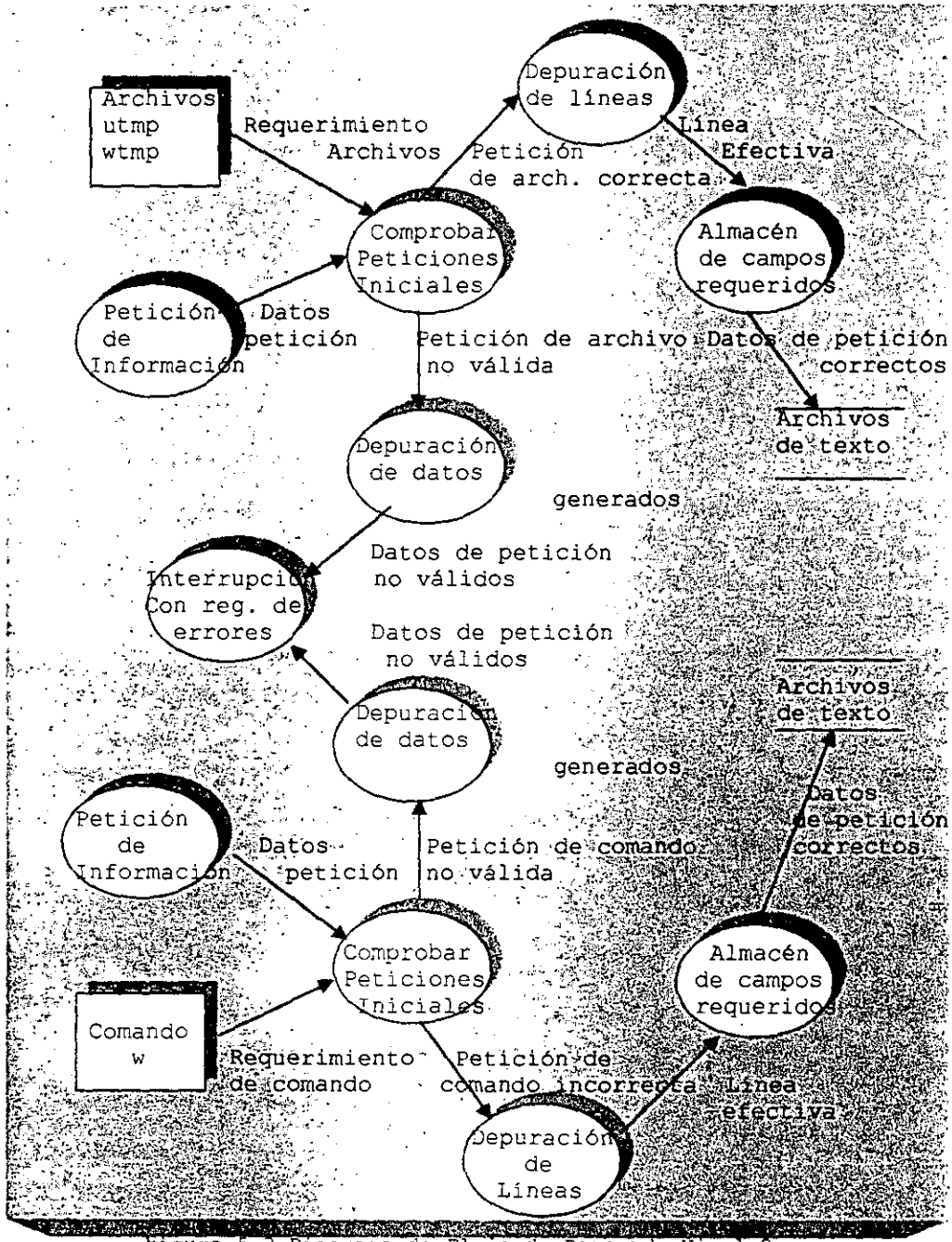

Figura 5.3 Diagrama de Flujo de Datos de Nivel 2.

Por último se muestra el DFD de Nivel 3 (figura 5.4) donde se muestra la generación de nuevos subprocesos para observar con detalle el flujo de datos donde se depura y se determina el tipo de gráfica a realizar por SCAU, en caso de error al igual que en el DFD de nivel 2, se genera un subproceso para clasificar el tipo de error que ha ocurrido. Este diagrama representa las dos entradas principales del sistema y cada parte hacer exactamente lo mismo, por lo cual solo se muestra una de las dos partes que componen este diagrama.

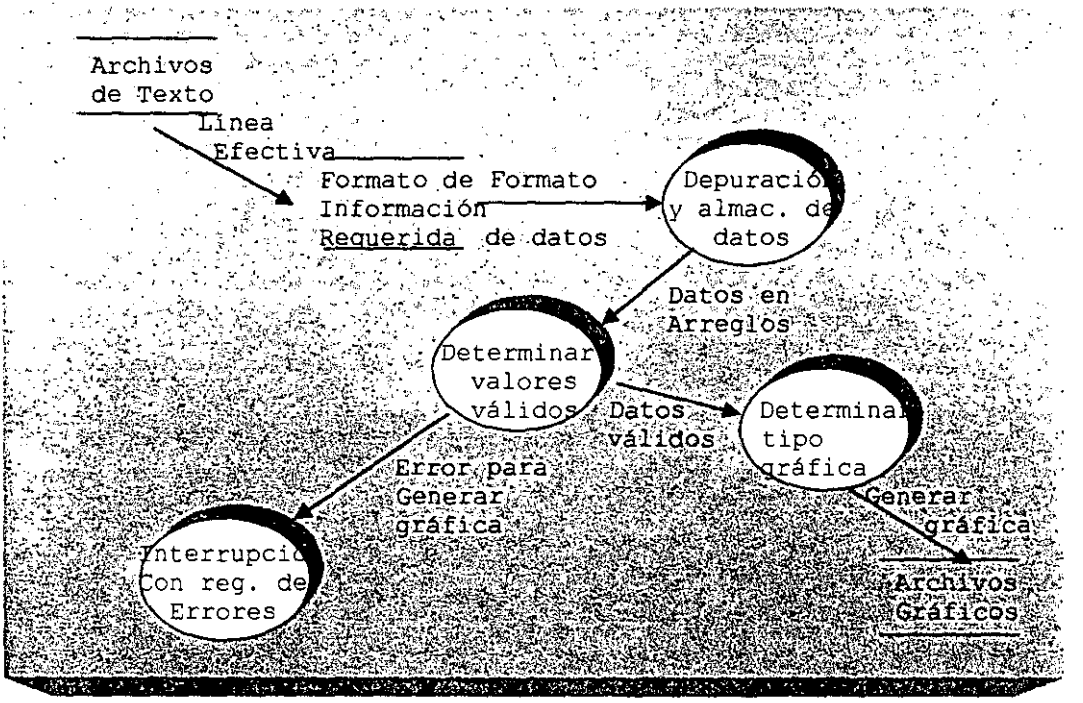

Figura 5.4 Diagrama de Flujo de Datos de Nivel 3.

5.2 ELECCION DEL LENGUAJE DE PROGRAMACION

"El arte de elegir un lenguaje comienza con el propio problema planteado, al decidir cuáles son sus requisitos y su importancia relativa, ya que probablemente será imposible satisfacerlos a todos por igual (con un único lenguaje)" [Pressman93].

Entre los criterios que se aplican durante la evaluación son:

- Area de aplicación general
- Complejidad algorítmica
- Entorno en que se ejecutará el software
- Complejidad de las estructuras de datos

La elección del lenguaje de programación en que se codificó SCAU tomando en cuenta los anteriores puntos fue de complicada, la elección de un solo lenguaje que satisfaga nuestros requisitos como programadores era difícil, de manera que el problema se soluciona con la utilización de dos lenguajes compatibles entre si y con el sistema propio. El lenguaje C es básico, necesario para el manejo de las estructuras de los archivos *utmp* y *wtmp* del sistema operativo UNIX y cumple con los requisitos anteriores. Perl, otro lenguaje de programación elegido que cumple con los requisitos necesarios, es decir, adecuado en el entorno de ejecución (sistemas UNIX), área de aplicación (administración de estaciones de trabajo), facilita el manejo de estructuras de datos e incluye una extensión para el manejo de interfaces gráficas en la interpretación de los datos de nuestro sistema.

5.2.1 CARACTERISTICAS DEL LENGUAJE C

El lenguaje C se definió en 1972 en los laboratorios Bell Telephone por Dennis Ritchie y ha seguido una evolución paralela a la del sistema operativo UNIX [12]. Se puede decir que C es un lenguaje de programación para sistemas o un lenguaje adaptado al desarrollo de programas de base. Esta afirmación se certifica por la propia existencia de UNIX y de toda la gama de productos que constituyen el paquete UNIX, pero también, el lenguaje C se practica cada vez más para desarrollar aplicaciones científicas de alto nivel, gestiones de bases de datos relacionales, comunicación entre computadoras, etc. Una de las mayores ventajas es la permitir la utilización natural de las funciones primitivas del sistema de explotación (UNIX en este caso) mediante apelación directa a ellas. Los principales puntos que constituyen la originalidad del lenguaje C son: su variado juego de operadores, su permisividad en el tratamiento de los diversos tipos de datos y su eficacia en cuanto a la producción de código objeto altamente optimizado. El compilador por si mismo, ocupa muy poco espacio en memoria y es fácilmente transportable de una máquina a otra o de un sistema a otro.

[12] Los principales programas del sistema operativo UNIX están escritos en lenguaje C.

5.2.2 CARACTERISTICAS DE PERL

Este lenguaje fue creado por Larry Wall en 1986. Programando en *shell* y *awk* [13], Wall se dio cuenta que era muy difícil generar reportes a partir de muchos archivos de texto con referencias cruzadas, entonces decide crear código en lenguaje C que enfatizara sobre la creación de reportes y la administración del sistema. Larry observó que en la mayoría de los programas necesarios para el mantenimiento del sistema, el punto primordial es el procesamiento y manejo de texto, ya sea para la creación de reportes, el consolidado de información o la clasificación de archivos de datos. Las características más importantes son: el procesamiento y manipulación de información como cadenas de caracteres, el manejo de archivos y la generación de reportes de texto, algunas llamadas al sistema UNIX, el manejo de expresiones regulares, señales y también sockets de red.

5.3 MODULOS DE COMUNICACION SCAU

Nuestro sistema cliente/servidor requiere de tres elementos fundamentales para su funcionamiento: el servidor, el cliente y la comunicación entre ellos. A continuación se presentan los módulos SCAU del servidor y clientes, de manera clara se tratará de explicar los programas fuentes que cliente y en el servidor utilizan para realizar sus tareas.

5.3.1 MODULO CONTROLADOR DEL SERVIDOR

El modelo cliente/servidor en el que SCAU opera, el programa residente en una sola máquina (la de más capacidad) que actúa como servidor, controla las peticiones hechas a éste. El servidor se encarga de atender a todas las máquinas cliente para realizar una conexión remota a través de la red. La respuesta positiva del servidor hace que se transfiera una línea de datos entre el servidor y el cliente. El servidor hace una copia de esta línea en una de las diferentes bitácoras que él opera (Ver apéndice para observar el programa fuente del servidor).

Las bitácoras creadas por el programa servidor son guardadas en disco duro. Cada línea que el servidor recibe a través de un socket es almacenada en una bitácora que lleva un prefijo y el nombre del cliente. Los prefijos son *be* para las bitácoras de entradas y *bs* para las bitácoras de salidas. Todas las bitácoras son almacenadas en subdirectorios con el nombre del mes en que se hayan registrado, así por ejemplo, encontraremos una bitácora llamada *Feb/becharanda* o *Mar/bscharanda*, donde charanda es el nombre del cliente. Las bitácoras contienen la información más importante de SCAU, ya que de estas dependen las gráficas mostradas y los avisos que vía e-mail realiza el sistema al administrador.

[13] Filtros comunes en UNIX

Llamada al sistema fork()

La llamada al sistema *fork()*[14] en UNIX permite la creación de un nuevo proceso que es una copia casi exacta del primero, las diferencias fundamentales son:

- El proceso hijo tiene un identificador de proceso único.
- El proceso hijo tiene un identificador de proceso padre diferente.
- El proceso hijo tiene su propia copia de los descriptores del proceso padre.
- La función *fork()* regresa 0 al hijo y al padre le regresa el identificador del proceso hijo.
- La llamada al sistema *wait()* esta ligada al *fork()* y generalmente se utiliza para hacer que el proceso padre espere por la terminación del proceso hijo.

Funciones Asociadas a un Servidor que Trabaja con Sockets

- *socket()* crea un socket genérico.
- *bind()* relaciona servidor con el socket a un puerto.
- *listen()* notifica al sistema que está listo para la conexión.
- *accept()* espera por la primera conexión.
- El servidor es notificado cuando se realiza una conexión. Típicamente el servidor realiza un fork() para que el servidor hijo maneje la conexión.
- La conexión hijo lee los datos del socket que han sido enviados por el cliente.
- El servidor padre sigue esperando más conexiones. [Schwartz94]

Aquí se muestra una figura con los fork() creados por el servidor para su comunicación con los clientes:

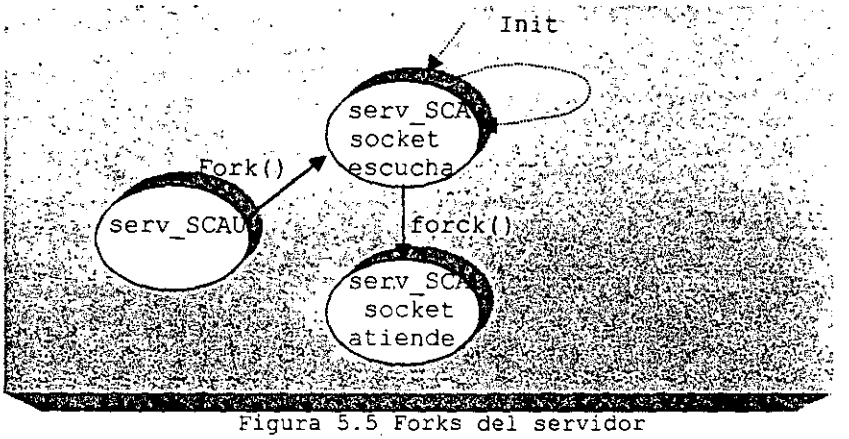

Figura 5.5 Forks del servidor

[14] Página de manual de UNIX fork(): man fork

5.3.2 MODULOS CONTROLADORES DEL CLIENTE

En SCAU, cada máquina cliente cuenta con un programa ejecutable C y dos más hechos en Perl. El primero (*lee_wtmp*) es el que realiza la lectura de manera directa al archivo wtmp o wtmpx (dependiendo del sistema operativo), el programa *ent_SCAU.pl* realizado en Perl es un *demonio* que monitorea la fecha de actualización del archivo wtmp para obtener información acerca del más reciente acceso de los usuarios al sistema y mandar esta información hacia el servidor, el último programa llamado *sal_SCAU.pl* también realizado en Perl es otro demonio que ejecuta el comando w en forma indefinida para observar y filtrar las conexiones que realizan los usuarios desde la máquina monitoreada y mandar estos datos al servidor para ser almacenados (Ver apéndice para observar los programas fuente del servidor).

