Escuela de Administración de Negocios

E . A . N .

TALLER PRÁCTICO SOBRE

EL SISTEMA OPERATIVO

UNIX

POR

JULIO REY DE PEREA CAMPOS

DIPLOMADO EN INFORMÁTICA

UNIVERSIDAD DE ALMERÍA - ESPAÑA

BLOQUE 1: ESTRUCTURA DE UNIX.

TEMA 1. INTRODUCCIÓN.

CARACTERÍSTICAS GENERALES:

¿Qué es "UNIX"? En sentido estricto, es el núcleo de un sistema operativo de tiempo compartido: un programa que controla los recursos de una computadora y los asigna entre los usuarios. Permite a los usuarios ejecutar sus programas; controla los dispositivos periféricos (discos, terminales, impresoras y otros) conectados a la máquina; y proporciona un sistema de archivos que administra el almacenamiento a largo plazo de información tal como programas, datos y documentos.

En un sentido más amplio, "UNIX" abarca no sólo el núcleo, sino que incluye también programas esenciales, entre ellos: compiladores, editores, programas para copiado e impresión de archivos, etc. En un sentido más amplio todavía, "UNIX" puede incluir programas desarrollados por usuarios para ser ejecutados en el sistema; por ejemplo, herramientas para preparar documentos, rutinas para análisis estadísticos y paquetes gráficos. Cual de estos usos del nombre "UNIX" sea el correcto depende del nivel del sistema que se esté considerando.

Estas son algunas de las razones por las que el sistema operativo UNIX ha conseguido tanto éxito y popularidad:

- El sistema está escrito en un lenguaje de alto nivel, haciéndolo fácil de leer, comprender, cambiar y mover a otras máquinas.
- Tiene una interfaz con el usuario simple que tiene el poder de suministrar los servicios que quiere el usuario.
- Provee primitivas[1] que permite la realización de programas complejos a partir de programas más simples.
- Usa un sistema de archivos jerárquico que permite un fácil mantenimiento y una implementación eficiente.
- Usa un formato para los archivos consistente, el flujo de bytes, haciendo que los programas de aplicación sean fáciles de escribir.

[1] Las primitivas son comandos o instrucciones básicas del sistema.

- Provee una simple y consistente interfaz con los dispositivos periféricos.
- Es un sistema multiusuario y multitarea, cada usuario puede ejecutar varios procesos simultáneamente.
- Oculta la arquitectura de la máquina del usuario, haciendo fácil el escribir programas que corran en diferentes implementaciones hardware.

Además de que el sistema operativo y muchos de los programas y comandos están escritos en lenguaje C, UNIX soporta otros lenguajes, incluyendo Fortran, Basic, Pascal, Ada, Cobol, Lisp y Prolog. UNIX soporta cualquier lenguaje de programación que tenga un compilador o intérprete y una interfaz con el sistema que permita transformar las peticiones del usuario de servicios del sistema al conjunto estándar de peticiones usadas en el sistema UNIX

HISTORIA DEL UNIX:

En la década de los sesenta, en la cual se vivía un ambiente de computación en el cual no se soportaba sino un usuario y un programa ejecutándose, además de que el procesamiento de los datos era por lotes y la interacción con los usuarios era escasa, parecía ambicioso pensar en un ambiente de multiprogramación y multitarea, pero a finales de la década se concibió la idea y se trató de implementar en un sistema llamado MULTICS, desarrollado por Bell Laboratories junto con el MIT y General Electric; pero no funcionó y el grupo de investigadores que se encontraba desarrollando este proyecto se dispersó.

Este sistema Multics no prosperó, pero a partir de él, Ken Thompson, uno de los investigadores del proyecto MULTICS, y sus colegas construyeron el sistema operativo UNIX. La primera versión de UNIX fue escrita en 1969. Esta versión de UNIX se ejecutaba en un computador PDP-7. En 1970, Thompson, junto con Dennis Ritchie, lo transportó a un PDP-11/20. Ritchie diseñó y escribió además el primer compilador de C con objeto de ofrecer un lenguaje que pudiera usarse para escribir una versión transportable del sistema. En 1973, Ritchie y Thompson reescribieron en C el kernel de UNIX, el corazón del sistema operativo.

Las primeras licencias de UNIX se entregaron en 1974 a las universidades, con fines educativos, en una versión conocida como quinta edición. La sexta edición, también conocida como V6, fue liberada en 1976 y su distribución fue mucho más extensa que la quinta. La séptima edición, liberada por Bell Laboratories en 1978, fue la primera que tenía como principal objetivo la transportabilidad. Esta edición, implantada en los computadores DEC PDP-11, Interdata 8/32 y VAX, ha servido como punto de partida común para todo el mundo UNIX. Si hay una versión que defina al UNIX ”clásico”, es la séptima edición.

La liberación de la sexta y la séptima ediciones dio lugar a varios caminos independientes, pero no aislados, del desarrollo de UNIX. Los tres vástagos más influyentes de la séptima edición fueron las desarrolladas por AT&T Informational System como versiones de System V (no confundir con la quinta edición); los sistemas de investigación que Bell Laboratories siguió desarrollando, y los diversos sistemas BSD (Berkeley Software Distribution).

Entre los descendientes de System V están Xenix, de Microsoft; HP-UX, de Hewlett Packard; UTS, de Amdahl; AIX, de IBM; System V/386, de Interactive; e IRIX, de Silicon Graphics. Actualmente, SCO (Santa Cruz Operation) vende tanto Xenix como UNIX; SCO Xenix se distribuye por licencia de Microsoft y está basado en la primera versión de System V, mientras que SCO UNIX incluye las últimas novedades de desarrollo. SCO será la versión de UNIX con la que vamos a trabajar

Los descendientes principales de BSD UNIX son SunOS, de Sun Microsystems, y Ultrix, de Digital Equipment Corporation. Berkeley no ofrece mantenimiento para los sistemas que

desarrolla. Mt. Xinu ofrece mantenimiento comercial de BSD UNIX además de su propia versión de BSD UNIX.

A finales de los años ochenta se presentó un cambio de gran magnitud en el mundo UNIX, al incorporarse el sistema X-Windows al entorno UNIX. Este sistema, conocido en forma abreviada como X, ocasionó un cambio de los entornos de trabajo basados en caracteres a interfaces gráficas con el usuario.

Hoy día, UNIX es un sistema muy diferente de lo que fue a principios de los años setenta. En aquella época, el sistema representativo era un solo procesador que servía a un conjunto de terminales de teletipo conectadas al procesador a través de líneas telefónicas directas o conmutadas. El sistema representativo actual es una estación de trabajo con una pantalla de alta definición de mapa de bits que opera con un sistema de ventanas y participa activamente en una extensa red de computadores. En aquella época, UNIX era pequeño, sencillo y no comercial, destinado a un público reducido y selecto. Ahora, UNIX es un producto comercial importante, grande, complicado, que se utiliza en una amplia gama de aplicaciones, muchas veces por personas que no tienen experiencia de programación

EL PAPEL DE C EN UNIX:

Al hablar de UNIX muchas veces se menciona el lenguaje de programación C, pero para muchos usuarios que no piensan efectuar tareas de programación no queda clara la importancia de esta relación.

- Muchos programas de UNIX siguen los convencionalismos sintácticos de C. Por ejemplo, el lenguaje de programación *awk* utiliza varios de los operadores y estructuras de control que se emplean en C.
- La mayor parte del software de UNIX disponible en forma pública se distribuye como programas en C que deben compilarse antes de usarse. Esta tradición de UNIX surgió por la necesidad de distribuir programas en forma transportable para que funcionaran en distintos tipos de computadores.
- Las llamadas al sistema[2] UNIX se definen como funciones en C. Aunque estas llamadas al sistema conciernen principalmente a los programadores, en la documentación para el usuario aparecen referencias a ellas.

La mayor parte del sistema UNIX está escrito en C; es más, C se diseñó para apoyar a UNIX. Las organizaciones con licencias de los programas fuente de UNIX pueden modificar el comportamiento del sistema UNIX alterando y recompilando los programas fuente.

[2] Las llamadas al sistema son utilizadas por los programas para pedir servicios al kernel.

TEMA 2. DISEÑO DE UNIX.

ESTRUCTURA DEL SISTEMA:

En la figura 2.1 se describe la arquitectura a alto nivel del sistema UNIX. El hardware en el centro del diagrama provee al sistema operativo con servicios básicos tales como interrupciones y excepciones, niveles de ejecución (prioridades), administración de memoria, etc. El sistema operativo interactúa directamente con el hardware, suministrando servicios a los programas y librándoles de la idiosincrasia del hardware. El sistema operativo es denominado normalmente como sistema kernel, o simplemente kernel. Como los programas son independientes del hardware que hay por debajo, es fácil moverlos entre sistemas UNIX que corren en diferentes máquinas si los programas no hacen referencia al hardware subyacente.

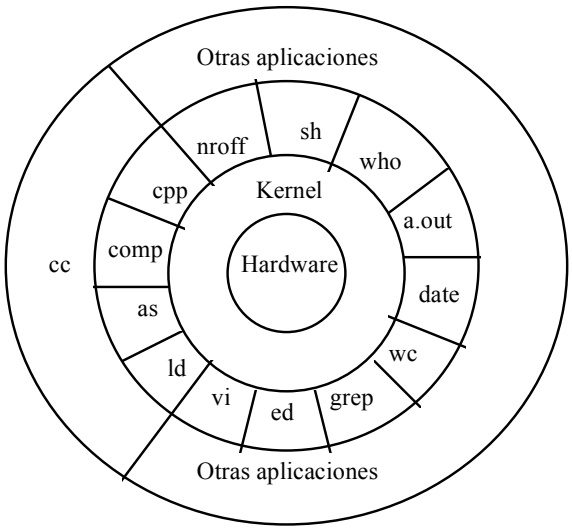
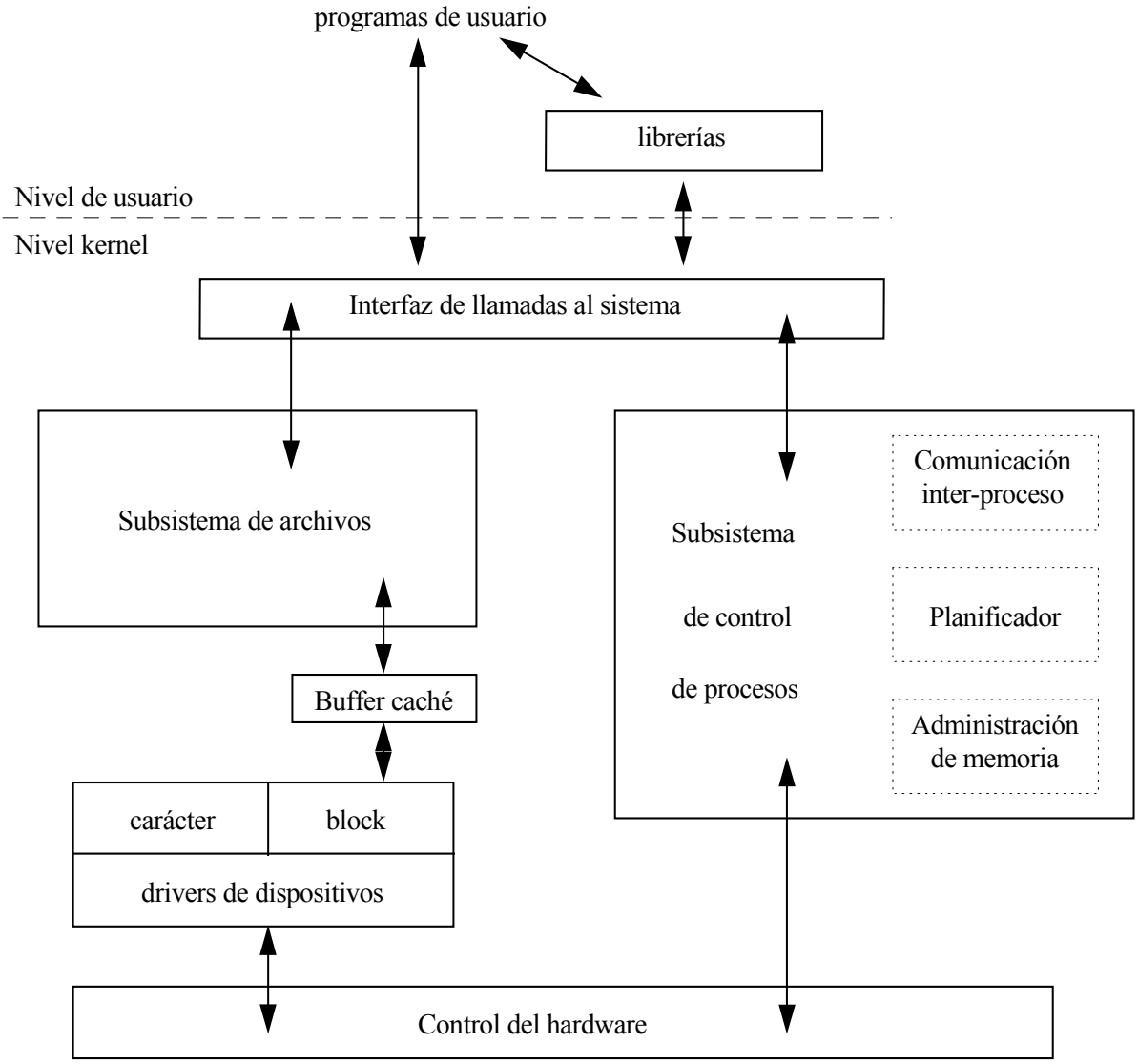


Figura 2.1: Arquitectura del sistema UNIX.

Programas como el shell y editores mostrados en niveles exteriores interactúan con el kernel por medio de un bien definido conjunto de llamadas al sistema. Las llamadas al sistema instruyen al kernel para hacer varias operaciones para el programa llamador e intercambiar datos entre el kernel y el programa. Muchos de los programas mostrados en la figura están disponibles en las configuraciones estándar del sistema y son conocidos como comandos, pero los programas de usuario pueden existir también en este nivel, indicado por el programa *a.out*, el nombre estándar para los archivos ejecutables producidos por el compilador de C. Otros programas de aplicación pueden construirse en el nivel más alto. Por ejemplo, el compilador de C, *cc*, está en el nivel más exterior de la figura: invoca al preprocesador de C, al compilador de dos pasos, al ensamblador y al enlazador, todos ellos programas de nivel inferior.

ARQUITECTURA DEL SISTEMA OPERATIVO UNIX:

Los dos conceptos centrales en el modelo del sistema UNIX son los archivos y los procesos. La figura 2.2 muestra un diagrama de bloques del kernel, mostrando varios módulos y las relaciones entre ellos. En particular, se muestra el subsistema de archivos en la parte izquierda y el subsistema de control de procesos en la parte derecha, los dos mayores componentes del kernel.



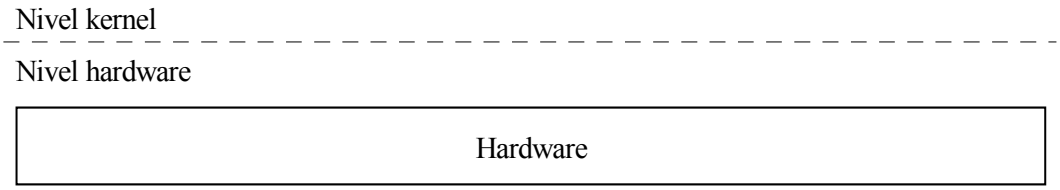


Figura 2.2: Diagrama de bloques del kernel del sistema.

La figura 2.2 muestra tres niveles: usuario, kernel y hardware. Las llamadas al sistema y las librerías representan el borde entre los programas de usuario y el kernel. Las llamadas al sistema son parecidas a las funciones en los programas en C y las librerías transforman estas funciones a las primitivas necesarias para entrar en el sistema operativo. Los programas en lenguaje ensamblador deben invocar a las llamadas al sistema directamente sin las librerías de las llamadas al sistema.

Las llamadas interactúan con el subsistema de archivos y con el subsistema de control de procesos. El subsistema de archivos controla los archivos, asigna espacio a los archivos, administra el espacio libre, controla el acceso a los archivos, etc. Los procesos interactúan con el subsistema de archivos mediante un específico conjunto de llamadas al sistema.

El subsistema de archivos accede a los archivos de datos usando un mecanismo de buffer que regula el flujo de datos entre el kernel y los dispositivos de almacenamiento secundario. El mecanismo de buffer interactúa con los controladores de dispositivos de E/S de tipo bloque para iniciar la transferencia de datos desde y hacia el kernel. Los controladores de dispositivos (device drivers) son los módulos del kernel que controlan las operaciones con los dispositivos periféricos. El subsistema de archivos además interactúa directamente con los controladores de dispositivos de E/S de tipo carácter sin la intervención de un mecanismo de buffer.

El subsistema de control de procesos es el responsable de la sincronización de los procesos, la comunicación entre procesos, administración de memoria principal y la planificación de procesos. El subsistema de archivos y el subsistema de control de procesos interactúan cuando se carga un archivo en memoria para su ejecución.

El módulo de administración de memoria controla la asignación de memoria. Si en algún momento no hay suficiente memoria física para todos los procesos, el kernel los mueve entre la memoria principal y la secundaria.

El módulo del planificador o scheduler asigna la CPU a los procesos. Planifica los procesos para ser ejecutados por turno hasta que voluntariamente liberen la CPU mientras esperan un recurso o hasta que el kernel los saca cuando su tiempo de ejecución supera el tiempo de cuantum.

Finalmente, el control del hardware es el responsable de las interrupciones y de las comunicaciones con la máquina. Los dispositivos como los discos o terminales pueden interrumpir a la CPU mientras un proceso se está ejecutando. Así, el kernel debe restablecer la ejecución del proceso interrumpido después de servir a la interrupción.

EL NÚCLEO O KERNEL DE UNIX:

El kernel de UNIX es el corazón del sistema operativo. Controla el acceso al computador y a sus archivos, asigna recursos a las distintas actividades que se llevan a cabo en el computador, mantiene el sistema de archivos y administra la memoria del computador. Aunque los usuarios comunes rara vez tienen una interacción explícita con el kernel, éste es un aspecto medular de UNIX.

La figura 2.1 nos muestra la capa del kernel inmediatamente por debajo de la capa de las aplicaciones de usuario. El kernel suministra varias operaciones primitivas que ayudan a los procesos del usuario a soportar la interfaz de usuario. Entre los servicios suministrados por el kernel están:

- Controlar la ejecución de procesos permitiendo su creación, finalización o suspensión y comunicación.
- Planificar los procesos para su ejecución en la CPU. El sistema es de tiempo compartido: la CPU ejecuta un proceso, el kernel lo suspende cuando termina su quantum de tiempo y el kernel planifica otro proceso para su ejecución. El kernel después replanifica el proceso suspendido.
- Asignar memoria principal a los procesos en ejecución. El kernel permite a los procesos compartir porciones de su espacio de direcciones bajo ciertas condiciones, pero protege el espacio de direcciones privado de un proceso de accesos exteriores. Si el sistema se encuentra bajo en memoria libre, el kernel puede liberar memoria escribiendo un proceso temporalmente en memoria secundaria, llamado área de swap. Si el kernel escribe el proceso completo en el área de swap, la implementación de UNIX se denomina sistema de swapping; si escribe páginas de memoria al área de swap, se denomina sistema paginado.
- Asignar memoria secundaria para un almacenamiento eficiente y reutilización de datos de usuario. Este servicio constituye el sistema de archivos o filesystem. El kernel asigna almacenamiento secundario a los archivos de usuario, controla zonas de memoria no usadas, estructura el sistema de archivos de forma fácil de comprender y protege los archivos de usuario de accesos ilegales.
- Permitir accesos controlados de los procesos a los dispositivos periféricos como terminales, unidades de cinta o disco y dispositivos de red.

Un administrador del sistema puede ajustar las características operativas del sistema mediante la configuración del kernel. La configuración del kernel es la acción de decirle a éste, de que tamaño deben ser las tablas que utiliza el UNIX para el manejo de sus recursos; dicho proceso se hace mediante la modificación de un archivo de parámetros. Después se toma el kernel y se enlaza con ciertas subrutinas y se genera un nuevo kernel con las nuevas entradas. Este proceso no queda operativo hasta que no se inicie de nuevo el equipo. Esta configuración solamente la puede realizar el administrador del sistema.

Por ejemplo, cuando se configura el kernel es el momento donde se determina el número de archivos que se pueden encontrar abiertos, el número de procesos que se pueden estar ejecutando a un mismo tiempo, y el tamaño de los buffers,…; todos estos recursos los maneja UNIX por medio de tablas y listas. El tamaño de estas tablas es definido por el administrador del sistema, por consiguiente este tamaño no se puede modificar en medio de la ejecución.

Cuando no se configura bien el kernel con una debida estructuración y se deja exceso de espacio para tablas se está desperdiciando la memoria, quitándole espacio a los procesos, debido a que tiene un kernel muy grande.

TEMA 3: ESTRUCTURA DEL SISTEMA DE ARCHIVOS (FILE SYSTEM).

EL SISTEMA DE ARCHIVOS:

El sistema de archivos o file system es la forma en el que UNIX gestiona los archivos existentes en el sistema. El sistema de archivos está caracterizado por:

- Una estructura jerárquica.
- Un tratamiento consistente de los archivos de datos.
- Habilidad para crear y borrar archivos.
- Crecimiento dinámico de los archivos.
- Protección de los archivos de datos.
- Tratamiento de los dispositivos periféricos como archivos.

El sistema de archivos está organizado como un árbol con un único nodo raíz llamado root (se representa con "/"); cada nodo que no es hoja de la estructura del sistema de archivos es un directorio de archivos. Cada archivo está identificado por un nombre de archivo. El nombre de archivo viene dado por un nombre de camino o ruta (path name) que describe como localizarlo en la jerarquía del sistema de archivos. Un nombre de camino es una secuencia de nombres de componentes separados por caracteres "/"; una componente es una secuencia de caracteres que identifica un archivo que está contenido en el componente precedente (directorio). Un camino completo comienza con un carácter "/" y especifica un archivo que puede ser encontrado empezando en el nodo raíz del sistema de archivos. Así, "/etc/passwd", "/bin/who" y "/usr/src/cmd/who.c" son caminos completos que identifican archivos en el árbol mostrado en la figura 3.1, pero "/bin/who" y "/usr/src/date.c" no. Un camino no tiene porqué empezar desde el root ya que pueden ser identificados de forma relativa al directorio actual o de trabajo de un proceso en ejecución, omitiendo el carácter "/" inicial en la ruta.

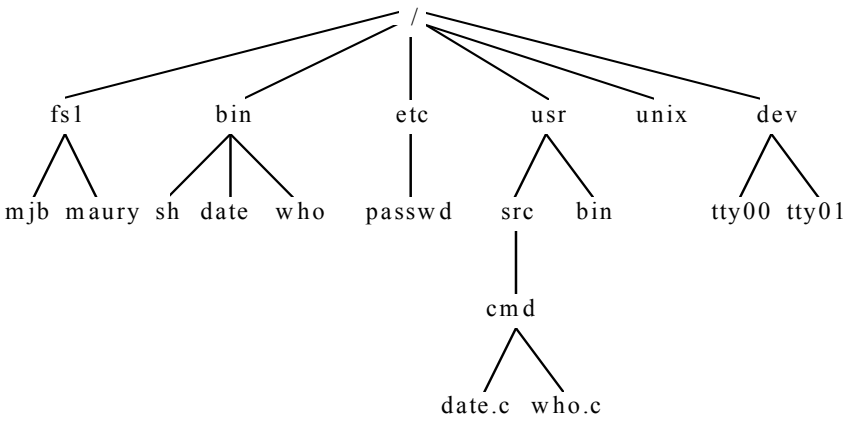


Figura 3.1: Ejemplo de árbol de un sistema de archivos.

Por ejemplo, si nos encontramos en el directorio src, la ruta cmd/date.c identifica un archivo existente en el árbol.

CONCEPTOS BÁSICOS SOBRE ARCHIVOS:

El sistema de archivos de UNIX consiste en un conjunto de archivos[3]. En UNIX, un archivo es una secuencia de bytes. El sistema no impone estructura alguna a los archivos, ni asigna significado a su contenido; el significado de los bytes depende únicamente de los programas que interpretan el archivo. Además, como veremos más adelante, esto es cierto no sólo para archivos en disco sino también para dispositivos periféricos. Cintas magnéticas, mensajes de correo, caracteres tecleados, salida para impresora, datos que fluyen en interconexiones, cada uno de estos archivos no es más que una secuencia de bytes desde el punto de vista del sistema y sus programas.

Hay tres tipos de archivos:

- Archivos ordinarios, que contienen datos.
- Archivos especiales, que permiten el acceso a dispositivos como terminales e impresoras y que también tienen otros propósitos.
- Directorios, que contienen información sobre un conjunto de archivos y que se emplean para localizar un archivo por su nombre.

La representación interna de un archivo viene dado por un inodo, el cual contiene una descripción del almacenamiento físico del archivo e información acerca del propietario del archivo, permisos de acceso y tiempos de acceso. Será descrito más adelante. Todos los archivos tiene un inodo, pero puede tener varios nombres, todos ellos accederían al mismo inodo. Cada nombre se denomina enlace. Cuando un proceso referencia un archivo por su nombre, el kernel analiza el nombre de archivo de cada componente, comprueba que el proceso tiene permiso para buscar los directorios en la ruta y recupera el inodo correspondiente al archivo. Por ejemplo, si un proceso realiza la llamada

open (“/fs2/mjb/rje/sourcefile”,1);

el kernel busca el inodo de “/fs2/mjb/rje/sourcefile”. Cuando un proceso crea un nuevo archivo, el kernel le asigna un inodo libre. Los inodos se almacenan en el sistema de archivos, tan pronto como sea posible, pero el kernel los lee desde una tabla de inodos en memoria principal cuando manipula archivos.

En el UNIX tradicional un nombre de archivo es de máximo 14 caracteres. En el BSD y el System V Release 4 pueden ser hasta de 256 caracteres. Las extensiones no son separadas necesariamente por punto. Existen caracteres que pueden causar conflictos en el intérprete de comandos. Por lo tanto deben evitarse en un nombre de archivo. Estos son:

/, $, &, !, (,), |, <, >, @, ^, { , }, [, , *, ?, \ , <Space>, <Tab>, <Backspace>

No se deben usar tampoco caracteres invisibles (caracteres de control) ni caracteres de opción de comando (+, -).

[3] Estos archivos serán los programas, los comandos y los dispositivos; contendrán código fuente, datos, etc...

Un archivo está almacenado en bloques. Existen dos tipos de bloques:

- Bloque lógico: Se define como la unidad mínima de asignación de espacio de almacenamiento.
- Bloque físico: Se define como la mínima unidad de lectura y escritura de un disco. También se conoce como el sector de un disco. El tamaño estándar es de 512 bytes.

El bloque lógico es siempre igual o mayor que el bloque físico. Lo mínimo que el sistema operativo UNIX asigna a un archivo es un bloque lógico, así se tenga un archivo con un tamaño real menor al tamaño del bloque lógico.

Por ejemplo: si se define un bloque lógico de 1K, y se tiene un archivo que ocupa 1200 bytes, entonces a este archivo le corresponderían dos bloques lógicos, pero si nos damos cuenta siempre se va a tener una pérdida o desperdicio de memoria, al menos que el tamaño real del archivo sea múltiplo del bloque lógico que se le asigne.

ESTRUCTURA INTERNA DEL SISTEMA DE ARCHIVOS:

Los distintos sistemas de archivos tienen básicamente la misma estructura, que se muestra en la figura 3.2, la cual está formada por cuatro partes fundamentales:

- Bloque de arranque (boot block): Es el bloque 0 de todo sistema de archivos. Está reservado al programa de carga inicial (bootstrap) y contiene la información necesaria para arrancar el sistema. Aunque solo es necesario un bloque de arranque, todos los sistemas de archivos del sistema tienen un bloque de arranque (posiblemente vacío).
- Superbloque (super block): Es el bloque 1. Contiene la información más importante del sistema de archivos y contiene los siguientes campos:
  - El tamaño del sistema de archivos.
  - El número de bloques libres.
  - Una lista de los bloques libres disponibles.
  - El índice del siguiente bloque libre en la lista de bloques libres.
  - El tamaño de la lista de inodos.
  - El número de inodos libres.
  - Una lista de inodos libres.
  - El índice del siguiente inodo libre en la lista de inodos libres.
  - Un flag indicando que el súper bloque ha sido modificado.
  - Campos de bloqueo para las listas de bloques libres e inodos libres.
- Lista de inodos (inode list): Esta lista está formada por un número de bloques que contiene el número de inodos especificado en el superbloque, que varía dependiendo del número total de bloques que tenga el sistema de archivos. La longitud de la lista de inodos se determina en el momento de creación del sistema de archivos.
- Bloques de datos (data block): El resto del dispositivo lógico está formado por los bloques de datos. Son matrices unidimensionales y contienen los datos realmente almacenados en directorios, archivos y bloques de punteros a bloques de datos y a bloques de la lista libre.

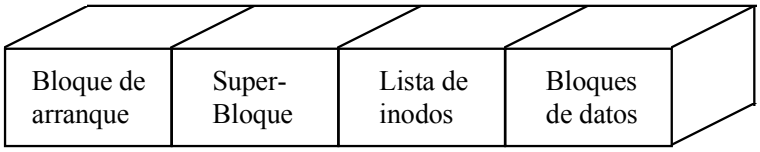


Figura 3.2: Estructura del sistema de archivos.

DESCRIPCIÓN DEL INODO:

Como hemos visto anteriormente, los sistemas de archivos en UNIX utilizan una estructura en disco, denominado inodo, para almacenar información sobre cada archivo. El contenido de un inodo, mostrado en la figura 3.3, es el siguiente:

- Modo: Su tamaño es de 2 bytes. Incluye el tipo de archivo y permisos de acceso. Se dedica 4 bits al tipo de archivo, 9 a los permisos de acceso y 3 a los permisos especiales.
- Cuenta de enlaces: Representa el número de nombres que tiene el archivo en la jerarquía de directorios.
- Identificador del usuario propietario (valor decimal).
- Identificador del grupo del propietario (valor decimal).
- Tamaño del archivo (en bytes).
- Direcciones de bloque: Su tamaño es de 40 bytes. Su utilización depende del tipo de archivo:
  - Ordinarios y directorios: Contiene 13 punteros (10 punteros directos, un indirecto simple, un indirecto doble y un indirecto triple) como se describe en la figura 3.4.
  - Especiales: Sólo se utilizan 4 bytes y en lugar de contener los punteros a bloques contienen el número principal (major) y secundario (minor) del dispositivo[4].
- Fecha de acceso: Fecha y hora en que fue accedido por última vez (lectura/escritura).
- Fecha de modificación: Fecha y hora de la última vez que fue modificado.
- Fecha de cambio de inodo: Fecha y hora en que el inodo del archivo fue modificado (por ejemplo, cambio en los permisos de acceso).

Modo
Cuenta de enlace
ID. de usuario
ID. de grupo
Tamaño del archivo
Direcciones de bloque
Fecha de acceso
Fecha de modificación
Fecha de cambio de inodo

Figura 3.3: Estructura de un inodo.

[4] Estos números serán descritos más adelante.

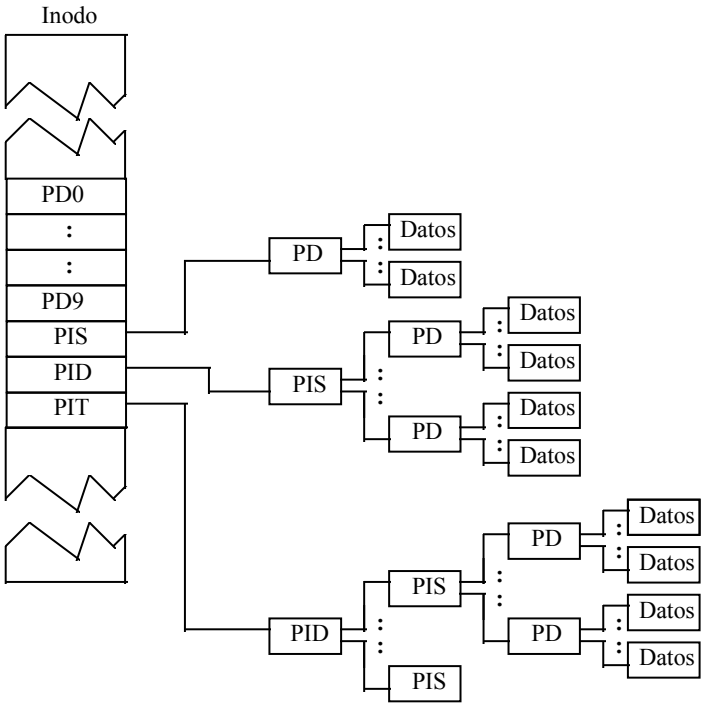


Figura 3.4: Organización de los punteros a bloques de un inodo.

DIRECTORIOS:

Como hemos visto, los directorios son los archivos que proporcionan al sistema de archivos su estructura jerárquica; ellos juegan un importante papel en la conversión de un nombre de archivo a un número de inodo. Un directorio es un archivo cuyos datos son una secuencia de entradas, cada una de ellas constituida por un número de inodo y el nombre de un archivo contenido en el directorio. Los bloques de almacenamiento de los directorios en los primeros sistemas de archivos que se diseñaron tienen el formato mostrado en la figura 3.5.

Inodo	Nobre del archivo
20	.
2	..
45	datos
53	fich
:	:

Figura 3.5: Contenido de un directorio.

Cada entrada en el directorio ocupa 16 bytes; de los cuales, dos corresponden al número de inodo y 14 al nombre del archivo. Como se puede observar, todo directorio comienza con dos entradas como mínimo: un identifica al directorio actual ('.') y la otra al directorio padre ('..').

Existen otros sistemas de archivos que permiten nombres de archivos de hasta 255 caracteres, como es el caso de los sistemas de archivos *ufs*.