Funciones Asociadas a un Cliente que Trabaja con Sockets

- *socket()* crea un socket genérico.
- *bind()* relaciona al cliente con la dirección del servidor.
- *connect()* una vez ligado, el cliente conecta su socket al socket del servidor. Típicamente el cliente realiza un *fork()* para que el proceso hijo lea el socket y le proceso padre escriba al socket.

Aquí se muestra una figura con los forks() creados por cada cliente para su comunicación con el servidor:

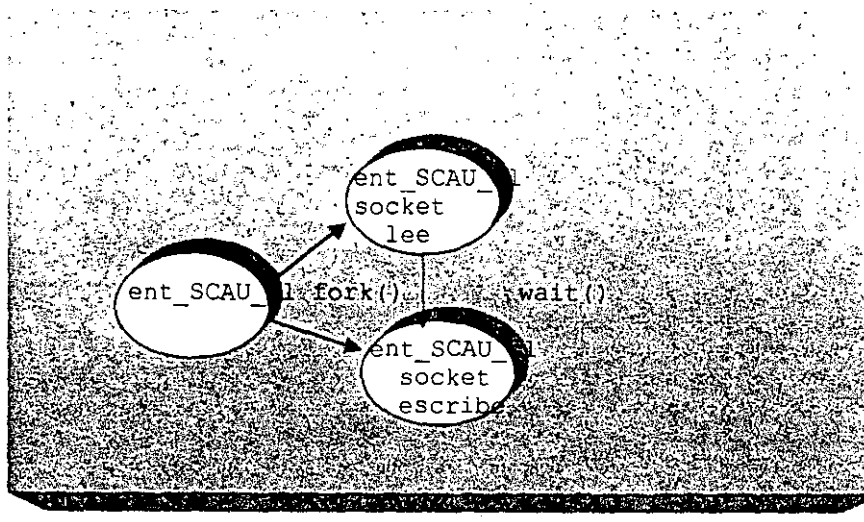

figura 5.6 Forks del cliente

5.4 INTERFAZ GRAFICA

El módulo Crea_SCAU_Interfaz.pl filtra la respectiva información para cada reporte solicitado en el rango de las bitácoras de entradas y salidas de las máquinas, trabajando uno o varios archivos a la vez por cada solicitud. Posteriormente se grafican los datos obtenidos dependiendo del tipo de gráfica (barra o histograma) que se solicita.

Este módulo nos permite la creación de la interfaz gráfica de usuario permite, de una manera fácil, la comunicación y manipulación de todos los módulos involucrados para el funcionamiento del SCAU. El usuario manipula en un ambiente de ventanas todas las ordenes disponibles a su elección.

5.4.1 Perl/Tk

Las plataformas de ventanas (Apple Mac, X window y Microsoft Windows), proporcionan un nivel de programación para crear y manejar ventanas, para reportar eventos diferentes al mouse y el teclado y para dibujar elementos gráficos como líneas o círculos. El problema es que dibujar desde la forma más simple toma una cantidad considerable de código y se tienen que leer cientos de páginas de la documentación. Los patrones mas usados de código para interfaces de usuarios han sido definidos dentro de los llamados widgets (llamados "controles" en el mundo Microsof). Ejemplos son los **buttons, scrollbars, listbox.**

Para la mayoría de los sistemas UNIX, **X window** es la plataforma de ventanas. Varios conjuntos de herramientas de widgets se han construido sobre X: Athena, InterViews, Motif y Tk. La ventaja de Tk es que no necesita C o C++ para manipular los widgets, se construyen intefaces rápidamente y es de distribución gratuita.

Tk fue desarrollado para ser manejado por el lenguaje Tcl[15]. Malcolm Beattie hizo un primer intento para proporcionar una capa de Perl que internamente utilizará Tcl para utilizar las librerías Tk.

Nick Ing-Simmons quitó de Tk todo el código Tcl y generó una capa portable para anadir Tk a otros lenguajes; este esfuerzo es llamado *ptk* (Tk portable), también añadió una extensión a Perl, de esta combinación de ptk y Perl surgió el módulo Tk.pm[Srinivasan 97], el cual se utilizó para crear la interfaz gráfica de usuario del SCAU (figura 5.7).

[15] Tcl y Tk fuerón desarrollados por el Dr. John Ousterhout

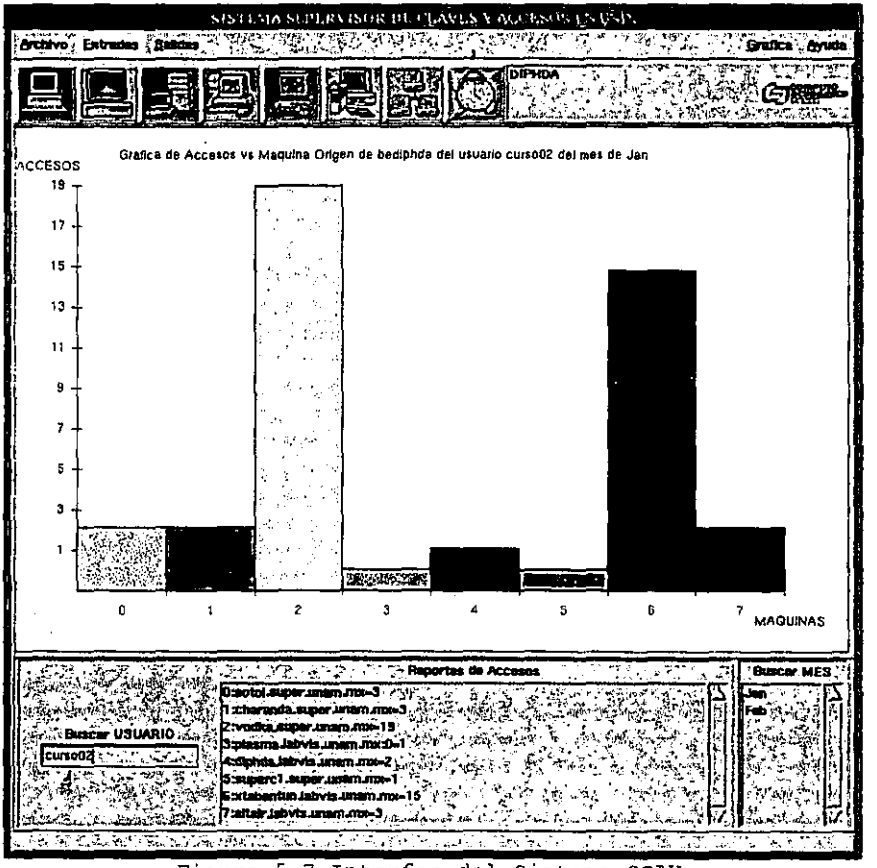

Figura 5.7 Interfaz del Sistema SCAU

5.4.2 AYUDA

El SCAU ofrece facilidades de ayuda interactiva que permite al usuario obtener una respuesta sin abandonar la interfaz. El tipo de ayuda es llamada añadida es decir, es incorporada al sistema despues de haber sido terminado (Ver apéndice para observar los programas fuente de la ayuda).

La incorporación de la ayuda es de gran utilidad, es una ventana separada fácil de usar, basta manipular un puntero de mouse para señalar el título refernte al concepto que se requiera. (figura 5.8).

Los mensajes de error del SCAU interactuan con el usuario y suministran los títulos de ayuda, donde se encuentra información referente.

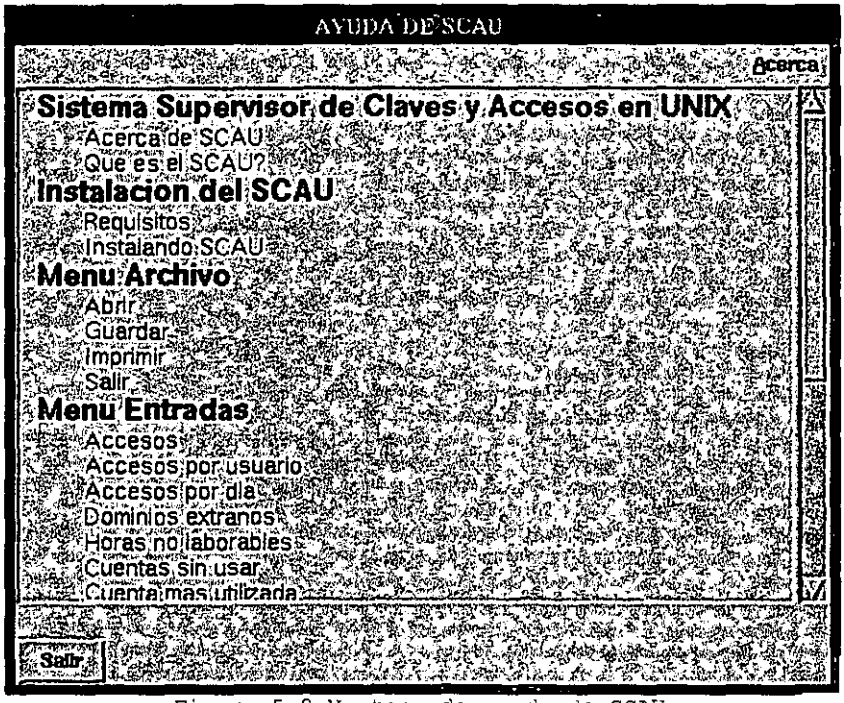

Figura 5.8 Ventana de ayuda de SCAU

5.5 INSTRUMENTACION DE SCAU

Se describe la operación de SCAU en el Departamento de Supercómputo y Laboratorio de Visualización de la Dirección General de Servicios de Cómputo Académico en la Universidad Nacional Autónoma de México.

5.5.1 MAQUINAS INVOLUCRADAS

Las máquinas involucradas que SCAU analizará en operación normal serán todas las máquinas el Departamento de Supercómputo y Laboratorio de Visualización, para efectos prácticos solo se han incluido 6 máquinas.

- **ALTAIR**, altair.labvis.unam.mx, Sun Sparc Classic del Laboratorio de Visualización
- **CHARANDA**, charanda.super.unam.mx, Sun Sparc IV del Departamento de Supercómputo
- **DIPHDA**, diphda.labvis.unam.mx, Sun Sparc Server 1000 del Laboratorio de Visualización

- **SOTOL**, sotol.super.unam.mx, Sun Sparc Classic del Departamento de Supercómputo
- **VODKA**, vodka.super.unam.mx, Sun Sparc Classic del Departamento de Supercómputo
- **XTABENTUN**, xtabentun.super.unam.mx, Sun Sparc Classic del Departamento de Supercómputo

5.5.2 SISTEMAS OPERATIVOS

Los sistemas operativos son un factor muy importante para la operación del SCAU, de ellos depende su correcto funcionamiento. Se mencionan los instalados en las máquinas involucradas:

- **Solaris 2.5**, diphda.labvis.unam.mx
- **OpenBSD 2.5**, altair.labvis.unam.mx
- **OpenBSD 2.5**, sotol.super.unam.mx
- **OpenBSD 2.5**, vodka.super.unam.mx
- **OpenBSD 2.5**, xtabentun.super.unam.mx
- **Solaris 2.7**, charanda.super.unam.mx

Estos sistemas se apegan a System V y BSD. Lo importante para SCAU es la estructura del archivo wtmp y de la salida del comando w.

5.5.3 CONFIGURACION DEL SERVIDOR

Para configurar el servidor es necesario en primer lugar haber obtenido el archivo SCAU.tar que contiene tanto la parte del servidor como la del cliente en archivos *.tar*[16]. Para instalar y configurar el servidor SCAU se debe seguir los siguientes pasos:

- Contar con una versión de Perl 5.0 o mayor.
- Contar con una versión de Tk 8.0 o mayor
- Desempaquetar el archivo SCAU.tar en un subdirectorio con el mismo nombre (SCAU).
- Desempaquetar el archivo SERV.tar que contiene los archivos SCAU_SERV_SOLARIS.tar y SCAU_SERV_IRIX.tar
- Elegir uno de los dos dependiendo de que sistema que se trate y desempaquetarlo.
- En todos los archivos con extensión *.pl* habrá que verificar la ruta del perl en la primera línea.
- Ejecutar el archivo crea_meses.pl con lo que se crearán subdirectorios con los nombres de los meses del año.
- Ejecutar el archivo serv_SCAU.pl.

[16] Extensión de los archivos comprimidos con la utileria tar de UNIX.

Nota. Si existe algún problema con el puerto de comunicación, habrá que editar el archivo y asignar otro puerto en la variable *"my $port=shift||"*. El puerto sirve para comunicar el servidor con el cliente, deberá ser mayor a 1000 ya que los anteriores están reservados para el sistema.

- Crear el directorio TEX dentro de SCAU para que el sistema introduzca aquí los archivos creados por el usuario.
- Ejecutar el archivo ent_SCAU_sr.pl
- Ejecutar el archivo sal_SCAU_sr.pl
- Realizar una copia del archivo /etc/passwd del servidor y cada cliente en el directorio de SCAU.
- Realizar una copia del archivo /etc/hosts del servidor al directorio de SCAU, esto con la intensión de reconocer los dominios extraños en accesos y conexiones remotas al sistema.
- Editar el archivo crea_SCAU_interfaz.pl para configurar la interfaz gráfica.

Aquí debemos realizar varios cambios para su correcta configuración:

1. Definir las variables iniciales:
   ```
   $archivoen="bediphda"; #Variable que valida la máquina en los accesos (diphda es el nombre de un cliente o el servidor mismo y será el default)
   $archivosal="bsdiphda"; #Variable que valida la máquina en las salidas
   $mes="Jan"; #Variable que define el mes a graficar (Poner el mes en que el sistema se está instalando)
   $nusuario="curso02"; #Válida el usuario a buscar (usuario valido en la máquina default)
   ```
2. Crear los botones de las máquinas:
   Ir a la línea:
   `#######Creando Botones de las máquinas#######`
   En la línea "-text =>'ALTAIR'" podemos poner el nombre de un cliente y por cada cliente que se quiera añadir se copian las 6 líneas, poner el nombre del cliente y el nombre de la función (recomendable el mismo nombre)
3. En la línea:
   `#####Balloons de ayuda#####`
   Poner los nombres de los clientes (el servidor también)

4. En la línea:
   `####Funciones de validación de salidas y entradas ####`
   Poner las subrutinas de validación de entradas y salidas con el nombre de los clientes.

NOTA. Al igual que se pueden añadir clientes se pueden eliminar. En las mismas líneas, quitar los elementos correspondientes.

Pruebas con el Servidor.