DISPOSITIVOS Y ARCHIVOS ESPECIALES:

Cuando un programa accede a un determinado dispositivo mediante una llamada al sistema, establece una comunicación con el driver correspondiente al dispositivo usado. Esta comunicación no es directa, existe un punto de unión entre el programa y el driver correspondiente. En la figura 3.6 se representa esta relación.

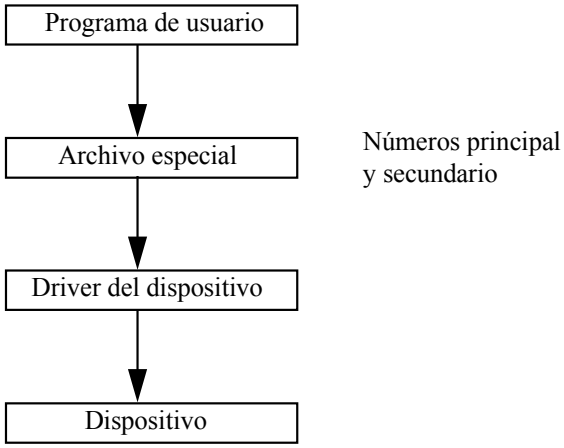


Figura 3.6: Relación entre los programas de usuario y los dispositivos periféricos.

Todos los dispositivos en UNIX son tratados como archivos. Estos archivos especiales (device special file), conocidos también como nodos de dispositivos (device nodes) constituyen el medio de acceso a los dispositivos. Entre los dispositivos que aparecen se encuentran los dispositivos de cinta, impresoras, particiones de disco y terminales. Básicamente existen dos tipos de archivos especiales:

- Archivos especiales tipo bloque: Se emplean para manejar dispositivos cuya unidad mínima de tratamiento de información es el bloque.
- Archivos especiales tipo carácter: Se emplean para dispositivos cuya unidad mínima de tratamiento de información es el carácter. También se engloban dispositivos que utilizan un tamaño distinto del bloque.

Estos archivos en realidad están vacíos. El sistema emplea dos números enteros, denominados número principal o mayor (14 bits) y un número secundario o menor (18 bits) y almacenados en el inodo del archivo, para acceder al dispositivo asociado.

El número principal identifica una clase de dispositivo (en realidad identifica al driver de dicha clase como pueden ser terminales, impresoras, discos, etc.) y el número secundario identifica a un elemento de dicha clase (un terminal específico, un disco concreto, ...).

ACCESO A LOS ARCHIVOS:

Desde el punto de vista del acceso a un archivo, existen tres tipos de usuarios a los que se les pueden dar o denegar permisos sobre un archivo:

(u)	user	Propietario del archivo.
(g)	group	Usuarios pertenecientes al grupo del propietario.
(o)	others	Resto de usuarios que no pertenecen al grupo.

La capacidad de un usuario para trabajar con archivos depende del tipo de acceso que tenga a dicho archivo. Los accesos disponibles en UNIX son:

Para un archivo:

- Permiso de lectura (r): Permite ver el contenido del archivo.
- Permiso de escritura (w): Permite cambiar el contenido del archivo.
- Permiso de ejecución (x): Permite ejecutar un archivo (como cualquier orden de UNIX).

Para un directorio:

- Permiso de lectura (r): Permite ver los nombres de los archivos de un directorio. Si se quiere información detallada sobre dichos archivos el directorio tiene que tener el permiso de ejecución para dicho usuario.
- Permiso de escritura (w): Permite cambiar el contenido de dicho directorio; crear nuevos archivos, suprimir los existentes (este último caso depende de los permisos de escritura de los propios archivos).
- Permiso de ejecución (x): Se debe de hablar más bien de permiso de búsqueda ya que permite situarse en dicho directorio y según el resto de los permisos, permitirá crear, borrar, modificar o copiar archivos.

Además de los permisos de acceso rwx para el propietario del archivo, grupo al que pertenece el propietario y resto de usuarios, existen tres permisos especiales que afectan cuando se emplea el archivo como programa ejecutable. Estos modos sólo se aplican a archivos ejecutables y sólo el superusuario puede fijarlos. Estos son:

- set-uid Permite fijar el identificador de usuario (valor octal 4000), e indica que cuando el programa se ejecuta, el identificador de usuario pasa a ser el del propietario del archivo.
- set-gid Permite fijar el identificador de grupo (valor octal 2000), e indica que cuando el programa se ejecuta, el identificador de grupo pasa a ser el del grupo propietario del archivo.
- sticky bit Se denomina “bit de adherencia”(valor octal 1000) y se aplica a programas que son compartibles por muchos usuarios. Con el bit de adherencia se consigue que el programa no abandone el espacio de intercambio (swap) aunque nadie lo esté utilizando. Se suele aplicar a programas de uso intensivo para mejorar sus tiempos de respuesta.

Los dos primero permisos son de utilidad para programas, como mail, que deben crear archivos en directorios no necesariamente poseídos por la persona que ejecuta el programa.

TEMA 4: PROCESOS.

ENTORNO DE PROCESAMIENTO:

Un programa es un archivo ejecutable, y un proceso es una instancia de un programa en ejecución y consiste en un conjunto de bytes que la CPU interpreta como instrucciones máquina, datos y pila. Se pueden ejecutar muchos procesos simultáneamente en sistemas UNIX (esta característica es denominada multiprogramación o multitarea) sin un límite lógico en su número. Además, varias instancias de un mismo programa pueden existir simultáneamente en el sistema. Para ello, el kernel los planifica para su ejecución. Un proceso puede leer o escribir en sus propias secciones de datos y pila, pero no puede hacerlo con los datos y pila de otros procesos. Los procesos se comunican con otros procesos y con el resto del mundo por medio de las llamadas al sistema.

En términos prácticos, un proceso en UNIX es la entidad que es creada por la llamada al sistema *fork*. Todos los procesos excepto el proceso 0 es creado cuando otro proceso ejecuta la orden *fork*. El proceso que invoca *fork* se denomina proceso padre y el nuevo proceso creado es el proceso hijo. Todos los procesos tienen un único proceso padre, pero un proceso puede tener varios procesos hijos. El kernel identifica cada proceso por su número de proceso, llamado process ID (PID). El proceso 0 es un proceso especial que es creado "a mano" cuando el sistema arranca; después de crear un proceso hijo (proceso 1), el proceso 0 se convierte en el proceso swapper[5]. El proceso 1, conocido como proceso *init*, es el antecesor de todos los demás procesos del sistema.

NIVELES DE EJECUCIÓN:

La ejecución de los procesos de usuario en UNIX está dividido en dos niveles: usuario y kernel. Cuando un proceso ejecuta una llamada al sistema, el modo de ejecución del proceso cambia del modo usuario a modo kernel: el sistema operativo ejecuta y atiende el servicio que el usuario requiere, devolviendo un código de error si falla la llamada. Muchas arquitecturas (y sus sistemas operativos) soportan más niveles que los dos descritos aquí, pero los dos niveles, usuario y kernel, son suficientes para el sistema UNIX. Las diferencias entre los dos modos son:

- Los procesos en modo usuario pueden acceder a sus propias instrucciones y datos pero no a las instrucciones y datos del kernel (o las de otros procesos). Los procesos en modo kernel, sin embargo, pueden acceder a las direcciones del kernel y de los usuarios.
- Algunas instrucciones máquinas están privilegiadas y producen error cuando se ejecutan en modo usuario.

[5] El proceso swapper se encarga de intercambiar procesos desde la memoria principal al área de swap y viceversa

ESTADOS DE UN PROCESO:

El tiempo de vida de un proceso puede estar conceptualmente dividido en un conjunto de estados que describen al proceso. La siguiente lista contiene el conjunto completo de estados:

1. El proceso está ejecutándose en modo usuario.
2. El proceso está ejecutándose en modo kernel.
3. El proceso no está ejecutándose pero está listo para correr tan pronto como el kernel lo planifique.
4. El proceso está dormido y reside en memoria principal.
5. El proceso está preparado para correr, pero el proceso swapper (proceso 0) debe cambiar el proceso a memoria principal antes de que el kernel pueda planificarlo para su ejecución.
6. El proceso está dormido y el swapper a movido el proceso a memoria secundaria para hacer sitio para otros procesos en memoria principal.
7. El proceso es devuelto del modo kernel al modo usuario, pero el kernel le baja la prioridad e intercambia el contexto para planificar otro proceso.
8. El proceso es creado y está en un estado transitorio; el proceso existe pero no está preparado para correr, tampoco está dormido. Este es el estado inicial para todos los procesos excepto el proceso 0.
9. El proceso ejecuta la llamada al sistema *exit* y está en un estado zombie. El proceso no existirá por mucho tiempo, pero deja información acerca del código de error y algunas estadísticas para su proceso padre. El estado zombie es el estado final de un proceso.

La figura 4.1 muestra un diagrama completo de los estados de los procesos y sus transiciones. El proceso entra en el modelo de estados en el estado de "creación" cuando el proceso padre ejecuta la llamada al sistema *fork* y eventualmente se mueve a un estado donde está listo para correr (3 o 5). Por simplicidad, asumimos que el proceso entra en el estado "listo para correr en memoria". El proceso planificador o scheduler[6] pondrá el proceso a ejecutar, y el proceso entra en el estado "ejecución en modo kernel", donde completará su parte de la llamada al sistema *fork*.

Cuando el proceso completa la llamada al sistema, se mueve al estado de "ejecución en modo usuario". Después de un periodo de tiempo, el reloj del sistema interrumpe el proceso y entra en el estado de ejecución en modo kernel otra vez. Cuando la menejador de la interrupción del reloj termina de servir la interrupción de reloj, el kernel decide planificar otro proceso a ejecutar, así el primer proceso entra en el estado de "espera" y el otro proceso empieza a ejecutarse. El estado de "espera" es realmente el mismo que el estado de "listo para correr en memoria", pero están representados separados para expresar que un proceso ejecutándose en el modo kernel puede ser pasado a modo "espera" sólo cuando va a retornar a modo usuario. Consecuentemente, el kernel puede llevar al área de swap un proceso en el estado de "espera" si es necesario. En cualquier momento, el scheduler eligirá el proceso para volver a ejecución y retornará al estado de "ejecución en modo usuario" otra vez.

Cuando un proceso ejecuta una llamada al sistema, deja el estado de "ejecución en modo usuario" y entra en el estado de "ejecución en modo kernel". Suponemos que el proceso realiza una operación de E/S y debe esperar a que la operación se complete. Entra en el estado de "dormido en memoria" quedándose bloqueado hasta que se le notifique que la operación de E/S esté completa.

[6] El proceso planificador o scheduler se encarga de planificar los procesos, es decir, es el encargado de elegir el proceso que va a ser ejecutado a continuación por la CPU.

Cuando la operación termina, el hardware interrumpe a la CPU y el manejador de interrupción desbloquea el proceso, causando su entrada en el estado se "listo para correr en memoria".

Supongamos que el sistema está ejecutando tantos procesos simultáneamente que no pueden guardarse todos en memoria principal, y el proceso swapper guarda el proceso en el área de swap para hacer sitio a otros procesos que están en el estado de "listo para correr guardado". Cuando desaloja la memoria principal, el proceso entra en el estado de "listo para correr guardado". Eventualmente, el swapper elige el proceso con mayor prioridad del área de swap y lo pasa a la memoria principal. y el proceso regresa al estado de "listo para correr en memoria". El scheduler después eligirá el proceso y lo enviará al estado de "ejecución en modo kernel".

Cuando un proceso termina, invoca la llamada al sistema *exit*, entra en el estado de "ejecución en modo kernel" y, finalmente, en el estado "zombie".

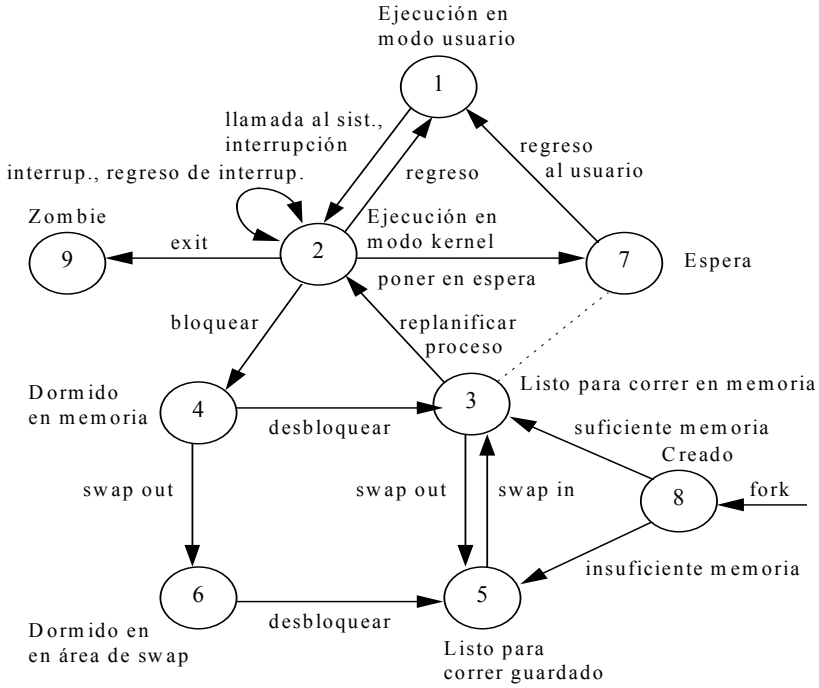


Figura 4.1: Diagrama de transición entre estados de un proceso.

BLOQUE 2: UNIX A NIVEL DE USUARIO.

TEMA 1: USUARIOS Y GRUPOS.

TIPOS DE USUARIO:

Para usar un sistema UNIX es necesario estar registrado como uno de sus usuarios. Cada usuario tiene un nombre de ingreso al sistema, una contraseña y un área del sistema de archivos reservada para almacenar sus archivos.

Básicamente, existen dos tipos de cuentas de usuarios:

- Cuentas de usuarios ordinarios: Son las más comunes. Contienen la siguiente información:
  - Un nombre de usuario (login).
  - Una clave de acceso o palabra de paso (password).
  - Un identificador de grupo.
  - Un directorio personal (home directory, $HOME)
  - Un shell de conexión o de ingreso (login shell).
- Cuenta de superusuario: Cualquier sistema UNIX debe tener un superusuario. Es la persona encargada de administrar el sistema; por ejemplo se encarga de crear nuevos usuarios, configurar dispositivos, ... El superusuario tiene el nombre "root" como nombre de usuario. Se identifica porque en pantalla aparece un prompt diferente al de un usuario normal (normalmente el carácter "#"). El superusuario tiene acceso a todos los archivos y directorios del sistema. En el bloque 3 se explicarán sus funciones.

El archivo /etc/passwd tiene información acerca de las cuentas de usuarios existentes en el sistema. Este archivo sólo puede ser modificado por el superusuario.

GRUPOS:

El sistema UNIX proporciona un entorno particularmente bueno para grupos de personas (usuarios) que trabajen conjuntamente en el mismo proyecto o proyectos relacionados. Cada usuario pertenece a un grupo. Algunos sistemas permiten que un usuario pertenezca a más de un grupo, y no es necesario que un grupo tenga más de un usuario. El grupo tendrá asignado un nombre de grupo e identificador de grupo. El grupo se establece al crear la cuenta de un usuario.

Cada usuario es conocido en el sistema de forma individual o como miembro de un grupo. Como miembro de un grupo, un usuario puede tener permiso para acceder a archivos y directorios a los que no tendría de forma personal.

El archivo /etc/group tiene información acerca de los grupos existentes en el sistema. Este archivo sólo puede ser modificado por el superusuario.

Estos archivos, el /etc/passwd y el /etc/group, serán descritos con mayor detalle en el bloque 3.

TEMA 2: ACCESO AL SISTEMA.

CONEXIÓN AL SISTEMA:

Para poder entrar en una cuenta aparece el llamado system login prompt:

```
eancol

Welcome to SCO UNIX System V/386 Release 3.2

eancol!login:
```

donde "eancol" es el nombre del host o hostname. Se introduce el nombre de usuario (también conocido como login name, user id o account). A continuación se nos pedirá la palabra de paso o clave de acceso:

```
Password:
```

y aparecerá información sobre la última conexión al sistema y revisión de SCO:

```
Last      successful login for <user>: <fecha> <hora> on <terminal>
Last unsuccessful login for <user>: <fecha> <hora> on <terminal>
SCO UNIX System V/386 Release 3.2
Copyright © 1976-1990 UNIX System Laboratories, Inc.
Copyright © 1980-1989 Microsoft Corporation
Copyright © 1983-1993 The Santa Cruz Operation, Inc.
All Rights Resered
eancol

                Welcome to SCO UNIX System V/386 Release 3.2

                From

                The Santa Cruz Operation, Inc.

Terminal type is ansi
```

Después se ejecutaría un archivo de configuración existente en el home directory del usuario. Cada shell tiene sus propios archivos de configuración.

El shell Bourne: El archivo de configuración que utiliza es:

- .profile Órdenes que se ejecutan antes del shell.

El shell C: Los archivos de configuración que utiliza son los siguientes:

- .cshrc Se ejecuta inicialmente. Contiene órdenes.

- .login Se ejecuta después que el archivo anterior y contiene órdenes para especificar el tipo de terminal y las variables de entorno.
- .logout Órdenes que se ejecutan cuando se desea abandonar el shell.

Después se ejecutaría el shell de ingreso y aparecerá el prompt correspondiente.

DESCONEXIÓN:

Se utiliza cuando se termina el trabajo con UNIX. Se conoce como “logout” o “log off”. Se realiza escribiendo *logout* o *exit*. En otros sistemas UNIX, también se puede mediante <Ctrl>-d (fin de archivo).

TEMA 3: EL SHELL.

¿QUÉ ES EL SHELL?:

Un shell o intérprete de comandos es un programa que interpreta y ejecuta los mandatos conforme se proporcionan desde la terminal. No se requiere ningún privilegio especial para ejecutar un shell; para el kernel de UNIX, un shell es como cualquier otro programa. Entre las características más comunes de un shell están la interpretación de procedimientos de shell, la expansión de caracteres comodines en nombres de archivos, la combinación de mandatos para formar interconexiones, la recuperación de mandatos previos, las construcciones condicionales y los ciclos, y las variables para crear abreviaturas.

El shell que se invoca cuando se ingresa al sistema se denomina shell de ingreso. El shell de ingreso de cada usuario se especifica en el campo correspondiente en el archivo /etc/passwd. El administrador del sistema que establece la cuenta de un usuario define su shell de ingreso y es el único que puede cambiarlo. Sin embargo, un usuario puede cambiar su shell de ingreso si coloca el mandato

exec *shell*

al final de su archivo de iniciación de ingreso. Este mandato transfiere el control de su terminal específicamente al shell *shell*. También se puede cambiar de shell durante una sesión, introduciendo el nombre del shell. Por ejemplo: *csh, ksh,* etc...

A continuación comentaremos algunas características de los shells más utilizados.

EL SHELL BOURNE (*sh*):

El shell Bourne, *sh*, escrito por Steve Bourne en 1979, es parte de la séptima edición de UNIX y el primero de los shells principales. Los shells más nuevos son más sencillos de usar porque ofrecen recursos de los que carecía el shell Bourne, como la edición de líneas de mandatos, la recuperación de mandatos emitidos previamente y los alias para los mandatos de uso común. No obstante, muchos usuarios de UNIX prefieren el shell Bourne para uso interactivo. Casi todos los procedimientos de shell siguen los convencionalismos del shell Bourne.

Entre los recursos importantes que ofrece *sh* están los siguientes:

- Operadores para la ejecución en segundo plano, o ejecución condicional de mandatos.
- Enunciados para repetir la ejecución de mandatos, incluida la iteración a lo largo de una secuencia de valores que pueden asignarse a una variable de iteración.
- Variables sustituibles, tanto nombradas como numeradas. Las variables numeradas, también conocidas como parámetros o parámetros de posición, contienen los argumentos de un mandato.
- Exportación de variables específicas a un proceso hijo.
- Tres formas de entrecomillado.
- Ejecución de mandatos en subshells.
- Notificación automática de la llegada de correo.

- Inclusión de datos de entrada para un mandato en un procedimiento de shell como parte del procedimiento.
- Atrapado de señales y ejecución de mandatos específicos cuando ocurre una señal determinada.
- Ejecución de mandatos en archivos de iniciación antes de leer cualquier entrada. Estos archivos de iniciación pueden servir para adecuar *sh* a las necesidades propias.

EL SHELL C (*csh*):

El shell C, disponible a través del mandato *csh*, se desarrolló como parte de BSD UNIX. A pesar de su nombre, el shell C no es mucho más parecido a C que el shell Bourne. Algunas de las características del shell C que no se incluyen en el shell Bourne son:

- La posibilidad de recuperar mandatos previos mediante un mecanismo de "historia".
- La capacidad de conmutar entre procesos y controlar su avance ("control de trabajos").
- Formas más flexibles de sustitución de variables.
- Operadores adicionales, como aparecen en C.
- Alias para mandatos de uso frecuente, sin tener que usar procedimientos de shell.

Varios de los mandatos en *csh* se comportan igual que sus contrapartes de *sh*.

EL SHELL KORN (*ksh*):

El shell Korn, *ksh*, ofrece una síntesis de las características de los shell Bourne y C, además de otras propias. Fue desarrollado por David Korn, de AT&T Bell Laboratories, en 1982, presentando versiones mejoradas en 1986 y 1988. Se incluye como característica estándar de la versión 4 de System V y otros sistemas; también se puede obtener por separado.

El shell Korn sigue de cerca los convencionalismos del shell Bourne, y casi todos los procedimientos de shell escritos para el primero funcionan con el segundo. Las características principales que se adoptaron del shell C son:

- Listas históricas para la recuperación de mandatos previos.
- Control de trabajos, con la capacidad para pasar trabajos específicos al primer plano o al segundo.
- Alias para los nombres de mandatos.
- Empleo de '~' para representar el directorio base del usuario o, al combinarse con un nombre de usuario, el de otro usuario.
- Capacidad para calcular expresiones numéricas generales y asignar el resultado a una variable.

Algunas de las características nuevas de *ksh* son:

- Edición interactiva de la línea de mandatos, incluida la complementación de nombres de archivo con las mismas características de *csh* y la posibilidad de editar la lista histórica.
- Mejores definiciones de funciones, que ofrecen variables locales y permiten escribir funciones recursivas.

- Comparación extendida de patrones para nombres de archivos y otras construcciones, parecidas a la de *egrep*.
- Capacidad para extraer la porción de una cadena especificada por un patrón.
- Capacidad para cambiar fácilmente de un directorio a otro.

EL SHELL C MEJORADO (*tcsh*):

El shell *tcsh* es una versión mejorada del shell C que ha adquirido mucha popularidad. Algunos de los recursos adicionales que ofrece son:

- Capacidad para editar la línea de mandatos interactivamente.
- Llamada sencilla de mandatos ejecutados con anterioridad, los cuales se pueden editar.
- Complementación interactiva de nombres de archivos y mandatos.
- Consulta de la documentación de un mandato al momento de teclearlo.
- Capacidad para programar la ejecución periódica de un mandato.
- Marcas de la hora en la lista histórica.

PROCEDIMIENTOS DE SHELL O SHELL SCRIPTS:

Un procedimiento de shell es un archivo que contiene secuencias de mandatos de shell, igual que si se hubieran tecleado. Los procedimientos de Shell, también llamados shell scripts, permiten adecuar el entorno añadiendo mandatos propios. Aunque la escritura de un script requiere un poco de programación, es mucho más fácil que escribir un programa en C.

Cuando un shell detecta un mandato que no es intrínseco, es decir, uno que el shell no reconoce y ejecuta directamente, llama al kernel para que lo ejecute. El mandato puede ser un programa compilado o un shell script. En el segundo caso, el kernel tiene que seleccionar un shell para ejecutar el script. Este shell hijo se denomina subshell.

La manera como el kernel hace su elección depende del sistema. El caso por omisión es siempre una versión del shell Bourne o, en algunas ocasiones, el shell Korn (que es casi compatible con aquél). La compatibilidad con el shell Bourne es esencial porque, históricamente, éste era el único disponible en la séptima edición; varios procedimientos de shell de uso común suponen, sin indicación explícita, que son interpretados por el shell Bourne.

La mayoría de los sistemas se adhieren al convencionalismo de BSD UNIX que especifica que la primera línea de un procedimiento de shell tiene la forma

```
#! shell
```

donde *shell* es el nombre de la ruta completa del shell que se usa para interpretar el script. El espacio en blanco después de "#!" es opcional. Por ejemplo, si se escribe un guión usando el conjunto de mandatos del shell C, deberá comenzar con

```
#! /bin/csh
```

Algunos sistemas que no respetan el convencionalismo "#!" ofrecen otra manera de etiquetar los procedimientos de shell: si el primer carácter es ":", se considera como un script del shell Bourne; si el primer carácter es "#", se considera como un script del shell C.

TEMA 4: COMANDOS MÁS COMUNES.

SINTAXIS DE LAS ÓRDENES:

Una orden o comando es una secuencia de palabras por uno o más espacios en blanco. Formato:

NombreOrden Opción(es) Expresión NombreArchivo(s)

La primera palabra es la propia orden. El resto de las palabras son argumentos de la orden. Estos son los argumentos de las órdenes:

- Opción(es): Es un literal, normalmente precedido por el signo menos. Por ejemplo: -al. Una opción modifica la acción de la orden de alguna manera o da detalles de cómo tiene que funcionar exactamente.
- Expresión: Describe una cadena de caracteres que se va a utilizar como entrada para la orden.
- NombreArchivo(s): Nombre de uno o más archivos que la orden va a manipular de alguna manera.

Ejemplos:

rm -fi datos.txt fich.pas Borra dos archivos con las opciones "f" e "i".
grep "Hola" saludo Busca patrones en un archivo donde el primer argumento es una expresión y el segundo argumento es un nombre de archivo.

Si por alguna razón tiene que utilizar un argumento que contiene un espacio o bien tiene que limitarlo, el argumento entero debería colocarse entre comillas dobles (") o entre comillas simples o apóstrofos.

METACARACTERES:

El shell reconoce como especiales una serie de caracteres, a los que se les denomina como metacaracteres. Tienen varias aplicaciones, que a continuación se describen.

Caracteres comodines:

Estos caracteres se utilizan para sustituir una secuencia de caracteres. Entre los más importantes están:

? Identifica a un único carácter.
***** Identifica a una cadena de caracteres.
~ Abreviatura del home directory.
. Abreviatura del directorio de trabajo actual.

.. Abreviatura del directorio padre del actual.

[..] Proporciona un sólo carácter de los que aparecen entre los corchetes.

[x-y] Proporciona algún carácter dentro del rango x,y. Se pueden especificar varios rangos o combinar rangos con caracteres. No hace falta ningún carácter separador.

Redireccionamiento de la E/S:

Estos caracteres se usan para cambiar la entrada estándar (que es el teclado) o la salida estándar (que es el monitor). Son los siguientes:

> Redirecciona la salida hacia un archivo o dispositivo. Si el archivo existe, dará error.

>> Añade la salida a un archivo.

< Redirecciona la entrada de un archivo. Hay comandos que no aceptan el redireccionamiento de entrada.

cmd << word

Indica que un comando o un programa, cmd, normalmente interactivo, acepta sus órdenes desde el mismo archivo o dispositivo (usualmente un shell script). word se interpreta literalmente como la marca de fin de entrada para el comando.

>& Salida de errores: Cuando se produce un error, se usa un canal diferente (que el de salida) para informar del error. Se puede enviar los errores a otro archivo o dispositivo.

‘’ Comillas simples: Es otra forma de redireccionar la salida. Se ejecuta primero lo que se encuentra entre las comillas, y el resultado puede servir como argumento para otro comando. Por ejemplo:

```
rm *          No elimina los archivos ocultos.
rm ‘ls -a’    Elimina todos los archivos.
```

Pipe. Pipeline:

Un pipe consiste en usar la salida de una orden como entrada a otra orden. Se representa con el carácter “|”. Un pipeline o interconexión es un conjunto de órdenes unidas por pipes. Por ejemplo:

```
ls /etc | wc
```

Muestra el número de líneas, palabras y caracteres del listado del directorio /etc.

Líneas multicomando:

; Separa órdenes en una línea de entrada, ejecutando una detrás de otra.

**** Al final de una línea, permite continuar la orden en la siguiente línea de entrada. En otro caso, antecede a un carácter con significado especial.

( ) Se utilizan para agrupar comandos. Por ejemplo:

```
(date;who) | wc
```

ejecuta el comando *date* y después el *who*. La salida de ambos sirve como entrada al comando *wc*.

ORDENES DE MANIPULACIÓN DE DIRECTORIOS:

- Cambiar de directorio (cd - Change Directory)

Sintaxis:

```
cd [<dir>]
```

casos especiales:

```
cd ..
cd $HOME
cd $home
cd
```

El primer caso especial nos permite situamos en el directorio padre del directorio actual. Los otros tres casos nos permite situarnos automáticamente en nuestro directorio de trabajo (home directory).

- Mostrar el directorio actual (pwd - Path Work Directory)

Sintaxis:

```
pwd
```

- Crear un directorio (mkdir - MaKe DIRectory)

Sintaxis:

```
mkdir <dir>
```

- Borrar un directorio (rmdir - ReMove DIRectory)

Sintaxis:

```
rmdir <dir>
```

- Renombrar un directorio (mv - MoVe)

Sintaxis:

```
mv <dir-antiguo> <dir-nuevo>
```

Esta orden no cambia el contenido del directorio.

- Copiar directorios (cp - COPY)

Sintaxis:

```
cp -R <dir-origen> <dir-destino>
```

Copia recursivamente los archivos y directorios que contenga (subdirectorios).

- Mostrar el contenido de un directorio (ls - LiSt)

Sintaxis:

```
ls [alFRxt] [<dir>]
```

El comando "l" es equivalente a "ls -l". Significado de la descripción detallada de "ls -l":

```
…
drwxr-x---    2   julio   group     224   Jul 19 11:19   bin
drwxr-x---    2   julio   group     336   Jul 17 17:14   fuentes
- rw -r------ 1   julio   group      24   Jul 25 15:57   cx
- rwxr-x--x   1   julio   group      35   Jul 19 11:44   whiet
…
```

- El primer campo es el modo del archivo. La primera columna nos indica el tipo de archivo (- ordinario, d directorio, b archivo especial de bloque, c archivo especial de carácter, l archivo simbólico, p archivo especial FIFO). El resto del campo nos dice cuáles son los permisos del archivo.
- Número de enlaces. Para archivos es normalmente 1. Si es mayor de 1 indica que existe entradas en otros directorios apuntando a ese archivo. Para directorios, indica el número de entradas de subdirectorios que contiene.
- Nombre del propietario. Muestra el nombre del propietario del directorio o archivo.
- Nombre del grupo del propietario.
- Tamaño del archivo, en bytes. Para archivos de texto es el número de caracteres.
- Fecha y hora de la última modificación.
- Nombre del archivo.

Otras opciones de la orden *ls* son:

ls -x Lista los archivos por columnas.
ls -F Marca los directorios con '/', los archivos ejecutables con un '*' y los enlaces simbólicos con '@'.
ls -t Lista ordenada según el tiempo de la última modificación.
ls -R Visualiza los contenidos de cada subdirectorio a partir del indicado (opción recursiva).
ls -g Igual que la opción '-l', excepto que el propietario no es mostrado.
ls -a Muestra los archivos ocultos. Los archivos ocultos son los que empiezan con el carácter '.'. Por ejemplo: .profile, .cshrc

ORDENES DE MANIPULACIÓN DE ARCHIVOS:

- Mostrar el tipo de un archivo (file)

Sintaxis:
file <filename>

Determina el tipo de un archivo examinando su contenido. Algunas de los posibles resultados son:

<filename> : ascii text
<filename> : empty
<filename> : directory
<filename> : symbolic link to filename
<filename> : shell script text
<filename> : commands text

- Crear o inicializar un archivo (touch)

Sintaxis:
touch <filename>

Crea o inicializa un archivo. Su tamaño será de 0 bytes.

- Mostrar el contenido de un archivo (more, cat, tail, head)

more

Sintaxis:
more [<filename>]

Visualiza un archivo de texto pantalla a pantalla. Una vez que se esta ejecutando se puede hacer lo siguiente:

- pulsar el espaciador para visualizar otra pantalla.
- pulsar <Return> para avanzar línea a línea.
- escribir /patrón para avanzar hasta la 1ª línea que contenga dicho patrón.
- pulsar el carácter 'q', para salir.

Ejemplo:

```
% more /etc/passwd
```

cat

Sintaxis:
cat [<filename>…]

Permite visualizar el contenido de uno o más archivos sin paradas a través del dispositivo de salida estándar. Se puede detener la imagen pulsando <Ctrl>-S, y para continuar <Ctrl>-Q. Ejemplo:

```
% cat profile exrc
```

pg

Sintaxis:

pg [- | + <número>] [+/<patrón>] <filename> …

Permite paginar la salida. Si el fichero a ver tiene más de 24 líneas, 'pg' visualizará las 23 primeras y presentará en la línea 24 un carácter ':' de petición de orden:

- si pulsa <Return> verá la siguiente página.
- si desea abandonar deberá pulsar 'q'.

Otras opciones de *pg*:

% pg -20 archivo	Fija el tamaño de pantalla en 20 líneas en lugar de las 24 que hay por defecto.
% pg +13 archivo	Comienza a visualizar a partir de la línea 13.
% pg +/patrón/ archivo	Comienza a listar a partir de la 1ª línea que contenga 'patrón'.

tail

Sintaxis:

tail [+ l - <número>] <filename>

Permite examinar el final de un archivo. Por defecto visualizará las diez últimas líneas, pero es posible modificar dicho número. Ejemplos:

% tail -3 /etc/passwd	Visualiza las tres últimas líneas del archivo /etc/passwd.
% tail +10 /etc/group	Visualiza el archivo a partir de la línea 10. Si se utiliza la opción '+nº líneas', y el archivo no tiene tantas líneas, no se visualizará nada.

head

Sintaxis:

head [- <número>] <filename>

Nos permite visualizar las diez primeras líneas de un archivo, aunque como en el caso de *tail* podemos modificar su valor. Ejemplo:

% head -20 /etc/tempcap	Muestra las primeras 20 líneas.