Las pruebas recomendadas para verificar si el servidor ha quedado instalado son ver que el programa quede como un proceso dentro del sistema.

```
%ps -fea|grep serv_SCAU.pl
```

Esta instrucción debe mostrar información acerca del proceso asociado a serv_SCAU.pl.

```
curso02 2741 1 0 Mar 13 0:07 /usr/local/bin/perl serv_SCAU.pl
```

Otra prueba que se puede realizar es referente al puerto de comunicación que serv_SCAU.pl esta utilizando y que ningún otro servicio puede utilizar, de manera que si tratamos de ejecutar nuevamente el programa habrá un mensaje en pantalla.

```
%serv_SCAU.pl
No puedo preparar el puerto 10001! at serv_SCAU.pl line 28.
```

Una vez superado estas dos pruebas podemos confiar en que el servidor esta listo para operar. Por otro lado, la interfaz gráfica podría crear errores si no hay elementos que analizar dentro del mes default, por lo tanto antes de ver gráficos habrá que instalar los clientes.

Una vez instalado los clientes, en el subdirectorio del mes actual aparecerán conforme haya conexiones al servidor o a los clientes las bitácoras de entrada (becliente) y las bitácoras de salida(bscliente). Entonces podremos ejecutar crea_SCAU_interfaz.pl sin ningún problema.

Con lo anterior el servidor estará listo para recibir peticiones de conexión, guardar la información correspondiente de cada cliente y observar gráficamente el sistema.

5.5.4 CONFIGURACION DEL CLIENTE

Para configurar los clientes en cada uno de ellos se deben llevar a cabo los siguientes pasos:

- Contar con una versión de Perl 5.0 o mayor
- Desempaquetar el archivo SCAU.tar en un subdirectorio con el mismo nombre (SCAU).
- Desempaquetar el archivo CLIENTE.tar que contiene los archivos SCAU_CLIENTE_OPENBSD.tar, SCAU_CLIENTE_SOLARIS.tar y SCAU_CLIETE_IRIX.tar.
- Elegir el adecuado y desempaquetarlo.
- Verificar la ruta de perl en los dos archivos con extensión *.pl*.
- Editar el archivo ent_SCAU_cl.pl

- Asignar el mismo puerto de comunicaciones que en el servidor en la variable *$port*.
- Asignar el nombre con todo y dominio en la variable *$server*.
- Ejecutar el archivo ent_SCAU_cl.pl
- Editar el archivo sal_SCAU_cl.pl.
- Realizar los pasos anteriores en las variables *$port* y *$server*.
- Ejecutar el archivo sal_SCAU_cl.pl

Pruebas con el cliente

Para verificar que el cliente ha quedado instalado verificar los procesos creados por la ejecución de los dos anteriores programas.

```
%ps -fea | grep perl

scau 6004 1 1  /opt/TWWfsw/perl5005/bin perl ent_SCAU_cl.pl
scau 5814 1 2  /opt/TWWfsw/perl5005/bin perl sal_SCAU_cl.pl
```

Otra forma de verificar que los clientes están funcionando correctamente es realizar conexiones remotas a los clientes hacia y desde, después ir al directorio del mes actual en el servidor y revisar que se han registrado las salidas y entradas de los clientes.

NOTA. Para evitar problemas de espacio en disco duro, se pueden borrar todos los archivos *.tar* tanto en los clientes como en el servidor.

Permisos en los Archivos del Servidor y Clientes.

SCAU es un sistema que no requiere de ser el superusuario en los clientes o en el servidor para ser ejecutado, tener una cuenta como usuario normal es suficiente. Para evitar que algún otro usuario realice cambios en la configuración o dañe de alguna manera el sistema, es necesario que todos los archivos de SCAU tengan permisos solo para el dueño (700).

CONCLUSIONES

La administración de sistemas es una de las tareas más importantes para obtener el mayor beneficio de los equipos de cómputo. Administrar de manera correcta equipos UNIX en especial, es de considerarse en la actualidad como una de las más importantes actividades dentro del ámbito de redes de computadoras, ya que una inmensa cantidad de equipos conectados en red cuenta con este tipo de sistema operativo. En la mayoría de los casos, las redes grandes y pequeñas se encuentran conectadas entre sí a través de Internet y por lo tanto conectadas a casi todas partes del mundo, esto significa que cualquier persona con los conocimientos necesarios y desde cualquier lugar puede tener acceso de manera ilegal a nuestros equipos conectados a la red. Es aquí donde entra el tema de seguridad en cómputo que trata de evitar estos accesos y proteger de la mejor manera la información en nuestros equipos.

Administración y seguridad en cómputo, son dos tareas ligadas, inseparables, tienen la finalidad de aprovechar de manera óptima los recursos con que cuenta nuestro sistema y proteger la información de los usuarios. Dentro de las actividades del administrador de sistemas, una de las más importantes para el resguardo de la información es la revisión de las bitácoras de acceso que el sistema genera. Esta tarea puede ser llevada a cabo de manera manual o automática. El administrador requiere de soluciones o herramientas que le permitan conocer la información acerca de las conexiones que sus usuarios realicen desde y hacia donde para tratar de evitar el mal uso de los recursos del sistema o la pérdida de información valiosa para nuestros usuarios.

SCAU es una herramienta de apoyo a los administradores de sistemas, que brinda la facilidad de obtener de manera gráfica reportes sobre los accesos de los usuarios a nuestras máquinas leyendo directamente del archivo crudo wtmpx, reportes gráficos sobre las conexiones remotas que nuestros usuarios realizan hacia otros puntos de la red ejecutando de manera indefinida el comando w que brinda esta información, lleva un control sobre las denominadas cuentas dormidas (cuentas sin usar), también brinda información sobre accesos extraños al sistema y muestra un informe sobre las conexiones a nuestros equipos en horas no laborables y poco usuales. Además de todo lo anterior, SCAU está diseñado en un modelo cliente/servidor para trabajar con varias computadoras cliente que mandan la anterior información a través de la red al servidor (que se monitorea a sí mismo) y guarda toda la información en bitácoras listas para ser utilizadas por un módulo SCAU, crear la interfaz gráfica y mostrar de esta manera los resultados del filtrado y análisis de las bitácoras.

Durante el desarrollo del sistema SCAU, se tuvo la oportunidad de conocer la filosofía del sistema operativo UNIX, practicarla y desarrollar un sistema de accesos gráfico de acuerdo a las características generales de UNIX. Para lograrlo, hubo la necesidad de aprender y manejar conceptos como los siguientes:

- Las actividades de un administrador de sistemas.
- La arquitectura general de UNIX, subsistema de procesos y subsistema de archivos.
- Archivos de configuración del sistema.
- La secuencia de inicialización del sistema de registro de accesos en UNIX.
- Las estructuras y tipos de archivos generados por el sistema de registro de accesos en UNIX.
- Las llamadas al sistema UNIX.
- Servicios de red en el sistema; herramientas de desarrollo de aplicaciones cliente/servidor.
- Procesos de servicio.

Por otra parte, SCAU nos ha permitido conocer el lenguaje de programación Perl, comparar las capacidades y facilidades del lenguaje junto con su extensión Tk que es una herramienta de desarrollo para interfaces gráficas de usuario.

En la actualidad, el sistema SCAU cumple con su principal objetivo que es el de apoyar a los administradores de sistemas con reportes gráficos que contienen información acerca de las conexiones de los usuarios de todas las computadoras del Laboratorio de Visualización y el Departamento de Supercómputo de la Dirección General de Servicios de Cómputo Académico.

APENDICE

En este apartado se muestran los programas fuente que tanto la máquina servidor como la máquina cliente, deben contener para poder operar con SCAU.

1. PROGRAMA SERVIDOR DE SCAU: serv_SCAU.pl

```
#!/usr/local/bin/perl
##////////////////////////////////////////////////////////////
##################Programa servidor de SCAU###################
#
#         Autores:
#              Cervantes Santiago Berenice
#              Velasco Segura Javier
#
#            Dpto. de Supercomputo, DGSCA, UNAM
##////////////////////////////////////////////////////////////
use Socket;
use FileHandle;
use POSIX 'WNOHANG';
#// Para el control de los procesos creados
$SIG{CHLD}=sub{
     my($pid);
     while(1){
     $pid=waitpid(-1,WNOHANG);
     last if $pid <1;
     }
     };
#// Asignando un puerto en el servidor para comunicación con
#//los clientes
my $port=shift || 10001;

#//preparacion del socket
socket(SOCKET,
PF_INET,SOCK_STREAM,(getprotobyname('tcp'))[2]);
bind(SOCKET,pack('Sna4x8', AF_INET,$port,"\0\0\0\0"))||die
"No puedo preparar el puerto $port!";
listen(SOCKET,5);
my($rbits,$pid)=(' ',0);
NEW_SOCKET->autoflush; SOCKET->autoflush;

#//Proceso hijo que se ejecuta indefinidamente
if(!($pid=fork)){
while(1){
     accept(NEW_SOCKET,SOCKET); #acepta nueva conexión
```

```
     chomp($da=`date`); #si hay algo en el socket, asigna la
                        #//fecha
        @fi=split(" ",$da);
     if(!($pid=fork)){ #si es el proceso hijo
     while(1){
          vec($rbits,fileno(NEW_SOCKET),1)=1;
        select($rbits,undef,undef,undef);
        $_=<NEW_SOCKET>;#lee la clave
        if(/^clave1$/i){
          print NEW_SOCKET "si\n";
          $_=<NEW_SOCKET>; #lee el nombre del archivo.
          open(ARCR,">>$fi[1]/be$_")||die "No se pudo abrir
el archivo";
          print ARCR while (<NEW_SOCKET>);
          #Escribo lo que leo en el socket en el archivo
          close (ARCR); close NEW_SOCKET;
                              }
            if(/^clave2$/i){  #lee la clave
          print NEW_SOCKET "si\n";
          $_=<NEW_SOCKET>; #lee el nombre del archivo.
          open(ARCR,">>$fi[1]/bs$_")||die "NO SE PUDO ABRIR
               EL ARCHIVO";
          #Escribo lo que leo en el socket en el archivo
          print ARCR while (<NEW_SOCKET>);
          close (ARCR);close NEW_SOCKET;
                  }
            else {  exit;}

                    }
     close NEW_SOCKET;
     exit;
}}}
```

2. PROGRAMA CLIENTE DE SCAU: ent_SCAU_cl.pl

```
#!/usr/bin/perl
##//////////////////////////////////////////////////////////////
#Programa para crear la bitácora de entradas del cliente  de
#manera remota
#
#              Autores:
#                   Cervantes Santiago Berenice
#                   Velasco Segura Javier
#
#              Dpto. de Supercomputo, DGSCA, UNAM
```

```
##////////////////////////////////////////////////////////////
use Socket;
use FileHandle;
#//Asignando el puerto y el nombre del servidor.
$port=10001;
$server = "diphda.labvis.unam.mx";

#// Monitoreando el archivo wtmp en un primer arreglo
chomp(@var=`ls -l /var/log/wtmp`);
$pid=0;
#// Proceso hijo que se ejecuta indefinidamente
if (! ($pid=fork)) {
while(1) {
#// Monitoreando el archivo wtmp en un segundo arreglo
     chomp(@var1=`ls -l /var/log/wtmp`);
     if($var1[0] ne $var[0]) {
#// Obteniendo el nombre de la máquina
                chomp($nombre=`uname -n`);
#// Ejecuta el programa lee_wtmp para leer el archivo wtmp
          chomp($salarch=`./lee_wtmp $nombre`);
          @var=();
          @var=@var1;
          @var1=();
     }
$val=length($salarch);
#// Si la salida del programa tiene información
         if($val > 2){
#//Preparando el Socket
 socket (SOCKET, PF_INET, SOCK_STREAM,
(getprotobyname('tcp'))[2]);
  connect(SOCKET,
pack('Sna4x8',AF_INET,$port,(gethostbyname($server))[4])) ||
`echo $salarch >>fall_ent`;

        SOCKET->autoflush();
        $pid = fork(); #//Proceso hijo para comunicarse con
                       #//el servidor
        if($pid == 0) {
        $_=<SOCKET>;
         if(/^si$/) {
     #//Envío la línea del ejecutable en C.
        print SOCKET "$salarch\n";
        close SOCKET; exit;  }
        else {close SOCKET;  }
        }
```

```
        else {
     #//Envio la clave y el nombre de la máquina.
        print SOCKET "clave1\n";
        print SOCKET "$nombre\n";
        wait;
        close SOCKET;
        exec ("ent_SCAU_cl.pl");
     exit;
         }
}}}
```

3. PROGRAMA CLIENTE DE SCAU: lee_wtmp.c

```
##////////////////////////////////////////////////////////////
#Programa en C que crea el ejecutable para leer el archivo
#  wtmpx del sistema.
#
#              Autores:
#                   Cervantes Santiago Berenice
#                   Velasco Segura Javier
#
#             Dpto. de Supercomputo, DGSCA, UNAM
##////////////////////////////////////////////////////////////
#include </usr/include/stdio.h>
#include </usr/include/utmpx.h>
#include </usr/include/stdlib.h>
#define WTMPX "/var/adm/wtmpx"
#include "/usr/include/sys/types.h"
#include "/usr/include/sys/stat.h"
#include "/usr/include/time.h"
#include "/usr/include/fcntl.h"

FILE *fichero,*fp;
void tipo(int );
char *entrada;
char *entr="USER_PROCESS";
struct utmpx w;
struct utmpx d;
char *nombre;
main(int argc, char *argv[])
     { nombre=argv[1];
       if ((fichero = fopen(WTMPX,"r")) == NULL)
```

```
        { fprintf(stderr," error : no encontrado el fichero
'%s'\n",WTMPX);
                exit(0);
        }
            while (fread(&w, sizeof(w), 1, fichero))
                {  strcpy(d.ut_user,w.ut_user);
                   strcpy(d.ut_host,w.ut_host);
                   d.ut_tv=w.ut_tv;
                   d.ut_type=w.ut_type;
                }
            tipo(d.ut_type);
            fclose(fichero);
            if(entrada==entr)
              {
          printf("%s;%s;%s;%s",d.ut_user,d.ut_host,nombre,
          ctime(&d.ut_tv));
              }
}
void tipo(int valor)
{  switch(valor)
    { case 0:
             entrada="EMPTY"; break;
      case 1:
              entrada="RUN_LVL"; break;
      case 2:
             entrada="BOOT_TIME"; break;
      case 3:
              entrada="OLD_TIME"; break;
      case 4:
               entrada="NEW_TIME"; break;
      case 5:
               entrada="INIT_PROCESS"; break;
      case 6:
               entrada="LOGIN_PROCESS";  break;
      case 7:
               entrada="USER_PROCESS";  break;
      case 8:
               entrada="DEAD_PROCESS";  break;
      case 9:
               entrada="ACCOUNTING";    break;
      default:
              entrada="FALLO";  break;
   }}
```