- Mover o renombrar un archivo (mv - MoVe)

Sintaxis:

mv <filename1> <filename2>

ó

mv <filename>… <directory>

Ejemplos:

mv mensaje01 mesj.01 — Cambia el nombre del archivo mensaje01; y en el caso de que existiese mesj.01 éste cambiará su contenido por el de mensaje0l.

mv user10/mbox user11 — Movemos el archivo 'mbox' del directorio user10 al directorio userll.

- Copiar archivos (cp - COPY).

Sintaxis:

cp <filename1> <filename2>
ó
cp <filename>… <directory>

Ejemplo:

cp /etc/motd mensaje — Copia en nuestro directorio de trabajo el archivo 'motd' y renombra el archivo.

Después de haber realizado la copia de un archivo, el modo del archivo es el mismo que tenía el original.

- Borrar archivos (rm - ReMove).

Sintaxis:

rm <filename> …

Permite borrar o suprime uno o más archivos de un directorio (sólo puede borrarlos el propietario o el super usuario). En el caso de que uno de los archivos a suprimir estuviese protegido contra escritura, 'rm' le informaría del modo real del archivo, y esperaría una respuesta: si pulsa 'y' procede a borrarlo y si pulsa cualquier otra tecla no lo borrará. Opciones:

-i : (Interactiva). 'rm' pide confirmación de cada uno de los archivos que va a borrar, independientemente de que esté o no protegido.
-f : Si se quiere forzar la supresión de archivos estén o no protegidos.
-r : Si se quiere borrar el contenido del directorio actual, y de los posibles subdirectorios que existan a partir de él (recursivamente).

MODIFICACIÓN DE PERMISOS Y PROPIETARIOS:

- Definir los permisos de creación de un archivo o directorio (umask)

Sintaxis:

umask [<valor>]

donde <valor> es un valor en octal indicando los permisos del propietario, grupo y otros. que se van inhabilitar cuando se cree un archivo. Normalmente es 022. El primer dígito indica los permisos del propietario, el segundo los permisos del grupo y el tercero los permisos de otros. El permiso de ejecución se indica con 1, el de escritura con 2 y el de lectura con 4. Para una combinación de estos permisos, se suman esos valores. Ejemplo:

```
% umask 037
```

Los archivos que se creen tendrán inicialmente los permisos de rwx para el propietario, r-- para el grupo y --- para el resto.

- Cambio del propietario de un archivo (chown).

Sintaxis:

```
chown <username> <filename> …
```

Cambia el propietario de un archivo o de un directorio. <username> pueden ser también el UID. Sólo el propietario y el super usuario puede cambiar al propietario de un archivo.

- Cambio del grupo de un archivo (chgrp).

Sintaxis:

```
chgrp <groupname> <filename> …
```

Permite cambiar la propiedad de grupo de directorios y archivos a otro grupo del sistema.

- Cambio de los permisos de un archivo (chmod).

Sintaxis:

```
chmod modo archivo …
```

chmod cambia los permisos de un archivo o directorio donde el modo es [ugoa+-rwx]. Siendo:

u	usuario/propietario
g	grupo
o	otros usuarios
a	todos los usuarios (equivale a ugo)
+	da el permiso
-	quita el permiso
rwx	permisos de lectura, escritura y ejecución

Sólo puede ser cambiado por el propietario del archivo o por el superusuario. El modo se puede especificar también en octal. Ejemplos:

chmod ug+x programa	Da permiso de ejecución al propietario y grupo del archivo "programa".
chmod 777 publico	Da todos los permisos a todos los usuarios para el archivo "público"

IMPRESIÓN:

- Impresión de archivos (lp).

Sintaxis:

```
lp <filename>
```

El archivo se imprime en la impresora por defecto del sistema. Tras ejecutar la orden el sistema nos indicará el número de solicitud de impresión o 'id-request'.

- Cancelación de una petición de impresión (cancel).

Sintaxis:

```
cancel [<id-request>]
```

Tan sólo se puede cancelar las peticiones de impresión que haya efectuado un determinado usuario. Ejemplo:

```
% cancel 65
```

- Estado del servicio de impresión (lpstat).

Visualiza el estado de las solicitudes de impresión efectuados. Opciones:

-o Permite ver el estado de las peticiones de todos los usuarios.
-p Permite ver el estado de todas las impresoras.

ORDENES DIVERSAS:

- Cambio de password (passwd)

Sintaxis:

```
passwd
```

Cambia la clave de acceso (palabra de paso) a una cuenta de usuario. Sólo el propietario de la cuenta o el superusuario pueden cambiar un password.

- ¿Quién está en el sistema? (who / who am i)

Sintaxis:

```
who
```

ó

```
who am i
```

informa de quién esta conectado al sistema o de quién soy yo. Aparece la siguiente información:

UserName Terminal FechaConexión (DireccInternet)

- Limpiar la pantalla del terminal (clear)

Sintaxis:

```
clear
```

- Enlaces a archivos (ln)

Sintaxis:

```
ln [-s] <filename>
```

Permite enlaces fijos (hard) o simbólicos (symbolic) a archivos o directorios.

- Enlace fijo: Es una entrada en un directorio estándar. Sólo se hacen a archivos existentes y dentro del mismo file system al que pertenezca el archivo. Para eliminar un archivo, se tiene que eliminar todos los enlaces fijos (incluyendo el primer nombre que se le dio al archivo).
- Enlace simbólico: Se realiza con la opción -s. Es una entrada de directorio especial que apunta a otro archivo existente. El archivo puede estar en otro file system. Si se elimina el archivo al que apunta, no eliminan todos sus enlaces simbólicos.

PROCESOS:

Como hemos visto anteriormente, cualquier usuario puede ejecutar varios procesos al mismo tiempo, uno en modo atendido o primer plano (foreground) y los demás en modo desatendido o segundo plano (background). Cuando un proceso se ejecuta en background el resultado del proceso no se visualiza en pantalla, siempre y cuando no utilice la E/S estándar. A partir de ese instante se puede introducir nuevas órdenes mientras se ejecuta ese proceso en background.

Colocación de un proceso en background

Para colocar un proceso en background hay que terminar la línea de órdenes con el carácter '&'. El sistema nos devolverá el PID (identificador de proceso) asociado al proceso en background.

Listado de procesos en ejecución (ps - Processes Status).

Sintaxis:

```
ps
```

La orden *ps* (Processes Status) muestra los procesos que están en ejecución. Por ejemplo:

```
PID     TTY     TIME    COMMAND
49      01      0:00    -csh
123     01      0:00    ps
```

donde : PID es el identificador de proceso, TTY es el terminal de control donde se creó el proceso, TIME es el tiempo de CPU usado por el proceso y COMMAND indica los procesos en ejecución.

- Detención de procesos (kill).

Su sintaxis es:

```
kill [ - <código>] <PID>
```

Para detener procesos ejecutándose en primer plano pulse la combinación de teclas <Ctrl>-Z (esto no elimina el proceso, sólo lo detiene). Para eliminar la ejecución de un proceso en primer plano (foreground), pulse <Ctrl>-C. Si desea eliminar (terminar su ejecución) un proceso background concreto se necesita conocer su PID y a continuación ejecutar la orden kill. Si no es posible detener un proceso con la orden *kill <PID>* , se tiene que utilizar el <código> 9, que corresponde a parada segura. Los usuarios tan sólo podrán detener los procesos que les pertenezcan.

- Ejecución en el tiempo (at)

Sintaxis:

```
at [<opciones>] <time> [<date>]
```

donde <time> estará formado por 1, 2 ó 4 dígitos. Si se usa 1 ó 2 dígitos, se considerará como la hora. Con 4 dígitos, se considerará hora y minutos. Permite ejecutar una orden en un determinado momento (en el tiempo). Los comandos los toma de la entrada estándar y los ejecuta en modo background. La salida de los comandos, si no se redirecciona, se almacenará como un mensaje en la utilidad *mail*. Ejemplo:

```
% at 1040 < cshrc
```

Estas son dos de las opciones del comando *at*:

at -l	Informa de que trabajos están esperando a que le llegue su momento de ejecución.
at -r <tarea>	Elimina el trabajo especificado de la cola de *at*.

TEMA 5: PROGRAMACIÓN CON EL SHELL.

¿POR QUÉ UN SHELL PROGRAMABLE?:

El shell de UNIX no es un caso típico dentro de los intérpretes de comandos: aunque permite al usuario ejecutar comandos en la forma usual, por ser un lenguaje de programación, puede hacer mucho más. Al usar el shell el usuario escribe siempre pequeños programas de una línea: una interconexión es un programa. El shell trabaja así: se le programa constantemente, pero eso es tan fácil y natural (una vez que uno está familiarizado con él) que uno no lo considera como programación.

El shell hace algunas cosas como iterar, redireccionar E/S y expandir nombres de archivos, por lo que ningún programa necesita hacer eso; y algo más importante: la aplicación de estos recursos es uniforme en todos los programas. Otras características como los procedimientos de shell y las interconexiones, son en realidad proporcionados por el kernel, pero el shell proporciona una sintaxis natural para crearlos. Estos van más allá de la comodidad; sirven en realidad para incrementar las capacidades del sistema.

Gran parte de la potencia y la comodidad del shell derivan del núcleo de UNIX que está atrás; por ejemplo, aunque el shell arma las interconexiones, en realidad es el núcleo el que mueve los datos a través de ellas. La forma en que el sistema maneja los archivos ejecutables hace posible escribir procedimientos de shell de manera que se ejecuten exactamente como programas compilados. El usuario no necesita estar enterado de que son archivos de comandos (no se invocan con un comando especial). Asimismo, el shell mismo es un programa, como hemos visto, y no forma parte del núcleo, por lo que puede adaptarse, extenderse y emplearse como cualquier otro programa. Esta idea no es exclusiva de UNIX, pero es donde mejor a sido explotada.

Veremos ahora algunas de las posibilidades que nos ofrece el shell. El shell con el que vamos a trabajar de aquí en adelante va ser con el C shell, *csh*.

CREACIÓN DE NUEVOS COMANDOS:

Ahora veremos cómo crear nuevos comandos a partir de los viejos por medio de los procedimientos de shell o shell scripts. Dada una serie de comandos que van a ser repetidos varias veces, sería conveniente convertirla en un "nuevo" comando con un nombre propio, de manera que pueda usarse como un comando normal. En concreto, supongamos que uno intenta contar usuarios frecuentemente mediante la interconexión

```
% who | wc -1
```

y se desea escribir un nuevo programa *nu* que lo haga.

El primer paso es crear un archivo ordinario que contenga *who | wc -l*. El lector puede emplear su editor favorito o bien, si es creativo, teclear:

```
% echo 'who | wc -l' > nu
```

Como se ha mencionado anteriormente, el shell es un programa como un editor o *who* o *wc*. Y puesto que es un programa, se puede ejecutar y redireccionar su entrada. Así, ejecútese el shell, con la entrada que provenga del archivo *nu* y no de la terminal:

```
% who
you     tty2    Sep 28 07:51
rhh     tty4    Sep 28 10:02
moh     ttys    Sep 28 09:38
ava     tty6    Sep 28 10:17
% cat nu
who | wc -l
% csh <nu
        4
%
```

La salida es la misma que la que seria si se hubiera tecleado *who | wc -1* en la terminal. Una vez más, al igual que muchos otros programas, shell toma su entrada de un archivo si se le da alguno como argumento; se podría haber escrito

```
% csh nu
```

para obtener el mismo resultado. Pero es molesto tener que teclear "csh" en uno u otro caso: Es más tardado y crea una distinción entre programas escritos en, digamos, C y otros que se escribieron conectando programas con el shell. Por lo tanto, si un archivo es ejecutable y si contiene texto, entonces shell supone que es un archivo de comandos de shell. Tal archivo se llama procedimiento de shell o shell script (como hemos visto anteriormente). Lo único que debe hacerse es incluir una línea, la primera del archivo, que contenga el carácter "#" para forzar que el subshell que se va a ejecutar sea el *csh* (en caso contrario sería el *sh*, que es el que se ejecuta por defecto)[7]. Después se hace *nu* ejecutable haciendo:

```
% chmod +x nu
```

y después se le puede invocar con

```
% nu
```

En lo sucesivo los usuarios de *nu* no podrán saber, con sólo ejecutarlo, que el lector lo ha obtenido de esta sencilla manera. La manera en que el shell realmente ejecuta *nu* es crear un nuevo proceso de shell exactamente igual que si se hubiera tecleado

```
% csh nu
```

csh nu no es lo mismo que *csh<nu* ya que su entrada, estándar está conectada aún a la terminal.

Existen otros comandos simples que el lector podría crear de esta manera para adaptar el ambiente a su propio gusto.

ARGUMENTOS Y PARÁMETROS EN LOS COMANDOS:

[7] También es válido el convencionalismo de BSD UNIX, comentado en el tema 3 de este mismo bloque.

Aunque *nu* es adecuado tal como se mostró, la mayoría de los programas en shell interpretan argumentos, de manera que por ejemplo las opciones y nombres de archivos puedan especificarse cuando el programa se está ejecutando.

Supongamos que queremos hacer un programa llamado *cx* para cambiar el modo de un archivo a ejecutable de manera que

```
% cx nu
```

sea una abreviatura de

```
% chmod +x nu
```

Ya conocemos casi todo lo que se necesita para hacer esto. Necesitamos un archivo llamado *cx* cuyo contenido sea

```
chmod +x archivo
```

Lo único nuevo que necesitamos saber es cómo indicar a *cx* cuál es el nombre del archivo, puesto que éste será diferente cada vez que se ejecute *cx*.

Cuando shell ejecuta un archivo de comandos, cada ocurrencia de $1 se reemplaza por el primer argumento, cada ocurrencia de $2 se reemplaza por el segundo argumento y así sucesivamente hasta $9. El argumento $0 es el nombre del programa en ejecución (en *cx*, $0 es “cx"). En consecuencia, si el archivo *cx* contiene

```
chmod +x  $1
```

cuando el comando

```
% cx nu
```

sea ejecutado, el subshell reemplaza "$1" por su primer argumento, "nu".

Veamos la secuencia complete de operaciones:

```
% echo '# \
chmod +x $1' >cx                Cree cx originalmente.
% csh cx cx                     Haga que cx sea ejecutable.
% echo echo Hi. there! >hola    Haga un programa de prueba.
% hola                          Pruébelo.
hola: Permission denied
% cx hola                       Hágalo ejecutable.
% hola                          Inténtelo otra vez.
Hi , there!                     ¡Funciona!
%
```

¿Y si se deseara manejar más de un argumento, por ejemplo si hubiera que hacer que un programa como *cx* manejara varios archivos al mismo tiempo? Un primer intento es poner nueve argumentos en el programa en shell, como en

```
chmod +x $1 $2 $3 $4 $5 $6 $7 $8 $9
```

(¡Sólo hasta $9 debido a que la cadena $10 se analiza como "primer argumento, $1, seguido de un 0"!) Si el usuario de este shell script proporciona menos de nueve argumentos, los restantes serán cadenas nulas; el efecto es que sólo los argumentos que fueron realmente tecleados son pasados a *chmod* por el subshell. Así, esto funciona, pero obviamente no es limpio y falla si se dan más de nueve argumentos.

Previendo este problema, shell proporciona una abreviatura $* que significa "todos los argumentos". La manera correcta de definir *cx* entonces es

```
chmod +x  $*
```

que funciona sin importar cuántos argumentos se den.

Los argumentos de un archivo de shell no necesitan ser nombres de archivos.

LA SALIDA DE PROGRAMAS COMO ARGUMENTOS:

Pasemos ahora de los argumentos en los comandos dentro de un archivo de shell a la generación de argumentos. Seguramente, la expansión de nombres de archivos a partir de metacaracteres como * es la forma más común de generar argumentos (aparte de darlos explícitamente), pero otra buena manera es hacerlo por medio de la ejecución de un programa. La salida de cualquier programa puede colocarse en una línea de comandos encerrando la invocación en comillas inversas `...`:

```
% echo At the tone the time will be `date`.
At the tone the time will be Thu Sep 29 00:02:15 colombia 1983.
%
```

Un pequeño cambio ilustra que `...` se interpreta dentro de las comillas "...":

```
% echo "At the tone \
the time will be `date`."
At the tone
the time will be Thu Sep 29 00:03:07 colombia 1983.
%
```

Como otro ejemplo, supóngase que se desea enviar correo a una lista de personas cuyos nombres de inicio de sesión están en el archivo *mailinglist*. Una manera difícil de tratar esto es editar *mai1inglist* para convertirlo en un comando de *mail* apropiado y presentarlo al shell, pero es mucho más fácil hacer:

```
% mail `cat mailinglist`<letter
```

Esto ejecuta *cat* para producir la lista de nombres de usuarios y éstos se convierten en argumentos para *mai1*. Cuando interpreta la salida en comillas inversas como argumentos, el shell trata a las nueva-líneas como separadores de palabras, no como terminadores de la línea de comandos.