4. PROGRAMA CLIENTE DE SCAU: sal_SCAU_cl.pl

```
#!/usr/bin/perl
##/////////////////////////////////////////////////////////////
#Programa para crear la bitácora de salidas de manera remota
#del cliente
#
#               Autores:
#                    Cervantes Santiago Berenice
#                    Velasco Segura Javier
#
#               Dpto. de Supercomputo, DGSCA, UNAM
##/////////////////////////////////////////////////////////////
use Socket;
use FileHandle;
#//Asignando el puerto y el nombre del servidor.
$port=10001;
$server="diphda.labvis.unam.mx";
@arr1=();
@arr2=();
#// Recolectando información a través del comando w en un
#primer arreglo
#// Si existe una salida con ssh, telnet o ftp, la registra.

foreach $i (`w`) {
if($i =~ /ssh/ || $i =~ /ftp/ || $i =~ /telnet/) {
if($i !~ m#/usr/local/sbin/sshd# && $i !~ /whereis/ && $i !~
/man/ && $i !~ m/-x -a/){
        $ii=substr($i,0,9);
           $iii=substr($i,43,90);
             $iiii=$ii.$iii;
       push(@arr1,$iiii);}
         }
       }
#// Obteniendo el nombre de la máquina
chomp($nombre=`uname -n`);
$pid=0;
#// Proceso hijo que se ejecuta indefinidamente
if (! ($pid=fork)) {  #// Si es el hijo entra aquí
while(1) {
@arr2=();
#//Ejecuto el comando w en forma indefinida para ver los
#//cambios
foreach $a (`w`) {
if($a =~ /ssh/ || $a =~ /ftp/ || $a =~ /telnet/) {
```

```
if($a |~ m/sshd/ && $a !~ /whereis/ && $a !~ /man/ && $a !~
m/-x -a/) {
          $aa=substr($a,0,9);
          $aaa=substr($a,43,90);
          $aaaa=$aa.$aaa;
          push(@arr2,$aaaa); }

          }
        }
#// Si el tamaño del primer arreglo es mayor, igualar con
#//arreglo2
if ($#arr1>$#arr2) { @arr1=@arr2; }
#// Obtiene la fecha actual del sistema
$fecha=`date`;
for($i=0;$i<=$#arr2;$i++)
{ #// Comparación de los arreglos para obtener solo la
  #//diferencia
    if($arr2[$i] ne $arr1[$i])
    {
     $linea=$arr2[$i];
     @arr1=@arr2;
     chomp($linea);
     chop($linea);
     $comp=$linea.";".$nombre.";".$fecha;
     chomp($comp);
#// Mandando la línea de salida a la bitácora de salidas en
#//el servidor
#//Preparando el socket
socket (SOCKET, PF_INET, SOCK_STREAM,
(getprotobyname('tcp'))[2]);
connect(SOCKET,
pack('Sna4x8',AF_INET,$port,(gethostbyname($server))[4])) ||
`echo $comp>>fall_sal`;
     SOCKET->autoflush();
     $pid = fork();
     if($pid == 0) { #/Si es el proceso hijo
     $_=<SOCKET>;
     if(/^si$/) {
#//Mandando la línea a través del socket
        print SOCKET "$comp\n";
        close SOCKET; exit;  }
        else {close SOCKET;  }
     }
     else {
#//Envío la clave y el nombre de la máquina.
```

```
        print SOCKET "clave2\n";
        print SOCKET "$nombre\n";
        wait;
        close SOCKET;
     exec ("sal_SCAU_cl.pl");
      }
    }
}}}
```