VARIABLES DE SHELL O DE ENTORNO:

El shell tiene variables, como las que hay en la mayoría de los lenguajes de programación, las cuales en la jerga de shell se llaman a veces parámetros o variables de entorno. Las cadenas como $1 son parámetros posicionales (variables que contienen los argumentos para un procedimiento de shell). El dígito indica la posición en la línea de comandos. Existen otras variables de entorno: PATH es la lista de directorios en donde se buscan los comandos, HOME es el directorio de inicio de sesión del usuario, etcétera. A diferencia de las variables en un lenguaje normal, los parámetros posicionales no pueden alterarse; aunque PATH es una variable cuyo valor es $PATH, no existe una variable 1 cuyo valor sea $1. $1 no es más que una notación compacta del primer argumento.

Dejando a un lado los parámetros posicionales, las variables de entorno pueden crearse, accederse y modificarse. Para asignar variables se usa el comando *set*, por ejemplo:

```
% set PATH=:/bin:/usr/bin
```

es una asignación que cambia la trayectoria de búsqueda. No debe haber espacios alrededor del signo de igual, y el valor asignado debe ser una sola palabra, lo cual significa que debe delimitarse si contiene metacaracteres de shell que no deben ser interpretados. El valor de una variable se extrae anteponiendo a su nombre el carácter "$":

```
% set PATH=$PATH':/usr/grames'          Añade /usr/games a la variable PATH.
% echo $PATH
:/usr/you/bin:/bin:/usr/bin:/usr/games
% PATH=:/usr/you/bin:/bin:/usr/bin      Restauración.
%
```

No todas las variables son especiales para el shell. Se pueden crear nuevas asignándoles valores. Tradicionalmente, las variables con significado especial se escriben con letras mayúsculas, y los nombres ordinarios con letras minúsculas.

Uno de los usos más comunes de las variables es recordar cadenas largas, como trayectorias:

```
% pwd
/usr/you/bin
% set dir=`pwd`          Recuerde donde estamos
% cd /usr/mary/bin               Vaya a otra parte
% ...                            Trabaje un rato
% cd $dir                        Retorne
% pwd
/usr/you/bin
%
```

Cada shell tiene sus propias variables de entorno intrínsecas. Estas variables contienen normalmente información acerca de cual es el home directory del usuario, el directorio donde colocar su correo, la forma del prompt, parámetros propios del shell, etc... El comando de shell *set* despliega los valores de todas las variables definidas por el usuario, además de que se utiliza para asignar valores a las variables. Para ver sólo una o dos variables, *echo* es más apropiado. A continuación se muestra un ejemplo de variables de entorno intrínsecas con el *csh*:

```
% set
LOGTTY          /dev/tty02
_d      ()
```

```
argv     ()
cdspell
history  20
home     /home/julio
ignoreeof
noclobber
path     (/bin /usr/bin /home/julio/bin .)
prompt   %
shell    /bin/csh
status   0
%
```

Para borrar una variable se usa el comando unset de la forma:

% unset *variable*

El valor de una variable se asocia con el shell que la crea, y no se pasa automáticamente a los hijos del shell.

```
% set x=Hola                  Cree x
% csh                         Nuevo shell
% echo $x
x : Undefined variable        Nueva línea solamente; x indefinida en el subshell
% exit                        Abandone este shell
%                             Retorne al shell original
% echo $x
Hola                          x definida todavía
```

Esto significa que un shell script no puede cambiar el valor de una variable, ya que el script es ejecutado por un subshell:

```
% echo 'set x="Adios"  \      Haga un archivo de shell con dos renglones...
echo $x' >setx                ... para definir e imprimir x
% cat setx
set x="Adios"
echo $x
% echo $x
Hola                          x es Hola en el shell original
% csh setx
Adios                         x es Adios en el subshell...
% echo $x
Hello                         ... pero todavía Hola en este shell
```

Cuando se desea hacer el valor de una variable accesible en subshells, se debe definir la variable con el comando *setenv* de shell. Pero no hay forma de exportar el valor de una variable de un subshell a su padre. He aquí un ejemplo:

```
% setenv x Hola
% csh                         Nuevo shell
% echo $x
Hola                          x conocido en subshell
% setenv x Adios              Cambie su valor
% echo $x
```

```
Adios
% exit                          Deje este shell
%                               Retorne al shell original
% echo $x
Hola                            x todavía es Hola
```

Las variables asignadas mediante *setenv* no se visualizan con *set*. Para ello se utiliza el comando *env*. Existen algunas variables de este tipo que son creadas automáticamente al iniciarse la sesión (puede ser modificado dentro del archivo .profile). A continuación se ve un conjunto de ellas:

```
HOME=/home/julio
PATH=/bin:/usr/bin:/home/julio/bin:
LOGNAME=julio
TERM=ansi
HZ=100
TZ=colombia06
SHELL=/bin/csh
MAIL=/usr/spool/mail/julio
HUSHLOGIN=FALSE
```

ITERACIONES EN LOS PROGRAMAS DE SHELL:

Iterar con un conjunto de nombres de archivos es muy común, y para ello existe la proposición *foreach*. La sintaxis es:

```
foreach name  (lista de palabras )
        comandos
end
```

Por ejemplo, una proposición *foreach* para obtener nombres de archivos, uno por línea, es simplemente

```
% cat > listar
#
foreach i (*) \
     echo $i \
end
^d                                  <CTRL>-D
#
```

La "i" puede ser cualquier variable de shell, aunque el uso de i es tradicional. Obsérvese que el valor de la variable se accede mediante $i, pero que la iteración *foreach* se refiere a la variable i. Hemos usado * para obtener todos los archivos en el directorio de trabajo, pero puede usarse cualquier otra lista de argumentos. Normalmente se desea hacer algo más interesante que simplemente imprimir nombres de archivos. Una cosa que hacemos a menudo es comparar un conjunto de archivos con versiones anteriores.

FILTROS:

Hay una gran familia de programas de UNIX que lee alguna entrada, realiza una transformación y escribe alguna salida. Entre los ejemplos figuran *grep* y *tail* para seleccionar una parte de la entrada, *sort* para ordenarla, *wc* para contarla y así sucesivamente. A esos programas se les da el nombre de filtros. A continuación describiremos algunos de ellos:

La familia *grep:*

Tiene el siguiente formato:

% grep *patrón archivo(s)*

Examina los archivos nombrados o la entrada estándar e imprime cada línea que contenga un caso del patrón. *grep* es de gran utilidad para encontrar ocurrencias de variables en programas o palabras en documentos; también sirve para seleccionar partes de la salida de un programa:

% grep -n variable *.[ch]	Localizar *variable* en fuentes en C.
% grep From $MAIL	Desplegar encabezados de mensaje en buzón.
% grep From $MAIL | grep -v mary	Encabezados que no provienen de *mary*.
% grep -y mary $HOME/lib/phone-book	Encontrar número telefónico de *mary*.
% who | grep mary	Comprobar si *mary* inició sesión.
% ls | grep -v temp	Archivos que no contienen la cadena *temp* en su nombre.

La opción -n imprime números de línea, -v invierte el sentido de la prueba y -y hace que las minúsculas en el patrón se acoplen con las letras mayúsculas o minúsculas en el archivo (las mayúsculas siguen correspondiendo exclusivamente con las mayúsculas).

En todos los ejemplos que hemos visto hasta ahora, *grep* ha buscado cadenas ordinarias de letras y números, pero en realidad maneja patrones mucho más complicados e interpreta expresiones en un lenguaje sencillo para describir cadenas. Desde el punto de vista técnico, los patrones son una forma ligeramente limitada de los especificadores de cadena llamados expresiones regulares. Las expresiones regulares se especifican dando un significado especial a ciertos caracteres. Los metacaracteres ^ y $ “anclan” el patrón al inicio (^) o al final ($) de la línea. Por ejemplo:

```
% grep '^From' $MAIL
```

imprime las líneas que comienzan con From. Conviene encerrar los patrones de *grep* entre apóstrofos.

grep maneja rangos de caracteres en forma muy parecida a los de shell, de forma que [a-z] concuerda con cualquier minúscula. Pero se dan algunas diferencias; si un rango de caracteres comienza con el símbolo ^, el patrón se acopla con cualquier carácter excepto los del rango. Por consiguiente, [^0-9] concuerda con cualquier carácter que no sea un dígito.

Un punto '.'se acopla con cualquier carácter. El operador de cerradura * se aplica al carácter o metacarácter anteriores (incluyendo el rango de caracteres) en la expresión, y en conjunto concuerdan con cualquier número de acoplamientos sucesivos del carácter o metacarácter. Por ejemplo x* concuerda con una secuencia de x lo más larga posible, [a-zA-Z]* se acopla con una cadena alfabética, .* reconoce cualquier cosa hasta una nueva-línea y .*x concuerda con cualquier

cosa hasta la última x en el renglón. La cerradura se aplica exclusivamente a un carácter, de modo que xy* reconoce una x seguidas por varias y. Además, "cualquier número" incluye cero.

Con expresiones regulares, *grep* es un lenguaje de programación simple. Por ejemplo, para seleccionar en un archivo con campos delimitados con dos puntos las líneas cuyo segundo campo esté vacío sería:

```
% grep '^[^:]*::' archivo
```

El patrón es: comienzo de línea, cualquier número de caracteres que no sean dos puntos, doble dos puntos.

grep es en realidad el más antiguo de una familia de programas, otros miembros de la cual se llaman *fgrep* y *egrep*. Su comportamiento básico es igual, pero *fgrep* busca muchas cadenas literales simultáneamente, mientras que *egrep* interpreta expresiones regulares verdaderas: lo mismo que *grep* pero con un operador "o" y paréntesis para agrupar expresiones que se explican más adelante.

Tanto *fgrep* como *egrep* aceptan el parámetro -f para especificar un archivo del cual leer el patrón. En este archivo, las nueva-líneas separan los patrones que deben buscarse en paralelo.

Las expresiones regulares interpretadas por *egrep* son las mismas que en *grep*, con un par de adiciones. Los paréntesis pueden utilizarse para agrupar, de manera que (xy)* concuerda con la cadena vacía, xy, xyxy, xyxyxy y así sucesivamente. La barra vertical | es un operador "o"; today | tomorrow se acopla con today o tomorrow, como lo hace to(day|morrow). Por último, hay otros dos operadores de cerradura en *egrep*, + y ?. El patrón x+ reconoce una o más x, y x? reconoce cero o una x pero no varias.

Otros filtros:

Esta sección tiene por objeto indicar al lector la existencia y posibilidad de un rico conjunto de pequeños filtros que ofrece el sistema. Esta lista no pretende ser exhaustiva.

Empezaremos con *sort*, que es tal vez el más útil de todos estos filtros. Clasifica (ordena) su entrada por renglones en orden ASCII. Aunque esto es lo que obviamente se hace por defecto, hay otras muchas maneras en que uno podría querer clasificar los datos; *sort* trata de tenerlas en cuenta proporcionando multitud de opciones. Por ejemplo, la opción -f vuelve equivalentes las mayúsculas y minúsculas, eliminando con ello la distinción entre ellas. La opción -d (orden de diccionario) ignora todos los caracteres menos la letras, dígitos y blancos en las comparaciones. Aunque las ordenaciones alfabéticas son las más comunes, algunas veces se necesita hacer una comparación numérica. La opción -n clasifica atendiendo al valor numérico y la opción -r invierte el sentido de cualquier comparación. Por tanto:

```
% ls | sort -f          Clasificar nombres de archivo por orden alfabético.
% ls -s | sort -n       Clasificar con los archivos más pequeños primero.
% ls -s | sort -nr      Clasificar con los archivos más grandes primero.
```

sort normalmente clasifica un renglón entero, aunque se le puede indicar que dirija su atención sólo a campos específicos. La notación +m significa que la comparación omite los primeros m campos; +0 es el inicio de la línea. Así por ejemplo:

```
% ls -l | sort +3nr      Clasificar por conteo de bytes, primero el más grande.
% who | sort +4n         Clasificar por tiempo de inicio de sesión, primero el más antiguo.
```

Otras opciones útiles de *sort* son: -o, que especifica un nombre de archivo para la salida (puede ser uno de los archivos de entrada) y -u, que suprime todos los grupos de renglones, menos uno, que sean idénticos en los campos de clasificación.

El comando *uniq* es lo que inspiró la opción -u de *sort*: excluye todos los grupos de renglones adyacentes duplicados menos uno. El hecho de tener un programa especial para esta función le permite hacer tareas no relacionadas con la ordenación. Por ejemplo *uniq* eliminará los renglones e blanco múltiples, esté clasificada o no su entrada. Las opciones invocan maneras especiales de procesar las duplicaciones: -d imprime sólo los renglones duplicados; -u imprime únicamente los que son únicos (o sea, no duplicados); y -c cuenta el número de ocurrencia de cada renglón. Tiene la siguiente estructura:

% uniq *opciones* [*input* [*output*]]

El comando *comm* para comparar archivos. Si tenemos dos archivos de entrada ordenadores, f1 y f2, *comm* imprime tres columnas de salida: los renglones que ocurren sólo en f1, los que ocurren tan sólo en f2 y los que ocurren en ambos archivos. Cualquiera de esas columnas puede suprimirse por medio de una opción:

```
% comm -12 f1 f2
```

imprime únicamente los renglones que se encuentran en ambos archivos (no imprime las columnas 1 y 2), y

```
% comm -23 f1 f2
```

imprime los que se hallan en el primer archivo pero no en el segundo (no imprime las columnas 2 y 3). Esto es útil al comparar directorios, y al comparar palabras con un diccionario.

El comando *tr* transforma los caracteres de la entrada estándar y los envía a la salida estándar. Sin duda su uso más común es la conversión de mayúsculas y minúsculas:

```
% tr "[a-z]" "[A-Z]"        Convertir minúsculas a mayúsculas.
% tr "[A-Z]" "[a-z]"        Convertir mayúsculas a minúsculas.
```

La relación en la transformación se realiza de forma que el primer carácter de la primera cadena se transforma en el primer carácter de la segunda, el segundo carácter con el segundo y así sucesivamente. Estas son la opciones de *tr*: -d borra todos los caracteres de la entrada que se correspondan con la primera cadena; -c complementa el conjunto de caracteres de la primera cadena respecto del código ASCII, es decir, transforma los caracteres que no aparecen en la primera cadena; -s fuerza a que no haya caracteres consecutivos iguales de la segunda cadena en el resultado. Las siguientes abreviaturas se usan para introducir rangos de caracteres o repeticiones en las cadenas: [a-z] introduce un rango en el que se incluyen todas las minúsculas; [a*n] introduce n repeticiones de a. Si n es cero o no aparece, todos los caracteres de la primera cadena se transforman en a. Además, el carácter \ seguido de un número en octal se corresponde con el carácter cuyo código ASCII es el especificado. Este ejemplo crea una lista de palabras, una por línea:

```
% tr -sc "[A-Z][a-z]" "[\012*]" <file1 >file2
```

El mandato *cut* extrae porciones específicas de cada línea de entrada. Las porciones se pueden definir especificando las posiciones que ocupan los caracteres, o los campos que ocupan y el carácter que delimita los campos. La forma del comando es:

% cut *opciones* [*archivos*]

donde la entrada consiste en la concatenación de los archivos especificados en *archivos*. Si no se especifican archivos, *cut* leerá la entrada estándar. La opción -c especifica extracción por caracteres; la opción -f especifica la extracción por campos. Sólo puede estar presente una de estas opciones.

En ambos casos, el argumento de la opción especifica las porciones que se extraerán mediante una lista que consiste en una secuencia de números sencillos e intervalos. El primer carácter o campo tiene el número 1 y los números deben estar en orden ascendente. Se puede omitir el primer número de un intervalo (para indicar el primer elemento) o el último (para representar el último elemento). Por ejemplo:

% cut -c 2,5-8,14-

extrae los caracteres 2, 5 a 8 y 14 a l de cada línea, donde l es la longitud de la línea.

En caso de extracción de campos existen otras dos opciones: -dc especifica el delimitador de los campos donde c es el carácter delimitador (por defecto es el tabulador); -s suprime las líneas que no contienen caracteres delimitadores. Los campos vacíos están permitidos y se reconocen. Por ejemplo:

% cut -f 1,3-4 -d :

especifica que deben extraerse los campos 1, 3 y 4 de cada línea, usando ":" como carácter delimitador.

El mandato *paste* une líneas correspondientes de varios archivos y envía el resultado a la salida estándar. Si sólo hay dos archivos de entrada, cada línea de la salida contendrá una línea del primer archivo seguida por un carácter separador y una línea del segundo archivo. La salida es similar si hay más de dos archivos de entrada. El carácter separador por omisión es el tabulador. También se puede usar *paste* con la opción -s para pegar líneas de un solo archivo y formar una sola línea.

La línea de mandatos de *paste* tiene la forma:

% paste [*opciones*] [*archivos*]

Los archivos de *archivo* contienen las líneas que se unirán. Si no se especifican archivos, se pegan las líneas de la entrada estándar. Otra opción de *paste* es -dlista que indica que se va a usar circularmente los caracteres de lista como separadores. Por ejemplo, si se especifica *-d,;* y hay cuatro archivos de entrada, una línea podría tener este aspecto: tweedledum,tweedledee;huevo,caer.

SELECCIÓN EN LOS PROGRAMAS DE SHELL:

La proposición *if* del shell ejecuta comandos basados en la condición de terminación de un comando. Su formato es el siguiente:

```
if (expresión )
then
        comandos si la expresión es verdadera
else
        comandos si la expresión es falsa
endif
```

o también:

```
if (expresión ) comandos si la expresión es verdadera
```

La localización de las nueva-líneas es importante: *if* y *else* se reconocen sólo después de una nueva-línea o un punto y coma. La parte *else* es opcional. La proposición *if* siempre ejecuta un comando (la condición). La condición será verdadera si el código de error que devuelve el comando es igual a cero, y será falsa si el código de error es distinto de cero. Por ejemplo, la siguiente sentencia *if* comprueba si existe el archivo prueba, en caso negativo emite un mensaje de aviso:

```
% if (! -f prueba) \
echo El archivo prueba no existe.
endif
```

El shell proporciona otros dos operadores para combinar comandos, || y &&, que a menudo resultan más compactos y adecuados que la proposición *if*. El operador || no tiene relación alguna con las interconexiones; es un operador condicional que significa OR (disyunción). El comando a su izquierda se ejecuta; si la condición de terminación es cero (éxito), el comando a la derecha de || se ignora. Si el lado izquierdo devuelve no-cero (fracaso) el lado derecho se ejecuta y el valor de la expresión completa es la condición de terminación del lado derecho. En otras palabras, || es un operador condicional OR que no ejecuta el comando a su derecha si el de la izquierda tiene éxito. El condicional && correspondiente es AND (conjunción); ejecuta el comando a su derecha sólo si el de la izquierda tiene éxito.

SELECCIÓN MÚLTIPLE EN PROGRAMAS DE SHELL:

El shell ofrece una proposición *case*, idónea para realizar una selección múltiple. Su formato es el siguiente:

```
switch (palabra )
case patrón :  comandos
                    breaksw
default: comandos
          breaksw
endsw
```

La proposición *switch* compara la *palabra* con los *patrones* de arriba hacia abajo y ejecuta los *comandos* asociados con el primer *patrón* que se reconozca, y solamente ése. Los patrones se escriben utilizando las reglas para reconocimiento de patrones, un tanto generalizadas a partir de las disponibles para identificación de nombres de archivo. Cada acción termina con *breaksw*. *default* recoge las acciones a realizar si *palabra* no coincide con ningún *patrón*.

EL CICLO *WHILE*: A LA ESPERA DE SUCESOS:

El ciclo *while* usa la condición de terminación de un comando para controlar la ejecución de los comandos en el cuerpo del ciclo. Este se ejecuta hasta que el comando de la condición devuelva una condición de no-cero. He aquí la forma básicas del ciclo:

```
while (comando )
        cuerpo del ciclo ejecutado a condición de que el comando devuelva verdadero
end
```

El comando condicional que controla un *while* puede ser cualquier comando. Para dar un ejemplo trivial mostramos un ciclo *while* para esperar a que alguien (mary, por ejemplo) inicie sesión:

```
while (sleep 60)
        who | grep mary
end
```

El *sleep*, que hace una pausa de 60 segundos, siempre se ejecutará normalmente (a menos que se le interrumpa) y, en consecuencia, devuelve "éxito", de modo que el ciclo se verificará una vez por minuto si mary ha iniciado sesión.

EL CICLO *REPEAT*:

Con el ciclo *repeat* se puede especificar que un comando o grupo de comandos se ejecute un número determinado de veces (similar al *for* del C). Tiene la siguiente forma:

```
repeat número  comandos
```

Como ejemplo tenemos la siguiente estructura que sirve para escribir en pantalla varias veces Hola:

```
% repeat 5 echo Hola
```

PRÁCTICAS:

1) Escribir un shell que muestre un menú con tres opciones:

1. Calendario
2. Correo electrónico.
3. Fin de sesión.

y que ejecute la opción escogida.

2. Escribir un shell que despliegue por pantalla las tablas de multiplicar 1-9 con *while*.

3. Realizar el ejercicio anterior, pero con estructura *for*.

4. Shell que despliegue por pantalla los subdirectorios del directorio de trabajo.

TEMA 6. EL EDITOR DE PANTALLA vi.

INTRODUCCIÓN:

El editor visual *vi* fue desarrollado por Bill Joy como parte del proyecto BSD UNIX y se ha adoptado como característica estándar de System V. *vi* es un editor de pantalla; siempre mantiene en la pantalla la imagen de una porción del archivo que se edita. Para ello se requiere una terminal que permita controlar la posición del cursor y colocar texto arbitrariamente en la pantalla. *vi* es en realidad una extensión del editor de línea *ex* y por eso desde *vi* se puede acceder a comandos de *ex*.

Cuando se está modificando un archivo en realidad no se modifica directamente el archivo original que se tiene en memoria secundaria. Esto se debe a que el archivo se copia primero en un área de anotación temporal ("buffer") y los cambios que efectuamos se realizan en dicha memoria temporal.

En un determinado momento sólo se puede visualizar en pantalla una porción del archivo que se desea modificar. A esta porción se le llama ventana en el archivo. Dentro de la ventana se puede mover el cursor para controlar dónde se van a hacer los cambios en el archivo.

La línea inferior de la pantalla es la línea de estado, que *vi* usa para los siguientes fines:

- Los mensajes de error aparecen en la línea de estado.
- Cuando *vi* genera información de estado (como por ejemplo el nombre del archivo actual) en respuesta a un mandato, aparece en la línea de estado.
- Cuando se teclea un patrón de búsqueda o un mandato de *ex*, *vi* lo presenta en la línea de estado.

MODOS DE FUNCIONAMIENTO:

vi siempre estará en uno de tres modos: el modo comando, el modo de entrada o el modo de línea de estado. El modo fundamental es el de mandatos, ya que los otros dos son en realidad variaciones del modo comando..

- El modo comando es el que se emplea para emitir comandos cortos. En este modo, *vi* trata cada carácter que se teclea en la terminal como mandato para hacer algo o como parte de un mandato; el carácter no se presenta en la terminal.
- El modo de entrada o modo texto es el que se emplea para insertar texto en el buffer. En este modo, *vi* trata como datos los caracteres que se teclean y los presenta en la terminal, en donde se encuentra el cursor; además, avanza el cursor una posición a la derecha.
- El modo de línea de estado o modo de última línea es el que se emplea para emitir mandatos largos. *vi* presenta en la línea de estado los caracteres que se teclean en este modo. El editor ejecuta el mandato cuando se pulsa <Return>.

Para pasar del modo comando al modo texto existe una serie de comandos (tales como a, i, o, O, etc…, cada uno de los cuales hará una acción determinada). Para pasar del modo texto al modo comando se pulsa la tecla <Esc>. Para introducir un comando en modo de línea de estado desde el modo comando se empieza con el carácter ':'.

ÓRDENES DE ENTRADA DEL EDITOR vi:

Estas son algunas de las formas de cómo se puede entrar en el editor *vi*.:

vi <archivo>	Entra en edición. Si el archivo no existe, lo crea
vi + <n> <archivo>	Comienza a editar el archivo a partir de la línea <n>.
vi + / <patrón> <archivo>	Entra en edición a partir de la primera ocurrencia de <patrón>
vi -r <archivo>	Recupera un archivo después de una salida anormal.
view <archivo>	Entra en edición en modo-lectura.

Cuando se entra en *vi*, se entra en modo comando.

COMANDOS BÁSICOS DEL vi:

	Comandos del cursor
h	Mover izquierda
j	Mover abajo
k	Mover arriba
l	Mover derecha
w	Mover izquierda una palabra
W	Mover izquierda una palabra (pasando puntuación)
b	Mover derecha una palabra
B	Mover derecha una palabra (pasando puntuación)
<Return>	Mover abajo una línea
<Back Space>	Mover izquierda un carácter
<Space Bar>	Mover derecha un carácter
H	Mover a lo alto de la pantalla
M	Mover a la mitad de la pantalla
L	Mover a la parte baja de la pantalla
<Ctrl>-F	Adelantar una pantalla
<Ctrl>-D	Adelantar media pantalla
<Ctrl>-B	Atrasar una pantalla
<Ctrl>-U	Atrasar media pantalla
	Inserción de caracteres y de líneas
a	Insertar caracteres a la izquierda del cursor
A	Insertar caracteres a la izquierda del cursor, al final de la línea
i	Insertar caracteres a la derecha del cursor
I	Insertar caracteres a la derecha del cursor, al principio de la línea
o	Insertar línea debajo del cursor
O	Insertar línea encima del cursor
	Cambiando texto
cw	Cambiar palabra (o parte de la palabra a la derecha del cursor)
cc	Cambiar línea
C	Cambiar la parte de la línea a la derecha del cursor

s	Substituir cadena por carácter debajo del cursor
r	Reemplazar carácter bajo el cursor por otro carácter
r-<Return>	Romper línea
J	Juntar la línea actual con la línea de arriba
xp	Intercambiar el carácter en el cursor con el carácter a la derecha
u	Deshacer último comando
U	Deshacer todos los cambios sobre la línea
:u	Deshacer el último comando de línea de estado
	Borrando texto
x	Borra el carácter sobre el cursor
dw	Borra la palabra (o la parte de la palabra a la derecha del cursor)
dd	Borra la línea actual
D	Borra la parte de la línea a la derecha del cursor
:5,10 d	Borra las líneas desde la 5 a la 10
	Copiando y moviendo texto
YY	Selecciona o copia línea
Y	Selecciona o copia línea
p	Pone la línea seleccionada o borrada debajo de la línea actual
P	Pone la línea seleccionada o borrada encima de la línea actual
:1,2 co 3	Copia las líneas 1 hasta 2 y las pone después de la línea 3
:4,5 m 6	Mueve las líneas 4 hasta 5 y las pone después de la línea 6
	Viendo los números de línea
:set nu	Muestra los números de línea
:set nonu	Oculta los números de línea
	Encontrando una línea
G	Va a la última línea del archivo
21G	Va a la línea 21
	Buscando y reemplazando
/cadena/	Busca cadena
?cadena?	Busca hacia atrás cadena
n	Encuentra la siguiente (o anterior) ocurrencia de cadena
:g/cadena1/s//cadena2/gc	Busca y reemplaza cadena1 por cadena2
	Limpiando la pantalla
<Ctrl>-L	Refresca la pantalla
	Insertando un archivo dentro de otro archivo
:r archivo	Inserta (lee) un archivo detrás del cursor
:34 r archivo	Inserta un archivo detrás de la línea 34
	Guardando y saliendo
:w	Guarda los cambios (escribe el buffer en disco)
:w archivo	Guarda los cambios en un archivo
:wq	Guarda los cambios y sale de vi
ZZ	Guarda los cambios y sale de vi
:!q	Sale sin guardar cambios

TEMA 7. COMUNICACIÓN ENTRE USUARIOS.

CORREO ELECTRÓNICO CON *mail*:

Envío de mensajes:

mail es la utilidad de UNIX con la cual los usuarios pueden enviarse mensajes entre ellos. Cuando un usuario entra en su cuenta se le notifica de la llegada de correo mediante el siguiente mensaje:

```
you have mail
```

Si ya se encontraba dentro de su cuenta, no se le notifica nada. Para enviar un mensaje se procede de la siguiente forma:

```
% mail <username>
Subject: <Motivo de la comunicación>

<texto del mensaje>
…
<Ctrl>-d
EOT
%
```

donde:

- Subject informa del tema o motivo de la comunicación.
- El texto puede ocupar tantas líneas como se desee.
- <Ctrl>-d debe de realizarse al comienzo de una línea y no aparecerá en pantalla.
- EOT representa el fin de la transmisión.

Recepción de mensajes:

Para poder determinar los mensajes que otros usuarios nos han enviado, se escribe lo siguiente:

```
% mail
```

Se muestra en pantalla todos los mensajes recibidos ordenados en orden inverso al de llegada; es decir, el primer mensaje que se muestra es el último que se recibió. La información que aparece es:

- Un número indicando el orden de recepción del mensaje.
- El remitente del mensaje.
- La fecha de recepción del mensaje.

- Tema o motivo del envío (Subject).

La utilidad *mail* tiene su propio prompt (&) para distinguirlo del shell. Si no existen mensajes, *mail* dará un mensaje de aviso y permanecerá en el shell.

Opciones de mail:

Hay dos grupos de opciones que se pueden utilizar:

- Opciones internas de *mail* (dentro de la utilidad *mail*).
- Opciones externas de *mail* (cuando nos encontramos en la línea de órdenes del shell).

Veamos algunas de las opciones más importantes de cada grupo.

Opciones internas de *mail*:

<Return>	Muestra el primer mensaje recibido. Si se continúa pulsando <Return> se visualizará el siguiente mensaje y así sucesivamente. Cuando no haya mas mensajes aparecerá el mensaje "Can´t beyond last message."
-	El guión nos permite ver el mensaje anterior al actual.
p	Muestra el último mensaje visualizado.
s archivo	Guarda el contenido del mensaje en edición en un archivo especificado.
w archivo	Igual que el anterior pero elimina la primera línea de la cabecera.
m NombreUsuario	Permite enviar correo a otro usuario sin salir de la utilidad *mail*.
?	Nos muestra todas las opciones internas que se pueden usar con *mail*.

Dentro de *mail* se pueden ejecutar órdenes del shell. Para realizar esto, basta con preceder a la orden con el signo de admiración ('!'). Ejemplo: !ls

La salida de *mail* se puede realizar de tres formas distintas:

- Pulsando <Ctrl>-d.
- Pulsando q (quit).
- Pulsando x (xit).

En este último caso se abandona *mail* sin "perder" los mensajes. En los dos primeros, se nos muestra un mensaje indicando que los mensajes quedan almacenados en el directorio de trabajo del usuario en un archivo llamado mbox.

Opciones externas a *mail*:

Se utilizan cuando se llama a *mail*.

mail -H	Muestra sólo el resumen de la cabecera sin entrar en *mail*.
mail -N	Entra en *mail* pero no muestra el resumen de cabecera. Aparece directamente el prompt de *mail*.

Se puede enviar correo a varias personas a la vez indicando más de un nombre de usuario en la línea de órdenes. Por ejemplo:

```
% mail user2 user3 user10
```

Por otro lado, si tiene un mensaje largo que enviar, lo puede redactar en un editor de textos y posteriormente lo puede enviar redirigiendo la entrada estándar. Por ejemplo:

```
% mail pepito < mensaje
```

LA ORDEN *WRITE*:

La orden *write* proporciona comunicación directa entre dos usuarios, mediante envío de mensajes directamente a sus dispositivos terminales. Estos mensajes interrumpirán cualquier visualización sobre el terminal del receptor. El comando *write* acepta el nombre de usuario como parámetro, determina cual es el dispositivo terminal del usuario y enviará mensajes allí. Cuando el mensaje haya sido transmitido, *write* acabará regresando al shell. La orden *write* devolverá un mensaje de error si el usuario especificado no está conectado.

El comando *write* obtendrá el mensaje de su entrada estándar. Si el carácter '!' se encuentra al principio de una línea, *write* llama al shell para que ejecute el resto de la línea como un comando y la salida es enviada al terminal.

Cuando alguien utiliza *write*, en el terminal de destino aparece una cabecera:

```
Message from pat on mi_sis (console) [ Mon Jun 18 08:29:58 ] …
```

El destinatario debe a su vez escribe *write* para responder al mensaje y se entra en mode de comunicación interactiva entre los dos usuarios. La comunicación terminará al pilsar <Ctrl>-d (fin de archivo).

Los usuarios pueden desactivar esta clase de comunicación directa con sus terminales por medio del comando *mesg* de esta forma:

```
% mesg n
```

Los mensajes pueden ser permitidos de nuevo especificando 'y' en vez de 'n'. La orden *mesg* funciona modificando los permisos del archivo de dispositivo asociado con un terminal. Si un usuario ha declarado desactivados mensajes, *write* devolverá un mensaje de error diferente.

Con frecuencia, el acceso de *write* se descapacita cuando se dirige salida de alta calidad al terminal, como ocurre en algunas tareas de impresión.

TEMA 8. LA DOCUMENTACIÓN DEL SISTEMA UNIX.

EL MANUAL DE USUARIO DE UNIX:

El manual del usuario es la documentación oficial del sistema UNIX y ha sufrido tanto desarrollo y modificaciones durante años como el propio software del sistema. Todas las órdenes, argumentos de órdenes, bibliotecas de subrutinas, formatos de ficheros, utilidades y herramientas están totalmente documentadas en el Manual de usuario y es casi la última autoridad en todas las cuestiones referentes al sistema UNIX. La autoridad última, naturalmente, es la comprobación empírica directa de un atributo del sistema sobre la propia máquina.

Desgraciadamente, aunque el Manual de usuario es un documento de referencia completo, no es tan accesible como pudiera desearse. No es raro que varios usuarios o creadores de software se enzarcen en una discusión acerca de una sentencia del Manual de usuario referente al significado o implicación de una palabra o frase. Invariablemente el Manual de usuario está correcto, pero a veces no es fácil de entender.

El Manual de usuario es un documento de referencia. Está diseñado para ser tan conciso como es posible, con objeto de exponer el gran alcance del sistema UNIX en un formato que los expertos y casi expertos puedan utilizar para encontrar cualquiera de los muchos detalles específicos del sistema. Es muy difícil, si no imposible, aprender a utilizar el sistema a partir del Manual de usuario, pero es igualmente imposible convertirse en un usuario experto sin el manual.