5. PROGRAMA SCAU: Crea_SCAU_Interfaz.pl

```
#!/opt/TWWfsw/bin/perl -w
##/////////////////////////////////////////////////////
##Programa Interfaz:SCAU.pl
##Autores:Berenice Cervantes Santiago
##        Javier Velasco Segura
## Depto. de Supercomputo, DGSCA, UNAM.
##/////////////////////////////////////////////////////
require 5.004;
use English;
use Tk;                          #uso de la extension Tk
use Tk::Dialog;
use Tk::ErrorDialog;
use Tk::Balloon;
use Archivo;

######Variables iniciales######
$archok=0;     # Variable que indica si se abrió el archivo
$ancho = 790; # Ancho del canvas
$alto = 520;  # Alto del canvas
$maximo = 30; # Máximo de valores a graficar en el eje X
$archivoen="bediphda"; #variable que valida la máquina en los
accesos
$archivosal="bsdiphda"; #variable que valida la máquina en
las salidas
$mes="Jan"; #Variable que define el mes a graficar
$reportet="1"; #valida casos en las entradas
$tipografica="barras";  #Variable que define el tipo de
gráfica
$conexion="1"; #Valida casos en las salidas
$como="1"; #Valida los tipos de arreglos a graficar
$comprobar="entra"; #valida las entradas y las salidas
$nusuario="curso02"; #Valida el usuario a buscar
$wPrincipal = MainWindow->new; #Ventana principal
$wPrincipal->title('SISTEMA SUPERVISOR DE CLAVES Y ACCESOS EN
```

```
UNIX');#título de la ventana principal
$wPrincipal->geometry("+10+10");

###### Menú de barras######
$menubar = $wPrincipal->Frame(
   -relief      => 'raised',
   -borderwidth => 2);
$menubar->grid(qw/-row 0 -column 0 -sticky ew/);
$menubar->gridColumnconfigure(qw/2 -weight 1/);

#### Opcion Archivo del menu de barras ####
$mbArchivo=$menubar->Menubutton(qw/-text  Archivo  -underline
0/)->grid(qw/-row 0 -column 0 -sticky nw/);
####Abrir#####
$cmAbrir = $mbArchivo->command(
   -label => '~Abrir',
   -command=>[\&abrir],-accelerator=>'Alt+a',-underline =>0);
   $wPrincipal->bind('<Alt-a>' =>sub{abrir()});
###Guardar####
$cmGuardar = $mbArchivo->command(
   -label => '~Guardar',
   -state => 'disabled',
   -command=>[\&guardar],-accelerator=>'Alt+v',-underline=>3
);
    $wPrincipal->bind('<Alt-v>' =>sub{guardar()});
####Imprimir####
$cmImprimir=$mbArchivo->command(
     -label => '~Imprimir',
   -state => 'disabled',
   -command=>[\&imprimir],-accelerator=>'Alt+i',-underline=>0
);
    $wPrincipal->bind('<Alt-i>' =>sub{imprimir()});
####Salir####
$cmAbrir = $mbArchivo->command(
   -label => '~Salir',
   -command=>[\&exit],-accelerator=>'Alt+s', -underline =>0);
    $wPrincipal->bind('<Alt-s>'=>[\&exit]);

####opcion de entradas####
$mbentradas=$menubar->Menubutton(qw/-text Entradas -underline
0/)->grid(qw/-row 0 -column 1  -sticky nw/);
###Accesos Individuales y Generales###
$mbentradas->radiobutton(
   -label => 'Accesos',
   -variable => \$reportet,
```

```
   -value =>'1',-command =>[\&entradas,$comprobar='entra'] );
####Accesos de Usuario Individual y Generales####
$mbentradas->radiobutton(
   -label => 'Accesos por Usuario',
   -variable => \$reportet,
   -value=> '2', -command =>[\&entradas,$comprobar='entra']);
####Accesos por Dia individual y Generales####
$mbentradas->radiobutton(
   -label => 'Accesos por Dia',
   -variable => \$reportet,
   -value=>'3', -command =>[\&entradas,$comprobar='entra']);
####Accesos Extranos de Todas las Maquinas#####
$mbentradas->radiobutton(
   -label => 'Dominios Extranos',
   -variable => \$reportet,
   -value=> '4', -command =>[\&entradas,$comprobar='entra']);
####Accesos en Horas no Laborables####
$mbentradas->radiobutton(
   -label => 'Horas no laborables',
   -variable => \$reportet,
   -value=> '5', -command =>[\&entradas,$comprobar='entra']);
####Cuentas  sin Usar####
$mbentradas->radiobutton(
   -label => 'Cuentas sin usar',
   -variable => \$reportet,
   -value=> '6', -command =>[\&entradas,$comprobar='entra']);
####Cuenta más Utilizada####
$mbentradas->radiobutton(
   -label => 'Cuenta mas utilizada',
   -variable => \$reportet,
   -value=> '7', -command =>[\&entradas,$comprobar='entra']);

######Opciones de salidas######
$mbsalidas = $menubar->Menubutton(qw/-text Salidas -underline
0/)->grid(qw/-row 0 -column 2 -sticky nw/);
###Salidas Generales e Individuales###
$mbsalidas->radiobutton(
   -label => 'Accesos',
   -variable => \$conexion,
   -value => '1', -command =>[\&salidas,$comprobar='sal']);
###Salidas de Usuario Generales e Individuales###
$mbsalidas->radiobutton(
   -label => 'Accesos por Usuario',
   -variable => \$conexion,
   -value => '2', -command =>[\&salidas,$comprobar='sal']);
```

```
###Salidas por Dia###
$mbsalidas->radiobutton(
   -label => 'Accesos por Dia',
   -variable => \$conexion,
   -value => '3', -command =>[\&salidas,$comprobar='sal']);

#####Salidas a Dominios Extraños#####
$mbsalidas->radiobutton(
   -label => 'Dominios Extraños',
   -variable => \$conexion,
   -value => '4', -command =>[\&salidas,$comprobar='sal']);

######## Opciones del menú de barras########
$mbgrafica = $menubar->Menubutton(qw/-text Gráfica -underline
0/)->grid(qw/-row 0 -column 3 -sticky nw/);
###Barras###.
$mbgrafica->radiobutton(
   -label => 'Barras',
   -variable => \$tipografica,
   -value => 'barras');
###Histograma###
$mbgrafica->radiobutton(
   -label => 'Histograma',
   -variable => \$tipografica,
   -value => 'histog');

######## Opción Ayuda del menú de barras ########
$menubar->Menubutton(qw/-text Ayuda -underline 0 -menuitems/
=>[[Button=>'~Acerca  de',-command=>sub{`./ayuda/ayuda.pl`}],
])->grid(qw/-row 0 -column 4 -sticky nw/);


#####Frame Principal######
$frPrincipal = $wPrincipal->Frame(
    -width      => $ancho,
    -height     => $alto,
    -relief     => 'raised',
    -borderwidth=> 2)->grid(qw/-row 1 -column 0/);

#######Canvas para la gráfica######
$canvas = $frPrincipal->Canvas(
    -height     => $alto,
    -width      => $ancho,
    -relief     => 'sunken',
    -borderwidth=> 2,)->grid(qw/-row 1 -sticky ew/);
```

```
$canvas->create('rectangle', 0, 0, $ancho, $alto, -fill =>
'white');
######Frame Botones######
$frBotones = $frPrincipal->Frame(
    -width      => $ancho,
    -height     => 50,
    -relief     => 'raised',
    -borderwidth=> 2)->grid(qw/-row 0 -sticky nsew/);

#######Creando Botones de las máquinas#######
$frBotones->Photo('img_alta',-file => "./imagen/comp03.gif");
$btalta= $frBotones->Button(
    -text       => 'ALTAIR',
    -image      => 'img_alta',
    -state      => 'normal',
    -command    => [\&altair])->pack(qw/-side left/);
###Busqueda charanda###
$frBotones->Photo('img_chara',-
file=>"./imagen/colorscr.gif");
$btcharanda=$frBotones->Button(
    -text       => 'CHARANDA',
    -image      => 'img_chara',
    -state      => 'normal',
    -command    => [\&charanda])->pack(qw/-side left/);
###Busqueda diphda###
$frBotones->Photo('img_diph',-file=>"./imagen/comp05.gif");
$btdiphda=$frBotones->Button(
    -text       => 'DIPHDA',
    -image      => 'img_diph',
    -state      => 'normal',
    -command    => [\&diphda])->pack(qw/-side left/);
###Busqueda Sotol###
$frBotones->Photo('img_soto',-file=>"./imagen/comp16.gif");
$btsotol=$frBotones->Button(
    -text       => 'SOTOL',
    -image      => 'img_soto',
    -state      => 'normal',
    -command    => [\&sotol])->pack(qw/-side left/);
###Busqueda vodka###
$frBotones->Photo('img_vod',-file=>"./imagen/comp06.gif");
 $btvodka=$frBotones->Button(
    -text       => 'VODKA',
    -image      => 'img_vod',
    -state      => 'normal',
    -command    => [\&vodka])->pack(qw/-side left/);
```

```
###Busqueda  xtabentun###
$frBotones->Photo('img_xta',-file=>"./imagen/cmptrdsk.gif");
$btxtabentun=$frBotones->Button(
    -text        => 'XTABENTUN',
    -image       => 'img_xta',
    -state       => 'normal',
    -command     => [\&xtabentun])->pack(qw/-side left/);
###Busqueda de todas ###
$frBotones->Photo('img_tod',-file=>"./imagen/network02.gif");
$bttodo=$frBotones->Button(
    -text        => 'TODOS',
    -image       => 'img_tod',
    -state       => 'normal',
    -command     => [\&todos])->pack(qw/-side left/);
###Búsqueda por nombre de mes###
$frBotones->Photo('img_mes',-file=>"./imagen/b035.gif");
$btmes= $frBotones->Button(
    -text        => 'MES',
    -image       => 'img_mes',
    -state       => 'normal',
    -command     => [\&ob_mes])->pack(qw/-side left/);
####Máquina actual####
$lis=$frBotones->Listbox(
   "-width" => 15,
   "-height" =>1 )->pack(-side=>'left', -fill=>'y');
$lis->bind('<Double-1>');

###Símbolo de DGSCA###
$frBotones->Photo('imggif',-file=>"./imagen/img.gif");
$frBotones->Label(-image=>'imggif')->pack(qw/-side right/);


#########Frame de Abajo de la Gráfica#########
$frAbajo = $frPrincipal->Frame(
    -relief      => 'sunken',
    -borderwidth=> 2,)->grid(qw/-row 2 -sticky ew/);

#####Títulos de los datos a desplegar#####
#$frAbajo->Label(-text   =>   "\nReportes   de     Accesos\n")-
>grid(qw/-row 0 -column 0/);

##Text donde se despliegan los accesos y salidas por máquina
(dependiendo del tipo)##
```

```
$frAbajo->gridColumnconfigure(qw/0 -weight 1/);
$frAbajo->gridColumnconfigure(qw/1 -weight 1/);
##Frames donde se despliegan los accesos por máquina ##
$f1=$frAbajo->Frame(
    -relief => 'sunken',
    -borderwidth=> 2)->grid(qw/-row 1 -column 0 -sticky ns/);
$fentry=$f1->Frame()->pack(-side=>'left',-fill=>'y');
$flista=$f1->Frame()->pack( -side=>'right',-fill=>'y');
$flista->Label(-text=>"Reportes de Accesos")->pack(-
side=>'top');

####Entrada para el Usuario####
$fentry->Label( -text => "\n\n\n")->pack(-side =>'top');
$fentry->Label( -text => "Buscar USUARIO")->pack(-
side=>'top');
$entrada=$fentry->Entry(-textvariable=>'hola',-
state=>'normal')->pack(-side=>'top',-fill=>'x',-padx=>'20');

###Cuando le das enter en el cuadro para introducir###
###el texto###
$entrada->bind('<Return>',\&verificar);
$lista=$flista->Listbox(
   "-width" => 65,
   "-height" =>5 )->pack(-side=>'left', -fill=>'y');
$lista->bind('<Double-1>');
$cursor=$flista->Scrollbar(
    -orient=>'vertical',
    -command=>['yview',$lista])->pack(-side=>'left',-
fill=>'y',-padx=>'0');
$lista->configure(-yscrollcommand=>['set',$cursor]);

####Búsqueda por mes######
$f2=$frAbajo->Frame(
    -relief => 'sunken',
    -borderwidth=> 2)->grid(qw/-row 1 -column 1 -sticky ns/);
$flista2=$f2->Frame()->pack( -side=>'right',-fill=>'y');
$flista2->Label( -text => "Buscar MES")->pack(-side=>'top');
$lista2=$flista2->Listbox(
   "-width" => 10,
   "-height" => 8,)->pack(-side=>'left', -fill=>'y');
$lista2->bind('<Double-1>',\&mesesito);
$cursor2=$flista2->Scrollbar(
    -orient=>'vertical',
    -command=>['yview',$lista2])->pack(-side=>'left',-
fill=>'y',-padx=>'0');
```

```
$lista2->configure(-yscrollcommand=>['set',$cursor2]);

#######Barra de estado#######
$lbStatus = $wPrincipal->Label(
    -width         => 85,
    -relief        => 'raised',
    -borderwidth   => 2,
    -anchor        => 'w',)->grid(qw/-row 3 -sticky ew/);

#####Balloons de ayuda#####
$blAyuda = $wPrincipal->Balloon (-state =>'status',-statusbar
=> $lbStatus);
$blAyuda->attach($btalta, -statusmsg => 'Cliente ALTAIR');
$blAyuda->attach($btcharanda,-statusmsg=>'Cliente CHARANDA');
$blAyuda->attach($btdiphda, -statusmsg => 'Servidor DIPHDA');
$blAyuda->attach($btsotol, -statusmsg => 'Cliente SOTOL');
$blAyuda->attach($btvodka, -statusmsg => 'Cliente VODKA');
$blAyuda->attach($btxtabentun,-statusmsg=>'Cliente
XTABENTUN');
$blAyuda->attach($bttodo, -statusmsg => 'Cliente TODAS');
$blAyuda->attach($btmes, -statusmsg => 'Busqueda por MES');

#####Dialogo para sobreescribir un archivo######
$dialog_sobreescribir = $wPrincipal->Dialog(
    -title          => 'Advertencia',
    -bitmap         => 'warning',
    -default_button => 'Cancelar',
    -buttons        => ['Ok', 'Cancelar'],
    -text           => "\241El archivo ya existe!  \n\n" .
                       "\277Desea sobreescribir?",);
$dialog_sobreescribir->configure(-font => '-*-Helvetica-Bold-
R-Normal--14-*-*-*-*-*-*');

######Dialogo de error al abrir el archivo######
$dialog_error_archivo = $wPrincipal->Dialog(
   -title   => "Error!",
   -bitmap  => 'error',
   -buttons => ['OK'],
   -text    => ' Error al abrir el archivo ',);
$dialog_error_archivo->configure(-font => '-*-Helvetica-Bold-
R-Normal--14-*-*-*-*-*-*');
```

```
####################INICIO DE SUBRUTINAS#########################
#//////////////////////////////////////////////////////////////#
###Abre el archivo de datos y prende la bandera ###
###si se seleccionó un archivo ###

sub abrir {
   if (not Exists $fs1) {   # Si no existe una ventana de
Abrir Archivo
     $fs1 = $wPrincipal->Archivo(-directory => '.');
     $fs1->configure(-title => "Abrir archivo");
   }
   $fs1->configure(-directory => './TEX');
   $arcSel = $fs1->Show;
   if (defined $arcSel) {       # Si se eligió un archivo
    print $arcSel;
    $archok=validar($arcSel);
    if ($archok==1) {          # Si el archivo es válido
      # Cuando se abre un archivo se habilitan los botones
       $btalta->configure(-state => 'normal');
       $btcharanda->configure(-state => 'normal');
       $btdiphda->configure(-state => 'normal');
       $btsotol->configure(-state => 'normal');
       $btvodka->configure(-state => 'normal');
       $btxtabentun->configure(-state => 'normal');
       $bttodo->configure(-state => 'normal');
       $tit=$todito[0];
       ($titulo,$tipografica,$como)=split("#",$tit);
       chomp ($como);
            if($como eq '1')#Abre y pone la gráfica(lee un
                            #arreglo asociativo)
           {
             `rm texto.txt`;
             `touch texto.txt`;
              $lista->delete(0,'end');
              %asociativo=();
              for ($g=1;$g<@todito;$g++)
               {
                ($key,$valor)=split("#",$todito[$g]));
                $asociativo{$key}=$valor;
               }
              $suma=0;
              foreach $k (keys(%asociativo))
                 {
```

```
$mrepo=$suma.":".$k."=".$asociativo{$k};
                    chop($mrepo);
                    `echo '$mrepo' >> texto.txt`;
                    $lista->insert('end',$mrepo);
                    $suma=$suma+1;
                   }
                graficar($titulo,$tipografica,%asociativo);
            }
          if($como eq '2') #abre y grafica
                           #(lee un arreglo normal).
            {
             `rm texto.txt`;
             `touch texto.txt`;
             $suma=0;
             $lista->delete(0,'end');
             @nuevo=();
             for($j=0;$j<=31;$j++) { $nuevo[$j]=0; }
             for($g=1;$g<@todito;$g++){
                 $nuevo[$g-1]=$todito[$g]; }
             for($j=0;$j<=31;$j++)
               {
                 $mrepo=$j."=".$nuevo[$j];
                 chop($mrepo);
                 `echo '$mrepo' >> texto.txt`;
                 $lista->insert('end',$mrepo);
               }
             graficar($titulo,$tipografica,@nuevo);
            }

      }
 }
}###Fin de la función abrir###

######Valida que el archivo se haya abierto#######
sub validar {
    ($archEleg)=@_;
if($archEleg =~ m/.scau/g)
  {
    if(open(ARCH, "$archEleg")){
       @todito=<ARCH>;
       close(ARCH);
       return 1;
    }
  }
    else {  # Muestra un mesaje de error si no pudo abrir el
```

```
            #archivo
        $dialog_error_archivo->Show;
        return 0;
    }
}###Fin de Validar###

#Guarda la gráfica en formato texto para poder recostruirla #
sub guardar {
   if (not Exists $fs2) {
     $fs2 = $wPrincipal->Archivo(-directory => '.');
     $fs2->configure(-title => "Guardar archivo");
   }
   $fs2->configure(-directory => './TEX');
   $arcGuardar = $fs2->Show;
   $arcGuardar=$arcGuardar.".scau";
   if (defined $arcGuardar)
   {
     #Verifica si el archivo existe
     if (open(PS, "$arcGuardar")) { # El archivo existe,
                                    #pregunta si se
                                    #sobreescribe
       $boton=$dialog_sobreescribir->Show;
       if ($boton eq 'Ok') {
           close(PS);
           `cp ./TEX/temp.txt  $arcGuardar`;#Copia el archivo
                                            #temporal de
                                            #texto en el arch
       }
     }
     else {                      # El archivo no existe
       close(PS);
       `cp ./TEX/temp.txt  $arcGuardar`;
     }
   }
} ###Fin de guardar###

#####Gráfica los datos de una archivo válido####
sub graficar{
@color=('gold','green','yellow','gray','red','orange','blue',
'black','magenta','brown');
$cmImprimir->configure(-state => 'normal');
$cmGuardar->configure(-state => 'normal');
`rm ./TEX/temp.txt`;
`touch ./TEX/temp.txt`;
if( $como eq '1') #si el arreglo es asociativo
```

```
{
  ($titulo,$tipografica,%arreglo)=@_; #obtiene los datos de
                                      #la grafica
  `echo  $titulo"#"$tipografica"#"$como  >>  ./TEX/temp.txt`;
#Manda el titulo al archivo
  while (($key,$valor)=each(%arreglo)) {
    `echo $key"#"$valor >> ./TEX/temp.txt`; #Manda los datos
                                  #al temporal para graficar
   }                                    #posteriormente y
                                        #guardarlo
                               #Despliega los datos en el text
  $divisionesx = scalar keys %arreglo;
  $divisionesy = 10;

# Si se excede el máximo de valores a graficar en el eje X
#solo grafica hasta el máximo
  if ( $divisionesx > $maximo) { $divisionesx = $maximo; }
#Fijo un margen de 60 para los márgenes de la gráfica
  $eje_y_xini = 60;
  $eje_y_yini = 60;
  $eje_y_yfin = $alto - 60;
  $eje_x_xini = 60;
  $eje_x_xfin = $ancho - 60;
  $magny = $eje_y_yfin - $eje_y_yini;
  $magnx = $eje_x_xfin - $eje_x_xini;
  $intervalox = $magnx / $divisionesx;
#Pinto los ejes en el canvas, ahora el eje Y
  $canvas->create('rectangle', 0, 0, $ancho, $alto, -fill =>
'white',-tags=>'popup');
  $canvas->create('line',      $eje_y_xini,      $eje_y_yini,
$eje_y_xini, $eje_y_yfin, -fill => 'black');
#ahora el eje X
  $canvas->create('line',      $eje_x_xini,      $eje_y_yfin,
$eje_x_xfin, $eje_y_yfin, -fill => 'black');
#Rutina para poner las divisiones en el eje x
  $i=1;
  while($i < $divisionesx+1){
     $canvas->create('line', $eje_y_xini+($intervalox * $i),
$eje_y_yfin-5,
     $eje_y_xini+($intervalox * $i), $eje_y_yfin+5, -fill =>
'black');
     $i++;
  }
#Rutina para poner las divisiones en el Eje Y
  $i=1;
```

```
  $intervaloy = ($magny/$divisionesy);
  while($i < $divisionesy+1){
     $canvas->create('line',    $eje_y_xini-5,    $eje_y_yfin-
($intervaloy * $i),
     $eje_y_xini+5, $eje_y_yfin-($intervaloy * $i), -fill =>
'black');
     $i++;
  }
#Pongo las etiquetas al eje X
  $pos = $eje_x_xini + ($intervalox / 2);
  for($i=0;$i<$divisionesx;$i++){
 $canvas->create('text', $pos, $eje_y_yfin+20, -text => $i );
    $pos = $pos + $intervalox;
  }
#De acuerdo a los resultados, busco el valor más grande en Y
#y hago una interpolación
#para el mapeo de los valores, aquí solo es un ejemplo
  $maxy = 0;
  foreach $i (keys(%arreglo)) {
     if($maxy <= $arreglo{$i}){ $maxy = $arreglo{$i}; }
  }
  $incry = $maxy/$divisionesy;
#Etiqueto al eje Y con los valores
  for($i=0;$i<$divisionesy;$i++){
$canvas->create('text',      $eje_y_xini-20,      $eje_y_yfin-
($intervaloy*($i+1)),
-text => sprintf("%3d",($i+1)*$incry));
  }
########
$canvas->create('text',32,41,-text=>sprintf("ACCESOS"));
$canvas->create('text',$eje_x_xfin+5,$eje_y_yfin+30,-
text=>sprintf("MAQUINAS"));
#obtengo valores para la grafica
@valores=();
foreach $valor (values(%asociativo)) { push(@valores,$valor);
}
  #Ahora dibujo la gráfica
  $b = 1/$incry;
  $unidad = $b * $intervaloy;
  $pos = $eje_y_xini+ $intervalox/2;
  $xini = $pos;
  $yini = $eje_y_yfin - ($unidad * $valores[0]);
  #Rutina para graficar histograma
$jw=0;
  if($tipografica eq 'histog'){
```

```
      for($i=1;$i<$divisionesx;$i++){
        $canvas->create('line',          $xini,          $yini,
$pos+($intervalox * $i),
        $eje_y_yfin-($unidad   *   $valores[$i]),   -fill   =>
$color[$jw]);
        $xini = $pos + ($intervalox * $i);
        $yini = $eje_y_yfin - ($unidad * $valores[$i]);
       $jw=$jw+1;
      if($jw== 10){ $jw=0;}
      }
  }
  elsif($tipografica eq 'barras'){
  $jw=0;
  #Rutina para graficar barras
    for($i=0;$i<$divisionesx;$i++){
     $canvas->create('rectangle',  $eje_y_xini+($intervalox  *
$i), $eje_y_yfin,
     $eje_y_xini+($intervalox * ($i+1)), $eje_y_yfin-($unidad
* $valores[$i]),
     -fill => $color[$jw],-tags=>'popup');
     $jw=$jw+1;
     if($jw== 10){ $jw=0;}
    }
  }

#Título de la gráfica
   $canvas->create('text',  350,  30,  -text  =>sprintf("%s",
$titulo) );
}
if ($como eq '2') #si el arreglo es normal
{
 ($titulo,$tipografica,@arreglo)=@_; # los  valores de la
                                     #gráfica
 `echo          $titulo"#"$tipografica"#"$como          >>
./TEX/temp.txt`;#Guarda el título en el temporal
#Despliega los datos en el text
  $divisionesx = scalar @arreglo;
  $divisionesy = 10;
  # Si se excede el máximo de valores a graficar en el eje X
#solo grafica hasta el máximo
  if ( $divisionesx > $maximo) { $divisionesx = $maximo; }
#Fijo un margen de 60 para los márgenes de la gráfica
  $eje_y_xini = 60;
  $eje_y_yini = 60;
  $eje_y_yfin = $alto - 60;
```

```
  $eje_x_xini = 60;
  $eje_x_xfin = $ancho - 60;
  $magny = $eje_y_yfin - $eje_y_yini;
  $magnx = $eje_x_xfin - $eje_x_xini;
  $intervalox = $magnx / $divisionesx;
#Pinto los ejes en el canvas, ahora el eje Y
  $canvas->create('rectangle', 0, 0, $ancho, $alto, -fill =>
'white',-tags=>'popup');
  $canvas->create('line',      $eje_y_xini,      $eje_y_yini,
$eje_y_xini, $eje_y_yfin, -fill => 'black');
#ahora el eje X
  $canvas->create('line',      $eje_x_xini,      $eje_y_yfin,
$eje_x_xfin, $eje_y_yfin, -fill => 'black');
#Rutina para poner las divisiones en el eje x
  $i=1;
  while($i < $divisionesx+1){
    $canvas->create('line', $eje_y_xini+($intervalox * $i),
$eje_y_yfin-5,
    $eje_y_xini+($intervalox * $i), $eje_y_yfin+5, -fill =>
'black');
    $i++;
  }
#Rutina para poner las divisiones en el Eje Y
  $i=1;
  $intervaloy = ($magny/$divisionesy);
  while($i < $divisionesy+1){
     $canvas->create('line',   $eje_y_xini-5,   $eje_y_yfin-
($intervaloy * $i),
     $eje_y_xini+5, $eje_y_yfin-($intervaloy * $i), -fill =>
'black');
     $i++;
  }
#Pongo las etiquetas al eje X
  $pos = $eje_x_xini + ($intervalox / 2);
  for($i=0;$i<$divisionesx;$i++){
    $canvas->create('text', $pos, $eje_y_yfin+20, -text => $i
);
    $pos = $pos + $intervalox;
  }
#De acuerdo a los resultados, busco el valor mas grande en Y
#y hago una interpolación
#para el mapeo de los valores, aquí solo es un ejemplo
  $maxy = 0;
  for($i=0;$i<@arreglo;$i++)
      {
```

```
      `echo "$arreglo[$i]" >> ./TEX/temp.txt`;
       if($maxy <= $arreglo[$i]){
       $maxy = $arreglo[$i];
      }
  }
  $incry = $maxy/$divisionesy;
#Etiqueto al eje Y con los valores
  for($i=0;$i<$divisionesy;$i++){
    $canvas->create('text',    $eje_y_xini-20,    $eje_y_yfin-
($intervaloy*($i+1)),
    -text => sprintf("%3d",($i+1)*$incry));
  }
$canvas->create('text',32,41,-text=>sprintf("ACCESOS"));
$canvas->create('text',$eje_x_xfin+5,$eje_y_yfin+30,-
text=>sprintf("DIAS"));
  #Ahora dibujo la grafica
  $b = 1/$incry;
  $unidad = $b * $intervaloy;
  $pos = $eje_y_xini+ $intervalox/2;
  $xini = $pos;
  $yini = $eje_y_yfin - ($unidad * $arreglo[0]);
  $jw=0;
  #Rutina para graficar histograma
  if($tipografica eq 'histog'){
     for($i=1;$i<$divisionesx;$i++){
       $canvas->create('line',          $xini,          $yini,
$pos+($intervalox * $i),
       $eje_y_yfin-($unidad   *   $arreglo[$i]),   -fill   =>
$color[$jw]);
       $xini = $pos + ($intervalox * $i);
       $yini = $eje_y_yfin - ($unidad * $arreglo[$i]);
     $jw=$jw+1;
     if($jw== 10){ $jw=0;}
     }
  }
  elsif($tipografica eq 'barras'){
  $jw=0;
  #Rutina para graficar barras
    for($i=0;$i<$divisionesx;$i++){
       $canvas->create('rectangle',  $eje_y_xini+($intervalox
* $i), $eje_y_yfin,
       $eje_y_xini+($intervalox   *   ($i+1)),   $eje_y_yfin-
($unidad * $arreglo[$i]),
       -fill => $color[$jw],-tags=>'popup');
       $jw=$jw+1;
```

```
        if($jw== 10){ $jw=0;}
    }
  }

#Titulo de la gráfica
   $canvas->create('text',  350,  30,  -text  =>sprintf("%s",
$titulo) );
}
###
} ###Fin de la función graficar

#######Funciones de validación de salidas y entradas #######
sub  altair{ $archivoen='bealtair'; $archivosal='bsaltair'; }
sub                          charanda{$archivoen='becharanda';
archivosal='bscharanda'; }
sub diphda{ $archivoen='bediphda'; $archivosal='bsdiphda'; }
sub sotol { $archivoen='besotol'; $archivosal='bssotol'; }
sub vodka { $archivoen='bevodka'; $archivosal='bsvodka'; }
sub           xtabentun{            $archivoen='bextabentun';
$archivosal='bsxtabentun'; }
sub todos{ $archivoen="todos"; $archivosal="todos"; }

#########Función para las entradas#########
sub entradas{
 $lis->delete(0,'end');#inicializo la lista
  #$lis->insert('end',$archivoen);
if($archivoen          ne          'todos'){          $lis-
>insert('end',uc(substr($archivoen,2)));}
if($archivoen                eq                'todos'){$lis-
>insert('end',uc($archivoen));}
if(($reportet eq '1') && ($archivoen ne 'todos'))#Opción de
accesos de una máquina
 {
  `rm texto.txt`; #borro el archivo temporal
  `touch texto.txt`;
   $suma=0;
   %asociativo=();
   $ruta="./".$mes."/".$archivoen; #asigno la ruta
   $ARCR=$ruta;
   open(ARCR)||die "No se pudo abrir el archivo";
   while(<ARCR>) {
                chomp(@arry=split(";",$_));
                if($arry[1] eq '') { $arry[1]="consola";
                        #asigno los accesos desde consola
                  }
```

```
                    $asociativo{$arry[1]}++; #Asigno los
                                             #valores al arreglo
                                             #asociativo
                  }
  $titulo="Reporte de Accesos vs Máquina Origen de
".$archivoen." del mes de ".$mes;
  `echo '$titulo' >> texto.txt`;   #mando el título de la
                                   #gráfica
  $lista->delete(0,'end');#inicializo la lista
  while(($mremota,$num)=each(%asociativo)){
       $mrepo=$mremota."=".$num;
       $salidita=$suma.":".$mrepo;
       `echo '$salidita' >> texto.txt`; #mando los valores al
                                        #archivo temporal
       $lista->insert('end',$salidita); #mando los valores a
                                        #la lista
       $suma=$suma+1;
      }
  $como='1';
  $titulo="Gráfica de Accesos vs Máquina Origen de
".$archivoen." del mes de ".$mes;
  graficar($titulo, $tipografica, %asociativo);#Mando a
                                               #graficar
}

if(($reportet eq '2') && ($archivoen ne 'todos') )#opcion de
                                        #accesos de usuario
{
 `rm texto.txt`;
 `touch texto.txt`;
 $suma=0;
 %asociativo=();
 $entrada->configure(-state => 'normal');
 $nusuario=$entrada->get;  #Obtengo el nombre del usuario
 chomp($nusuario);
 $ruta="./".$mes."/".$archivoen;#Obtengo la ruta
 $ARCR=$ruta;
 open(ARCR)||die "No se pudo abrir el archivo";
 while(<ARCR>) {
       chomp(@arry=split(";",$_));
       if($nusuario eq $arry[0])
          {
            $asociativo{$arry[1]}++; #pongo los valores en el
                                     #arreglo
          }
```

```
        }
 $titulo="Reporte de Accesos vs Máquina Origen de
".$archivoen." del usuario ".$nusuario." del mes de ".$mes;
 `echo '$titulo' >> texto.txt`;
 $lista->delete(0,'end');
 while(($mremota,$num)=each(%asociativo)){
       $mrepo=$mremota."=".$num;
       $salidita=$suma.":".$mrepo;
       `echo '$salidita' >> texto.txt`;#pongo
                          #la información en el archivo
                          #temporal
       $lista->insert('end',$salidita);
       $suma=$suma+1;
       }
 $como='1';
 $titulo="Gráfica de Accesos vs Máquina Origen de
".$archivoen." del usuario ".$nusuario." del mes de ".$mes;
 graficar($titulo, $tipografica, %asociativo); #Graficar el
                                               #arreglo
}

if(($reportet eq '3') && ($archivoen ne 'todos'))#Opción de
                                          #accesos por dia
{
 %asociativo=();
 `rm texto.txt`;
 `touch texto.txt`;
 $suma=0;
 $ruta="./".$mes."/".$archivoen;#asigno la ruta
 $ARCR=$ruta;
 open(ARCR)||die "No se pudo abrir el archivo";
 while(<ARCR>)       {       chomp(@arry=split("       ",$_));
$asociativo{$arry[2]}++; }
 $lista->delete(0,'end');
 @nuevo=();
 for($j=0;$j<=31;$j++) { $nuevo[$j]=0; }     #inicializo el
                                            #arreglo en ceros
 $titulo="Reporte de Accesos vs  Dia   de ".$archivoen." del
mes de ".$mes;
 `echo '$titulo' >> texto.txt`;
  while(($dia,$num)=each(%asociativo)){ $nuevo[$dia]=$num; }
  for($j=0;$j<=31;$j++) {
     $mrepo=$j."=".$nuevo[$j];
     `echo '$mrepo' >> texto.txt`;
     $lista->insert('end',$mrepo);
```

```
  }
  $como='2';
  $titulo="Gráfica de Accesos vs Dia  de ".$archivoen." del
mes de ".$mes;
  graficar($titulo, $tipografica, @nuevo);
}

if(($reportet eq '1') && ($archivoen eq 'todos'))#Accesos de
                                                  #todos
 {
  `rm texto.txt`;
  `touch texto.txt`;
  $suma=0;
  %asociativo=();
  chomp(@hosts=`ls $mes| grep be`);
  $lista->delete(0,'end');
  $ruta="./".$mes."/";
  for ($i=0;$i<=$#hosts;$i++)
   {
    $num=`wc -l  $ruta$hosts[$i]`;
    ($host,$n)=split(" ",$num);
    @bas=split("/",$n);
    $mrepo=$host."=".$bas[2];
    if($bas[2]      ne    'bsxtabentun')    {    $nom=$bas[2];
$asociativo{$nom}=$host; }
   }
  $titulo="Reporte  de  Accesos  vs  Máquina  Origen  de
".$archivoen." del mes de ".$mes;
  `echo '$titulo' >> texto.txt`;
  foreach $k (keys(%asociativo))
   {
    $mrepo=$suma.":".$k."=".$asociativo{$k};
    `echo '$mrepo' >> texto.txt`;
    $lista->insert('end',$mrepo);
    $suma=$suma+1;
   }
 $como='1';
 $titulo="Gráfica  de  Accesos  vs  Máquina  Origen  de
".$archivoen." del mes de ".$mes;
 graficar($titulo, $tipografica, %asociativo);
}

if(($reportet eq '2') && ($archivoen eq 'todos'))#Accesos de
                                                  #usuario de todos
{
```

```
 `rm texto.txt`;
 `touch texto.txt`;
 $suma=0;
 %asociativo=();
 @accesos=();
 $entrada->configure(-state => 'normal');
 $nusuario=$entrada->get;
 chomp($nusuario);
 $titulo="Reporte de Accesos vs Máquina Origen de
".$archivoen." del usuario ".$nusuario." del mes de ".$mes;
 `echo '$titulo' >> texto.txt`;
 chomp(@hosts=`ls $mes| grep be`);
 $lista->delete(0,'end');
 for ($i=0;$i<$#hosts;$i++)
   {
    $res=0;
    $ARCR="./".$mes."/".$hosts[$i];
    $cont=0;
    open(ARCR)||die "No se pudo abrir el archivo";
    while(<ARCR>) {
          chomp(@arry=split(";",$_));
          if($nusuario eq $arry[0]) { $res=$cont;
$cont=$cont+1; }
           $accesos[$i]=$res;
           }
  $mrepo=$suma.":".$hosts[$i]."=".$accesos[$i];
  `echo '$mrepo' >> texto.txt`;
  $lista->insert('end',$mrepo);
  $suma=$suma+1;
  }
$como='2';
$titulo="Gráfica de Accesos vs Máquina Origen de
".$archivoen." del usuario ".$nusuario." del mes de ".$mes;
graficar($titulo, $tipografica, @accesos);
}

if (($reportet eq '3') && ($archivoen eq 'todos')) #accesos
                                                   #por dia de todos
{
 `rm texto.txt`;
 `touch texto.txt`;
 $suma=0;
 $titulo="Reporte de Accesos vs Dia de ".$archivoen." del
mes de ".$mes;
 `echo '$titulo' >> texto.txt`;
```

```
 %asociativo=();
 @nuevo=();
 for($j=0;$j<=31;$j++) { $nuevo[$j]=0; }
 chomp(@hosts=`ls $mes | grep be`);
 $lista->delete(0,'end');
 for ($i=0;$i<$#hosts;$i++)
    {
      $ARCR="./".$mes."/".$hosts[$i];
      $cont=0;
      open(ARCR)||die "No se pudo abrir el archivo";
      while(<ARCR>)     {     chomp(@arry=split("     ",$_));
$asociativo{$arry[2]}++; }
    }#
 while(($dia,$num)=each(%asociativo)){ $nuevo[$dia]=$num; }
 for($i=0;$i<=31;$i++) {
    $mrepo=$i.":".$nuevo[$i];
    `echo '$mrepo' >> texto.txt`;
    $lista->insert('end',$mrepo);
   }
 $como='2';
 $titulo="Gráfica de Accesos vs Dia  de ".$archivoen." del
mes de ".$mes;
 graficar($titulo, $tipografica, @nuevo);
}

if(($reportet eq '4') && ($archivoen eq 'todos'))#Dominios
                                                 #extraños
{
 `rm texto.txt`;
 `touch texto.txt`;
 $titulo="Reporte de Accesos de Dominios Extraños"." del mes
de ".$mes;
 `echo '$titulo' >> texto.txt`;
 %asociativo=();
 chomp(@hosts=`ls $mes| grep be`);
 $lista->delete(0,'end');
 for ($i=0;$i<$#hosts;$i++)
   {
    $ARCR="./".$mes."/".$hosts[$i];
    open(ARCR)||die "No se pudo abrir el archivo";
    while(<ARCR>) {
              chomp(@arry=split(";",$_));

if(!(($arry[1]=~m/labvis/g)||($arry[1]=~m/132.248.159/g)||($a
rry[1]=~m/super/g)||($arry[1]=~m/132.248.159.15/g)||($arry[1]
```

```
=~m/132.248/g)||($arry[1]=~//)))
                {
                   if(length($arry[1])>=1)
                   {

if(!(($arry[1]=~m/labvis/g)||($arry[1]=~m/132.248.159/g)||($a
rry[1]=~m/super/g)||($arry[1]=~m/132.248.159
.15/g)||($arry[1]
=~m/132.248/g)))
                {
                 if($arry[0] eq 'root')
                   {
                    $old=$arry[0];
                    $arry[0]=$arry[0].$hosts[$i];
                    $mrepo=$old."  acceso  de:".$arry[1]."  el
".$arry[-1]."a ".$arry[2];
                    `echo '$mrepo' >> texto.txt`;
                    $lista->insert('end',$mrepo);
                    $asociativo{$arry[0]}++;
                   }
                else
                   {
                    $mrepo=$arry[0]."  acceso  de:".$arry[1]."
el ".$arry[-1]."a ".$arry[2];
                    `echo '$mrepo' >> texto.txt`;
                    $lista->insert('end',$mrepo);
                    $asociativo{$arry[0]}++;
                   }
                }
              }
         }
    }}

 while(($usuario,$num)=each(%asociativo)){
 $mrepo=$usuario."=".$num;
 $lista->insert('end',$mrepo);
 `echo '$mrepo' >> texto.txt`;
 }
}

if(($reportet eq '5') && ($archivoen eq 'todos'))#Horas no
                                                  #laborables
{
 `rm texto.txt`;
 `touch texto.txt`;
```

```
 $titulo="Reporte de Accesos de Horas no Laborables"." del
mes de ".$mes;
 `echo '$titulo' >> texto.txt`;
 %asociativo=();
 $lista->delete(0,'end');
 chomp(@hosts=`ls $mes| grep be`);
 for ($i=0;$i<$#hosts;$i++)
   {
    $ARCR="./".$mes."/".$hosts[$i];
    open(ARCR)||die "No se pudo abrir el archivo";
    while(<ARCR>) {
               chomp(@arry=split(" ",$_));
               chop($_);
               @hora=split(":",$arry[3]);
               if($hora[0] < 9 || $hora[0] > 21) { $lista-
>insert('end',$_); `echo '$_' >> texto.txt`; }
                 }
    }
}

if(($reportet eq '6') && ($archivoen eq 'todos'))#cuenta mas
                                                  #usada
{
 `rm texto.txt`;
 `touch texto.txt`;
 $suma=0;
 $titulo="Reporte de Cuentas sin Usar"." del mes de ".$mes;
 `echo '$titulo' >> texto.txt`;
 $lista->delete(0,'end');
 chomp(@hosts=`ls $mes| grep be`);
 chomp(@passwd=`ls | grep passwd`);
 for ($i=0;$i<=$#passwd;$i++)
   {
    $ARCR=$passwd[$i];
 $nombre=substr($passwd[$i],6);
    open(ARCR)||die "No se pudo abrir el archivo";
    while(<ARCR>) {
                  $ban=0;
                  chomp(@arry=split(":",$_));
                  $ARC="./".$mes."/".$hosts[$i];
                  open(ARC)||die  "No  se  pudo  abrir  el
archivo";
                  while(<ARC>) {
                                chomp(@arr=split(";",$_));
                          #     $nombre=$arr[2];
```

```
                                        if($arry[0] eq $arr[0]) {
$ban=1; }
                                      }
                        if($ban!=1)
                          {
       if($arry[0] ne 'daemon' &&$arry[0] ne 'operator' &&
$arry[0] ne 'bin' && $arry[0]
 ne 'uucp' && $arry[0] ne 'www' && $arry[0] ne 'named' &&
$arry[0] ne 'nobody' && $arry[0] ne 'sys' && $
arry[0] ne 'adm' && $arry[0] ne 'lp'&& $arry[0] ne 'smtp' &&
$arry[0] ne 'nuucp' && $arry[0] ne 'listen'
 && $arry[0] ne'noaccess' && $arry[0] ne 'nobody4')
                         {
                          $repo=$suma.":".$nombre."=".$arry[0];
                          `echo '$repo' >> texto.txt`;
                          $lista->insert('end',$repo);
                          $suma=$suma+1;
                         }
                        }
                        }
}
}
##########
if(($reportet eq '7')  && ($archivoen eq 'todos'))#Cuenta mas
                                                  #utilizada
{
 $lista->delete(0,'end');
 chomp(@hosts=`ls $mes| grep be`);
 for ($i=0;$i<$#hosts;$i++)
   {
   %asociativo=();
   $ARCR="./".$mes."/".$hosts[$i];
   open(ARCR)||die "No se pudo abrir el archivo";
    while(<ARCR>)        {        chomp(@arry=split(";",$_));
$maquina=$arry[2]; $asociativo{$arry[0]}++; }
 $numero=0;
 $login="";
 while(($nombre,$num)=each(%asociativo)){
     if($num > $numero) { $numero=$num; $lon=length($nombre);
$login=substr($nombre,0,$lon); }
    }
   $repo=$maquina."=".$login."=".$numero;
   $lista->insert('end',$repo);
   }
 }
```

```
}#Fin de la  subrutina entradas

sub salidas {   #Subrutina de salidas
 $lis->delete(0,'end');#inicializo la lista
  #$lis->insert('end',uc(substr($archivosal,2)));
if($archivosal          ne          'todos'){          $lis-
>insert('end',uc(substr($archivosal,2)));}
if($archivosal                eq                'todos'){$lis-
>insert('end',uc($archivosal));}

if(($conexion eq '1') && ($archivosal ne 'todos'))#salida de
                                                  #individual
 {
  `rm texto.txt`;
  `touch texto.txt`;
  $suma=0;
  %asociativo=();
  $RUTA=$mes."/".$archivosal;
  if(!chomp(@lin1=`more    $RUTA    |cut    -c1-9`)){    print
"\nERROR\n"; exit; }
  (@lin2=`more $RUTA |cut -c10-65`);
  for              ($s=0;$s<@lin1;$s++)                    {
$arr[$s]="$lin1[$s]".";"."$lin2[$s]"; }
  foreach $d (@arr) {
   ($c1,$c2,$c3,$c4)=split(";",$d);
   $pos=rindex($c2," ")+1;
   $nombre=substr($c2,$pos);
   if(!rindex($nombre,"-")){ next; }
   if($c2      =~      /@/)      {      $pos=rindex($c2,"@")+1;
$nombre=substr($c2,$pos); }
   if($c2 =~ /-1/) { @c22=split(" ",$c2); if($c22[2] =~ /-1/)
{ $nombre=$c22[1]; } }
   $graf1[$g1]="$c1".";"."$nombre";
   $g1++;
   }
  foreach      $s      (@graf1)      {      @gra=split(";",$s);
$asociativo{$gra[1]}++; }
  $lista->delete(0,'end');
  $titulo="Reporte   de   Salidas   vs   Máquina   Origen   de
".$archivosal." del mes de ".$mes;
  `echo '$titulo' >> texto.txt`;
  while(($mremota,$num)=each(%asociativo)){
      $mrepo=$suma.":".$mremota."=".$num;
      `echo '$mrepo' >> texto.txt `;
      $lista->insert('end',$mrepo);
```

```
      $suma=$suma+1;
      }
  $como='1';
  $titulo="Gráfica de Salidas vs Máquina Origen de
".$archivosal." del mes de ".$mes;
  graficar($titulo, $tipografica, %asociativo);
}

if(($conexion eq '2') && ($archivosal ne 'todos'))#reporte
                                                  #por usuario
                                                  #de uno
 {
  `rm texto.txt`;
  `touch texto.txt`;
  $suma=0;
  %asociativo=();
  $RUTA=$mes."/".$archivosal;
  if(!chomp(@lin1=`more    $RUTA    |cut    -c1-9`)){    print
"\nERROR\n"; exit; }
  (@lin2=`more $RUTA |cut -c10-65`);
  for                ($s=0;$s<@lin1;$s++)                {
$arr[$s]="$lin1[$s]".";"."$lin2[$s]"; }
  foreach $d (@arr) {
    ($c1,$c2,$c3,$c4)=split(";",$d);
    $pos=rindex($c2," ")+1;
    $nombre=substr($c2,$pos);
    if(!rindex($nombre,"-")){ next; }
    $pos1=index($c1," ");
    $usu=substr($c1,0,$pos1);
    $graf1[$g1]="$usu".";"."$nombre";
    $g1++;
    }
  $entrada->configure(-state => 'normal');
  $usuario=$entrada->get;
  chomp($usuario);
  foreach $s (@graf1) { @gra=split(";",$s); if($usuario eq
$gra[0]) { $asociativo{$gra[1]}++; } }
  $lista->delete(0,'end');
$titulo="Reporte de Salidas vs Máquina Origen de
".$archivosal." del usuario ".$usuario." del mes de ".$mes;
  `echo '$titulo' >> texto.txt`;
  while(($mremota,$num)=each(%asociativo)){
    $mrepo=$suma.":".$mremota."=".$num;
    `echo '$mrepo' >> texto.txt`;
    $lista->insert('end',$mrepo);
```

```
    $suma=$suma+1;
    }
  $como='1';
$titulo="Gráfica de Salidas vs Máquina Origen de
".$archivosal." del usuario ".$usuario." del mes de ".$mes;
  graficar($titulo, $tipografica, %asociativo);
}


if(($conexion eq '3') && ($archivosal ne 'todos'))#Reporte
                                                  #por dia individual
 {
  `rm texto.txt`;
  `touch texto.txt`;
  $suma=0;
  %asociativo=();
  $RUTA=$mes."/".$archivosal;
  open(RUTA)||die "No se pudo abrir el archivo";
  while(<RUTA>) { chomp(@arry=split(";",$_)); @arr1=split("
",$arry[2]); $asociativo{$arr1[2]}++; }
  $lista->delete(0,'end');
  @nuevo=();
  for($j=0;$j<=31;$j++) { $nuevo[$j]=0; }
  while(($dia,$num)=each(%asociativo)){ $nuevo[$dia]=$num; }
  $titulo="Reporte de Salidas vs Dia de ".$archivosal." del
mes de ".$mes;
  `echo '$titulo' >> texto.txt`;
  for($j=0;$j<=31;$j++) {
       $mrepo=$j.":".$nuevo[$j];
       $lista->insert('end',$mrepo);
       `echo '$mrepo' >> texto.txt`;
    }
  $como='2';
  $titulo="Gráfica de Salidas vs Dia de ".$archivosal." del
mes de ".$mes;
  graficar($titulo, $tipografica, @nuevo);
}#fin if3

if(($conexion eq '1') && ($archivosal eq 'todos'))#salidas de
                                                    #todos
 {
  `rm texto.txt`;
  `touch texto.txt`;
  $suma=0;
  %asociativo=();
```

```
  $RUTA=$mes."/".$archivosal;
  $lista->delete(0,'end');
  chomp(@hosts=`ls $mes| grep bs`);
  for ($i=0;$i<=$#hosts;$i++)
   {
    $num=`wc -l  $mes/$hosts[$i]`;
    ($host,$n)=split(" ",$num);
    @bas=split("/",$n);
    $mrepo=$host."=".$bas[1];
    $nom=$bas[1];
    $asociativo{$nom}=$host;
   }
  $titulo="Reporte de Salidas vs Máquina Origen de
".$archivosal. " del mes de ".$mes;
  `echo '$titulo' >> texto.txt`;
  foreach $k (keys(%asociativo))
     {
      $mrepo=$suma.":".$k."=".$asociativo{$k};
      $lista->insert('end',$mrepo);
      `echo '$mrepo' >> texto.txt`;
      $suma=$suma+1;
     }
  $como='1';
  $titulo="Gráfica de Salidas vs Máquina Origen de
".$archivosal." del mes de ".$mes;
  graficar($titulo, $tipografica, %asociativo);
  }

if(($conexion eq '2') && ($archivosal eq 'todos'))#Salidas
                                    #del usuario de todos
 {
  `rm texto.txt`;
  `touch texto.txt`;
  $suma=0;
  %asociativo=();
  $RUTA=$mes."/".$archivosal;
  $lista->delete(0,'end');
  @salidas=();
  $entrada->configure(-state => 'normal');
  $nusuario=$entrada->get;
  chomp($nusuario);
  $titulo="Reporte de Salidas vs Máquina Origen de
".$archivosal." del usuario ".$nusuario." del mes de ".$mes;
  `echo '$titulo' >> texto.txt`;
  chomp(@hosts=`ls $mes| grep bs`);
```

```
  for ($i=0;$i<=$#hosts;$i++)
     {
      $res=0;
      $ARCR=$mes."/".$hosts[$i];
      $cont=0;
      if(!chomp(@lin1=`more   $ARCR   |cut   -c1-9`)){   print
"\nERROR\n"; exit; }
    foreach $d (@lin1) {
        ($c1,$c2,$c3)=split(";",$d);
     $pos=index($c1," ");
     $nombre=substr($c1,0,$pos);
        if($nusuario eq $nombre) { $res=$cont; $cont=$cont+1;
}
         $accesos[$i]=$res;
        }
    $mrepo=$suma.":".$hosts[$i]."=".$accesos[$i];
    `echo '$mrepo' >> texto.txt`;
    $lista->insert('end',$mrepo);
    $suma=$suma+1;
  }
 $como='2';
$titulo="Gráfica   de   Salidas   vs   Máquina   Origen   de
".$archivosal." del usuario ".$nusuario." del mes de ".$mes;
  graficar($titulo, $tipografica, @accesos);
}#fin del if5

if(($conexion eq '3') && ($archivosal eq 'todos'))#Reporte de
                                                  #salidas de todos
 {
  `rm texto.txt`;
  `touch texto.txt`;
  $titulo="Reporte de Salidas vs Dia  de ".$archivosal." del
mes de ".$mes;
  `echo '$titulo' >> texto.txt`;
  %asociativo=();
  @nuevo=();
  for($j=0;$j<=31;$j++) { $nuevo[$j]=0; }
  $RUTA=$mes."/".$archivosal;
  $lista->delete(0,'end');
  chomp(@hosts=`ls $mes| grep bs`);
  for ($i=0;$i<=$#hosts;$i++)
   {
    $ARCR=$mes."/".$hosts[$i];
    $cont=0;
    open(ARCR)||die "No se pudo abrir el archivo";
```

```
    while(<ARCR>) {
                chomp(@arry=split(";",$_));
                 @arr1=split(" ",$arry[2]);
                 $asociativo{$arr1[2]}++;
               }
    }
  while(($dia,$num)=each(%asociativo)){ $mrepo=$dia."=".$num;
$nuevo[$dia]=$num; }
  $titulo="Reporte de Salidas vs Dia  de ".$archivosal." del
mes de ".$mes;
  `echo '$titulo' >> texto.txt`;
  for($j=0;$j<=31;$j++)
   {
    $mrepo=$j.":".$nuevo[$j];
    $lista->insert('end',$mrepo);
    `echo '$mrepo' >> texto.txt`;
   }
   $como='2';
   $titulo="Gráfica de Salidas vs Dia  de ".$archivosal." del
mes de ".$mes;
   graficar($titulo, $tipografica, @nuevo);
}#fin del if6

if(($conexion eq '4') && ($archivosal eq 'todos'))#Dominios
                                                  #extraños
 {
  `rm texto.txt`;
  `touch texto.txt`;
   $titulo="Reporte de Salidas a Dominios Extraños"." del mes
de ".$mes;
  `echo '$titulo' >> texto.txt `;
  %asociativo=();
  $RUTA=$mes."/".$archivosal;
  $lista->delete(0,'end');
  @lin1=();
  @lin2=();
  $AR="./hosts";
  chomp(@hosts=`ls $mes| grep bs`);
  $nombre="";
  for ($i=0;$i<=$#hosts;$i++)
   {
     @arr=();
     $ARCR=$mes."/".$hosts[$i];
     $cont=0;
     if(!chomp(@lin1=`more   $ARCR   |cut   -c1-9`)){   print
```

```
"\nERROR\n"; exit; }
     (@lin2=`more $ARCR |cut -c10-65`);
     for                ($s=0;$s<@lin1;$s++)                {
$arr[$s]="$lin1[$s]".";"."$lin2[$s]"; }
     foreach $d (@arr) {
        ($c1,$c2,$c3,$c4)=split(";",$d);
        $pos=rindex($c2," ")+1;
        $nombre=substr($c2,$pos);
        @nombre2=();
        @nombre2=split(" ",$c2);
        if(!rindex($nombre,"-")){ next; }
        @nomb=();
        if($nombre !~ /132.248/ && $nombre !~ /unam/ ){
        $ban=0;
        $band=0;
        if($nombre   =~   /@/)   {   @nomb=split("@",$nombre);
$band=1; }
         open(AR) ||die "no puedo abrir la tabla de hosts";
           while (<AR>) {
                     if($_ =~ /$nombre/) { $ban=1;     }
                     if(defined($nombre2[1]))   {   if($_   =~
m/$nombre2[1]/) { $ban=1; }
                      }
                      if($band == 1) { if($_ =~ /$nomb[1]/)
{ $ban=1; }}
                    }
        if($ban == 1) { next; }
        else { chop($d); $lista->insert('end',$d); `echo '$d'
>> texto.txt`; }
           }
   }
}
}#fin del if7##
}########Fin de la subrutina de salidas########

sub ob_mes { #obtiene los meses monitoreados por SCAU

@meses=('Jan','Feb','Mar','Apr','May','Jun','Jul','Aug','Sep'
,'Oct','Nov','Dec');
  @resu=();
  $lista2->delete(0,'end');
  for($k=0;$k<@meses;$k++)
    {
     @resu=`du $meses[$k]`;
     ($tam)=split(" ",$resu[0]);
```

```
      if ($tam != 2) { $lista2->insert('end',$meses[$k]); }
    }
}#fin de ob_mes

sub mesesito{ #pone los meses activos es la interfaz
  $mes=$lista2->get('active');
#print $mes;
}

sub verificar{#verificar si son entradas o salidas
  if($comprobar eq 'sal') { salidas(); $conexion="2";
$archivosal="bediphda";}
  if($comprobar eq 'entra') { entradas(); $reportet="2";
$archivoen="bediphda";}
}

sub imprimir{  #imprime las gráficas y el texto
 $psi = $canvas->postscript(
       -width => $canvas->Width,
       -height =>$canvas->Height);
$impresion="./TEX/temporal.gif";
open (PS, ">$impresion");
       print PS $psi;
       close (PS);
my $impre='';
my $top= $wPrincipal->Toplevel;
$top->title("Imprimir"); $top->iconname('Imprimir');
$top->Label(-text => "Introducir nombre de la Impresora") -
>pack(-side => 'top');
my  $frame=$top->Frame(-relief => 'flat', -borderwidth =>5);
$frame->pack(-fill=>'both', -expand=>'both',-side => 'left');
my $frame2=$top->Frame(-relief=>'flat', -borderwidth =>5);
$frame2->pack(-fill=>'both',    -expand    =>'both',    -side
=>'right');
$frame->Label(-text  =>'Impresora:')->pack(-side  =>'top',  -
fill =>'x');
my      $Entroimpre=$frame2->LabEntry(-width           =>40,-
labelPack=>[qw/-side left -anchor w/],
-textvariable => \$impre)->pack(qw/-side top/);
my $ejecutar=$frame2->Button(-text =>'Imprimir') ->pack(-side
=>'left');
my $comando =sub{hacerimpre($impre)};
$ejecutar->configure(-command => $comando);
$Entroimpre->bind('<Return>' => $comando);
$frame2->Button(-text  =>'Salir',  -command  =>sub{  destroy
```

```
$top})->pack(-side =>'right');

}
sub hacerimpre{
  my ($impre)=@_;
if(($reportet eq '4' && $archivoen eq 'todos') || ($reportet
eq  '5' && $archivoen eq 'todos') || ($reportet eq '6' &&
$archivoen eq 'todos')|| ($conexion eq '4' && $archivosal  eq
'todos'))
{
 if((system("lpr   -P $impre texto.txt"))>0) { Error('7');}
}
else
{
 if((system("lpr -P   $impre $impresion texto.txt")) >0) {
Error('7'); }
}
}
sub error{
my ($numfalla)=@_;
case3:{
if($numfalla eq '7'){ $error_texto="Impresora no valida";last
case3;}
}
my $DialogoError=$wPrincipal->Dialog(
  -title  =>  'Error',-text  =>  "$error_texto",-bitmap  =>
'warning', -buttons =>['OK'],);
my $buttone = $DialogoError->Show('-global');
}

MainLoop;
##################################################################
##################################################################
                          #FIN DEL PROGRAMA
##################################################################
##################################################################
```

6. PROGRAMA SCAU: ayuda.pl

```
#!/opt/TWWfsw/bin/perl -w
require 5.004;
use English;
use Tk;
use Tk::widgets qw/Dialog ErrorDialog/;
use subs qw/llamada lbusqueda mu_estado /;
use vars qw/$VP $FUENTE $DIRARCH/;
use strict;
$VP = new MainWindow;
$VP->title('AYUDA DE SCAU');
$FUENTE = '-*-Helvetica-Medium-R-Normal--*-140-*-*-*-*-*-*';
my $menubar = $VP->Frame;
$menubar->grid(qw/-sticky ew/);
$menubar->gridColumnconfigure(qw/0 -weight 1/);
my $about = $menubar->Menubutton(qw/-text Acerca -underline 0
-menuitems/ =>
    [
     [Button    => '~Acerca'],
    ])->grid(qw/-row 0 -column 1 -sticky w/);

my $T = $VP->Scrolled('Text',
    -scrollbars => 'e',
    -wrap       => 'word',
    -width      => 60,
    -height     => 20,
    -font       => $FUENTE,
    -setgrid    => 1,
)->grid(qw/-sticky nsew/);
$T->bindtags(qw/widget_demo/);     # remove all bindings but
dummy "widget_demo"

my $EST_VAR;
my $status = $VP->Label(-textvariable => \$EST_VAR, qw/-
anchor w/);
$status->grid(qw/-sticky ew/);

$T->tag(qw/configure title -font -*-Helvetica-Bold-R-Normal--
*-180-*-*-*-*-*-*/);
$T->tag(qw/configure demo -lmargin1 1c -lmargin2 1c -
foreground blue/);

if ($VP->depth  == 1) {
```

```
    $T->tag(qw/configure hot -background black -foreground
white/);
    $T->tag(qw/configure visited -lmargin1 1c -lmargin2 1c -
underline 1/);
} else {
    $T->tag(qw/configure hot -relief raised -borderwidth 1 -
foreground red/);
    $T->tag(qw/configure visited -lmargin1 1c -lmargin2 1c -
foreground/ =>
        '#303080');
}
$T->tag(qw/bind demo <ButtonRelease-1>/ => sub {llamada $T-
>index('current')});
my $last_line = '';
$T->tag(qw/bind demo <Enter>/ => [sub {
     my($texto, $sv) = @ARG;
     my $e = $texto->XEvent;
     my($x, $y) = ($e->x, $e->y);
     $last_line = $texto->index("\@$x,$y linestart");
     $texto->tag(qw/add hot/, $last_line, "$last_line
lineend");
     $texto->configure(qw/-cursor hand2/);
     mu_estado $sv, $texto, $texto->index('current');
    }, \$EST_VAR]
);
$T->tag(qw/bind demo <Leave>/ => [sub {
     my($texto, $sv) = @ARG;
     $texto->tag(qw/remove hot 1.0 end/);
     $texto->configure(qw/-cursor xterm/);
     $$sv = '';
    }, \$EST_VAR]
);
$T->tag(qw/bind demo <Motion>/ => [sub {
     my($texto, $sv) = @ARG;
     my $e = $texto->XEvent;
     my($x, $y) = ($e->x, $e->y);
     my $new_line = $texto->index("\@$x,$y linestart");
     if ($new_line ne $last_line) {
         $texto->tag(qw/remove hot 1.0 end/);
         $last_line = $new_line;
         $texto->tag(qw/add hot/, $last_line, "$last_line
lineend");
     }
     mu_estado $sv, $texto, $texto->index('current');
    }, \$EST_VAR]);
```

```
#Menú de la ayuda
$T->insert('end', "  Sistema Supervisor de Claves y Accesos
en UNIX\n", 'title');
$T->insert('end', " Acerca de SCAU \n", [qw/demo demo-
acerca_scau/]);
$T->insert('end', " Que es el SCAU? \n", [qw/demo demo-
quees_scau/]);
$T->insert('end', "  Instalacion del SCAU \n", 'title');
$T->insert('end', " Requisitos \n",[qw/demo demo-
inst_scau/]);
$T->insert('end', " Instalando SCAU \n",[qw/demo demo-
inst_scau/]);
$T->insert('end', "  Menu Archivo \n", 'title');
$T->insert('end', " Abrir \n",[qw/demo demo-arch_scau/]);
$T->insert('end', " Guardar \n",[qw/demo demo-arch_scau/]);
$T->insert('end', " Imprimir \n",[qw/demo demo-arch_scau/]);
$T->insert('end', " Salir \n",[qw/demo demo-arch_scau/]);
$T->insert('end', "  Menu Entradas \n", 'title');
$T->insert('end', " Accesos \n",[qw/demo demo-
entradas_scau/]);
$T->insert('end', " Accesos por usuario \n",[qw/demo demo-
entradas_scau/]);
$T->insert('end', " Accesos por dia \n",[qw/demo demo-
entradas_scau/]);
$T->insert('end', " Dominios extranos \n",[qw/demo demo-
entradas_scau/]);
$T->insert('end', " Horas no laborables \n",[qw/demo demo-
entradas_scau/]);
$T->insert('end', " Cuentas sin usar \n",[qw/demo demo-
entradas_scau/]);
$T->insert('end', " Cuenta mas utilizada \n",[qw/demo demo-
entradas_scau/]);
$T->insert('end', "  Menu Salidas \n", 'title');
$T->insert('end', " Accesos \n",[qw/demo demo-
salidas_scau/]);
$T->insert('end', " Accesos por usuario \n",[qw/demo demo-
salidas_scau/]);
$T->insert('end', " Accesos por dia \n",[qw/demo demo-
salidas_scau/]);
$T->insert('end', " Dominios extranos \n",[qw/demo demo-
salidas_scau/]);
$T->insert('end', "  Menu Grafica \n", 'title');
$T->insert('end', " Barras \n",[qw/demo demo-graf_scau/]);
$T->insert('end', " Histograma \n",[qw/demo demo-
graf_scau/]);
```

```
$T->insert('end', "  Menu Ayuda \n", 'title');
$T->insert('end', " Acerca de \n",[qw/demo demo-
ayuda_scau/]);
$T->insert('end', " Ayuda de SCAU \n",[qw/demo demo-
ayuda_scau/]);
$T->configure(-state => 'disabled');
# Create all the dialogs required by this demonstration.
my $dialo_sicu = $VP->Dialog(
    -title          => 'Acerca de SCAU',
    -bitmap         => undef,
    -default_button => 'OK',
    -buttons        => ['OK'],
    -text           => "       SCAU\n Berenice Cervantes \n
Javier Velasco");
$about->entryconfigure('Acerca', -command => [$dialo_sicu =>
'Show']);

$VP->Button(-text => 'Salir', -command => [\&exit])-
>grid(qw/-column 0 -sticky w/);
MainLoop;

sub llamada {
    # Cuando el usuario hace click en una linea se realiza
esta llamada.
    my($index) = @ARG;
    my @tags = $T->tag('names', $index);
    my $i = lbusqueda('demo\-.*', @tags);
    return if $i < 0;
    my($demo) = $tags[$i] =~ /demo-(.*)/;
    $T->tag('add', 'visited', "$index linestart",
                              "$index lineend -1 chars");
#Ejecuta otro programa en perl.
system ("./ayuda/ayuda2.pl $demo &");
} # fin llamada

sub lbusqueda {
    # busca en la lista buscando la expresión y retorna el
valor.
    my($regexp, @list) = @ARG;
    my($i);
    for ($i=0; $i<=$#list; $i++) {
        return $i if $list[$i] =~ /$regexp/;
    }
    return -1;
} # fin lbusqueda
```

```
sub mu_estado {
    my($sv, $texto, $index) = @ARG;
    my @tags = $texto->tag('names', $index);
    my $i = lbusqueda('demo\-.*', @tags);
    return if $i < 0;
    my($demo) = $tags[$i] =~ /demo-(.*)/;
    $$sv = "Button-1 para ayuda de \"$demo\".";
} # fin mu_estado
```

7. PROGRAMA SCAU: ayuda2.pl

```
#!/opt/TWWfsw/bin/perl -w
use strict;
use English;
use Tk;
use Tk::Dialog;

my $archivo = ($ARGV[0]);
my $VP = new MainWindow;
$VP->title('Informacion acerca de ...');

my $cont_1 = $VP->Frame(-relief => 'groove',
                        -borderwidth => 5)->pack(-fill =>
'both',
                                                 -expand =>
'both');


   open(ARC,"./ayuda/$archivo.txt");
   my  @lineas = <ARC>;
     chomp(@lineas);
   my $scroll=$cont_1->ScrlListbox(
               -scrollbars => 'se',
               -label      => " ",
               -font => '-*-Helvetica-Medium-R-Normal--*-
140-*-*-*-*-*-*',
               -height     => 20,
               -width      => 50);
   $scroll->insert('end', @lineas);
   $scroll->pack(-side => 'top', expand => 1, -fill =>
'both');

   my $bot=$cont_1->Button(
               -text => 'OK',
```

```
                -width => 2,
                -command => sub { exit},
    )->pack(-side => 'left');

MainLoop;
```

REFERENCIAS BIBLIOGRAFICAS

[Bach 86] Maurice J. Bach. *The Desing of the Unix Operating System.* Prentice Hall PRT, 1986.

[Frisch 95] Aeleen Frisch. *Essential System Administration.* O'Reilly & Associates, 1995.

[Garfinkel 91] Simson Garfinkel y Gene Spafford. *Practical UNIX Security.* O'Reilly, 1991.

[Kernighan 91] Brian W. Kernighan. *El Lenguaje de Programación C.* Prentice Hall, 1991.

[Leffler 89] Leffler S.J., McKusinck M.K., Karels M. J., Quarterman J.S. *The Desing and Implementation of the 4.3 BSD UNIX Operating System.* Addison-Wesley, 1989.

[Márquez 93] F.M. Márquez, *UNIX. Programación Avanzada. RA-MA,, 1993.*

[Medinets 97] Medinets David . *Perl 5 a través de Ejemplos.* Prentice Hall, 1997.

[Morgan 87] Rachel Morgan, Henry McHilton. *Introducing UNIX System V.* McGraw-Hill, 1987.

[Nemeth 95] Evi Nemeth. *UNIX System Administration Handbook.* Prentice Hall PTR, 1995.

[Pressman 93] Roger Pressman. *Ingeniería del Software.* McGraw-Hill, 1993.

[Raya 97] Jose Luis Raya, Victor Rodrigo. *Domine TCP/IP.* RA-MA, 1997.

[Sánchez 96] Sánchez Sebastián. *UNIX Guía del Usuario.* Alfaomega,1996.

[Schwartz 94] Randal L. Schwartz. *Learning Perl.* O'Reilly , 1994.

[Srinivasan 97] Sriram Srinivasan. *Advanced Perl Programming.* O'Reilly, 1997.

[Stevens 90] Richard Stevens. *UNIX Network Programming.* Prentice-Hall, 1990.

[Tanenbaum 88] Tanenbaum, Andrew S. *Sistemas Operativos. Diseño e Implementación.* Prentice Hall Hispanoamericana,1988. (Traducción de la obra Operating System: Desing and Implementation. Prentice Hall, Inc. 1987).

[Walsh 99] Nancy Walsh. *Learning Perl/Tk.* O'Reilly, 1999.

[Wylder 93] Floyd Wylder. *A Guide to TCP/IP Protocol Suite.* Artech House, 1993 .