ESTRUCTURA DEL MANUAL DE USUARIO:

Originalmente, el Manual del usuario fue publicado como un único volumen pequeño que incluía toda la información que había disponible para el sistema operativo en ese momento. El documento original contenía ocho secciones principales, que aún hoy se conservan:

1. Ordenes y programas de usuario.
2. Llamadas al sistema.
3. Subrutinas.
4. Formatos de ficheros.
5. Misceláneas.
6. Juegos.
7. Ficheros especiales.
8. Procedimientos de mantenimiento del sistema.

Estas ocho secciones son con frecuencia referenciadas por su número.

La mayoría de las órdenes disponibles en el sistema están documentadas en la sección 1. Aunque algunas cuestiones relacionadas con ellas se tratan en las secciones 4, 5, 7 y 8, las órdenes de usuario se encontrarán documentadas en la sección 1 del manual.

Las secciones 2 y 3 son de interés principalmente para los programadores, ya que describen las subrutinas que un programador utilizaría en un programa en lenguaje C o Fortran. Algunos de los nombres de las funciones en las secciones 2 y 3 son similares a los nombres de las órdenes de la

sección 1, por lo que hay que tener cuidado de consultar la sección del manual correcta cuando se utilice el Manual de usuario.

La sección 4 documenta la manera en que los datos están almacenados en los ficheros que el sistema utiliza.

La sección 5 contiene información útil que no sería adecuada en otras secciones. Por ejemplo, incluye una tabla de códigos de caracteres ASCII, comentarios sobre las variables de entorno, descriptores de los códigos de caracteres específicos de cada país, una discusión de las especificaciones de tabulador en terminales, y mucho más.

En la sección 6 se describen juegos. Algunos sistemas no incluyen esta sección si la implementación no incluye juegos, pero el software adicional para juegos incluirá generalmente documentación que debería ser archivada en la sección 6.

La sección 7 incluye los formatos de los archivo especiales que residen en el directorio /dev. Este material es del máximo interés para los programadores que deseen utilizar estos archivos de dispositivos especiales en sus programas. La sección 7 incluye discusiones de los formatos de disco y cinta, las interfaces de redes de área local.

La sección 8, sobre procedimientos de mantenimiento, incluye procedimientos para arrancar el sistema, diagnosticar fallo hardware y otros aspectos de bajo nivel de la administración del sistema.

LA ORDEN *man* EN-LÍNEA:

Muchas máquinas grandes incluyen el contenido entero del manual en-línea, incluyendo generalmente las páginas de manual para software de instalación local junto con el Manual de usuario estándar. El texto completo ocupa generalmente entre dos y tres megabytes, por lo que las máquinas más pequeñas no siempre incluyen el manual en-línea.

Estas páginas del manual en-línea pueden ser visualizadas con la orden *man*, que escribe la página de manual solicitada en la salida estándar. Esta orden se invoca con un argumento que especifica la página del manual que se desea visualizar. Por ejemplo:

```
% man diff
```

visualizará la página del manual correspondiente a *diff(1)*[8]. Si existe una entrada bajo el nombre especificado en más de una sección del manual, se visualizarán todas las páginas del manual, una tras otra. Es posible redirigir la salida a un fichero o a una impresora para un examen más detenido. Las páginas del manual están generalmente formateadas con *nroff* cuando se visualizan, por lo que la orden *man* puede tardar un tiempo relativamente largo en generar la salida.

Para restringir la salida a una página de manual de una única sección de manual, se puede dar un número de sección antes del nombre de la página. Así, la orden

```
% man 1 passwd
```

mostrará la página de manual correspondiente a la orden *passwd(1)* solamente.

[8] El número entre paréntesis indica la sección del manual al que pertenece la entrada especificada.

BLOQUE 3: ADMINISTRACIÓN DEL SISTEMA.

TEMA 1. EL SUPERUSUARIO Y SUS FUNCIONES.

EL SUPERUSUARIO:

Como se comentó al principio del bloque anterior, existe un usuario especial. Este usuario es el superusuario, cuyo nombre de usuario (login) es "root", el que tiene poderes olímpicos sobre la ejecución (y posible destrucción) del sistema. El superusuario no está restringido por ninguna de las protecciones de que dispone el sistema: puede obtener cualquier fichero, puede cancelar cualquier proceso. Hay algunas operaciones del sistema que sólo el superusuario puede realizar.

Existen varias formas para pasar a superusuario. Una forma de entrar es arrancar el sistema en modo monousuario. Otra forma es entrar en la cuenta de superusuario como si de un usuario normal se tratase, introduciendo el nombre de usuario root cuando aparece el mensaje login; después se introduce su clave. Una última forma de entrar es estando en una cuenta de usuario normal, introducir el comando *su* y después introducir la clave.

La principal tarea del superusuario es la administración del sistema UNIX.

TAREAS ADMINISTRATIVAS BÁSICAS:

La administración del sistema consiste básicamente en gestionar los recursos del sistema para que se puedan utilizar de la forma más eficiente posible así como llevar control de los usuario potenciales del sistema. Entrando más en detalle, las tareas de administración del sistema más comunes son las siguientes:

- Añadir usuarios.
- Instalar software (como aplicaciones y actualizaciones del sistema operativo).
- Instalar hardware (como tarjetas, impresoras, terminales y modems).
- Mantener la seguridad e integridad del sistema y de la red.
- Diagnosticar y arreglar problemas software y hardware cuando ocurran.

- Comprobar el uso de los sistemas de archivos para asegurarse de que no estén llenos y además controlar el uso indiscriminado de los mismos.
- Mantener impresoras, modems y terminales remotos.
- Realizar copias de seguridad de los sistemas de archivos.
- Mantener servicios de red, correo y otros servicios de comunicaciones.

TEMA 2. ARRANQUE Y PARADA DEL SISTEMA.

ARRANQUE DEL SISTEMA:

La carga inicial (booting) es el proceso que hay que realizar para echar a andar el sistema cuando se conecta la computadora inicialmente o cuando lo reinicializa después de una detención.

El programa de carga inicial, llamado bootstrap, que se encuentra almacenado en el bloque de arranque del sistema de archivos raíz. Este programa se encarga de traer a memoria un programa más complejo, el cual se encarga de cargar, configurar e iniciar el sistema UNIX.

El proceso de arranque consta de tres pasos fundamentales:

- Cargar el sistema operativo.
- Chequear los sistemas de archivos (si el sistema se paró anormalmente).
- Elegir el modo de funcionamiento.

A continuación se describen estos tres pasos:

Cargar el sistema operativo:

El primer paso en el arranque del sistema es cargar el sistema operativo desde el disco duro de la computadora.

Se enciende la computadora y ésta carga el bootstrap y muestra el siguiente mensaje:

```
SCO System V/386

Boot
:
```

Se pulsa <Return> y el bootstrap carga el sistema operativo[9]. Cuando el sistema está cargado, muestra información sobre sí mismo y verifica que el sistema de archivos raíz está correcto y no tiene errores. Si no hay errores se pasa al paso tercero donde se elige el modo de funcionamiento. Si hay errores en el sistema de archivos, se pasará al paso donde se reparará el sistema de archivos.

Reparando el sistema de archivos:

Es necesario reparar el sistema de archivos si aparece el siguiente mensaje:

```
fsstat: root filesystem needs checking
OK to check the root filesystem (/dev/root) (y/n)?
```

[9] Si existe una partición de DOS, se puede cargar desde aquí introduciendo el comando *dos* detrás del prompt ':'.

Cada sistema de archivos que se quiera cargar y tenga errores, mostrará un mensaje similar. Para trabajar sin problemas, el sistema operativo necesita reparar el sistema de archivos.

Para reparar el sistema de archivos, introduzca 'y'. La utilidad *fsck* (que será explicada más adelante) limpiará el sistema de archivos, reparando los archivos dañados o borrando los archivos que no se hayan reparado.

Cuando la reparación se haya completado, el sistema preguntará por el modo de funcionamiento.

Eligiendo el modo de funcionamiento:

Se debe elegir el modo de funcionamiento tan pronto como salga el siguiente mensaje:

```
INIT: SINGLE USER MODE

Type CONTROL-d to continue with normal startup,
(or given the root password for system maintenance):
```

El sistema tiene dos modos de funcionamiento: funcionamiento normal y mantenimiento del sistema. El funcionamiento normal es la forma ordinaria de trabajar con el sistema. Este es el modo multiusuario, o sea, que permite la entrada de los usuarios. El modo de mantenimiento del sistema está reservado para trabajar siendo el superusuario y no permite la entrada de múltiples usuarios: se trabaja en modo monousuario.

Para elegir el modo multiusuario pulse <Crtl>-d. El sistema mostrará un mensaje de comienzo, y se le pedirá que introduzca el tiempo del sistema. Entonces el sistema ejecuta los comandos encontrados en los directorios /etc/rc, generando mensajes de comienzo para varios servicios del sistema, como los servicios de impresora o de red. Después el sistema mostrará el mensaje "login:". A partir de aquí, cualquier usuario se puede iniciar su sesión (el proceso está descrito en el bloque 2).

Para elegir el modo de mantenimiento de sistema se introduce la clave de acceso del superusuario. Se muestra el prompt del superusuario ("#"). Los comandos de los directorios /etc/rc no se ejecutan.

PARADA DEL SISTEMA:

Parar un sistema UNIX requiere algo más que apagar la computadora ya que hay que realizar una serie de pasos para preparar al sistema para la parada. El procedimiento exacto varía dependiendo de la configuración. Un procedimiento típico podría ser parecido al siguiente.

- Enviar mensajes de aviso a todos los usuarios presentes informando que el sistema va a ser desconectado próximamente.
- Cancelar (*kill*) todos los procesos en ejecución excepto el proceso de la consola.
- Desmontar los sistemas de archivos.
- Utilizar la orden *sync* para asegurar que toda la actividad de entrada-salida con el sistema de archivos se ha terminado.
- Apagar el computador si es necesario.

Existen dos comandos para parar el sistema: el comando *shutdown* o (bajo ciertas condiciones) el comando *haltsys*.

Usando el comando *shutdown*:

El comando *shutdown* es la forma normal de para el sistema y debe ser usada cuando el sistema s encuentra en modo multiusuario. El mismo avisa automáticamente a los usuarios mediante el siguiente mensaje:

```
Broadcast Message from root (<terminal>) on eancol <Fecha>
THE SYSTEM IS BEING SHUT DOWN NOW    !   !   !
Log off now or risk your files being damaged.
```

Esperará unos segundos, para dar tiempo a los usuarios a desconectarse, y pedirá confirmación para la parada del sistema. Después cerrará todas las cuentas y cancelará todos los servicios del sistema. Sólo se podrá apagar el sistema cuando aparezca el siguiente mensaje:

```
      **  Safe to Power Off  **
                  - or -
**  Press Any Key to Reboot  **
```

Usando el comando *shutdown* con la opción -gn, el proceso de parada comenzará después de n minutos.

Usando el comando *haltsys*:

El comando *haltsys* para el sistema inmediatamente. El comando debe usarse sólo cuando se está trabajando en modo monousuario. Si hay usuarios conectados cuando se ejecuta *haltsys*, se terminará su sesión y perderán los trabajos que estaban haciendo. Además, los servidores de red así como otros programas son terminados anormalmente y pueden crear problemas cuando rearranquen.

Al igual que el comando *shutdown*, no se debe desconectar la máquina hasta que no aparezca el mensaje:

```
      **  Safe to Power Off  **
                  - or -
**  Press Any Key to Reboot  **
```

TEMA 3. ADMINISTRACIÓN DE LOS SISTEMAS DE ARCHIVOS

MONTAR UN SISTEMA DE ARCHIVOS:

Como se ha comentado en el bloque 1, un sistema de archivos está formado por una estructura de directorios completa. Sin embargo, para poder utilizar un sistema de archivos hay que montarlo, o sea, enlazarlo a la estructura de directorios ya existente para que el sistema tenga noticia de su existencia. Para montarlo se necesita un punto de montaje (mount point), que generalmente es un directorio vacío.

Para montar un sistema de archivos se usa la orden *mount*. Su sintaxis es la siguiente:

```
mount [- r] [- t tipo] dispositivo directorio
```

donde dispositivo indica el archivo especial asociado con el sistema de archivos que se va a montar y directorio es el punto de montaje. Con la opción -r se indica que el sistema de archivos será de sólo lectura. Con la opción -f se especifica el tipo de sistema de archivos que se va ha instalar. Si se omite esta opción, se toma por defecto el mismo tipo que el del sistema de archivos raíz. Esto son los sistemas de archivos disponibles:

AFS	Acer Fast Filesystem.
DOS	DOS filesystem.
EAFS	Extended Acer Fast Filesystem.
HS	High Sierra CD-ROM filesystem.
S51K	AT&T UNIX System V 1 Kbyte filesystem.
XENIX	XENIX 286/386 filesystem.

Para desmontar un sistema de archivos se usa el comando *umount* cuya sintaxis es la siguiente:

```
umount dispositivo
```

donde dispositivo es el archivo especial asociado al sistema de archivos. Un sistema de archivos no se puede desmontar si está en uso: tiene algún archivo abierto o hay algún usuario en alguno de sus directorios.

CREAR UN SISTEMA DE ARCHIVOS:

La orden *mkfs* crea un sistema de archivos nuevo. La forma básica de la orden *mkfs* es la siguiente:

```
mkfs dispositivo tamaño
```

Al construir un sistema de archivos nuevo, hay que especificar el número de bloques que contiene. El comando *mkfs* utiliza ese número para determinar el número de bloques que reservará para inodos. Por ejemplo, supongamos que tenemos un dispositivo sobre disco flexible llamado /dev/fd0. Supongamos además que ese disco flexible puede contener un máximo de 2000 bloques. Construiremos un sistema de ficheros sobre este dispositivo así:

```
# mkfs /dev/fd0 2000
```

A continuación, sólo queda montar el sistema de archivos creado con la orden *mount*.

INTEGRIDAD DE LOS SISTEMAS DE ARCHIVOS:

Como se ha comentado en el tema 2, un sistema de archivos puede perder su integridad. Esto puede producirse por diversas razones: errores hardware, interrupciones de programas, errores humanos, ... El sistema de archivos mantiene en memoria una copia de las siguientes estructuras de los sistemas de archivos, las cuáles se actualizan periódicamente en disco:

- Cada superbloque de los sistemas de archivos montados.
- Cada inodo activo (archivos abiertos).
- Cada bloque de datos referenciado (se almacena en el área de buffers).

Las inconsistencias pueden producirse cuando la versión en disco de estas estructuras no coincide con la versión en memoria; como puede pasar cuando se produce una caída del sistema no prevista.

Existe un proceso en el sistema que se encarga de ejecutar la orden *sync* cada 30 segundos para realizar una actualización automática de las estructuras modificadas desde la memoria al disco.

En los sistemas UNIX existe un programa que permite comprobar la integridad de un sistema de archivos, el programa *fsck* (File System ChecK program). Este programa se lanza automáticamente al arrancar el sistema cuando se detecta algún error en algún sistema de archivos. *fsck* es un programa que comprueba la consistencia de los sistemas de archivos y los repara en su caso. *fsck* realiza cinco fases de comprobación básicas:

- Fase 1 - Comprueba bloques y tamaños (check blocks and sizes).
  Verifica la consistencia de los inodos: el número de enlaces, los tipos y los formatos de los inodos, etc…
- Fase 2 - Comprueba nombres completos (check path names).
  Verifica los directorios que apuntan a los inodos descubiertos previamente con error.
- Fase 3 - Comprueba la conectividad (check connectivity).
  Determina los errores producidos como consecuencia de directorios sin referenciar.
- Fase 4 - Comprueba las cuentas de referencia (check reference counts).
  Verifica la consistencia del número de enlaces en directorios y archivos.
- Fase 5 - Comprueba la lista libre (check free list).
  Verifica la existencia de bloques defectuosos y bloques duplicados en la lista de bloques libres, de bloques libres no

utilizados que deberían estar en la lista de bloques libres pero que no están y el número total de bloques libres.

Además de las fases descritas hay algunas fases secundarias.

UTILIZACIÓN DEL DISCO:

Veamos a continuación las órdenes *df*, que nos permite determinar el grado de ocupación de los sistemas de archivos; *du*, que determina el tamaño de los archivos y directorios; y *quot*, que nos muestra el número de bloques utilizado por cada usuario.

La orden *df*:

La orden *df* muestra la cantidad de espacio de disco asignado a los sistemas de archivos montados, la cantidad de espacio usado y disponible, y el porcentaje total que ha sido usado por los sistema de archivos montados. Su sintaxis básica es:

```
df [-vi] [<filesystem>]
```

Ejemplo:

```
/          (/dev/root    ):    323304  blocks    45192  i-nodes
/home      (/dev/u       ):    157358  blocks    19951  i-nodes
```

Muestra los siguientes campos: punto de montaje, dispositivo asociado, bloques (de 1024 bytes) libres, inodos libres. Un sistema de archivos no podrá llenarse completamente. Esto es así porque el sistema reserva una fracción del espacio en el sistema de archivos para permitir trabajar bien a las rutinas de asignación en el sistema de archivos. La cantidad reservada es normalmente un 10%. Cuando todo el espacio disponible está usado (excepto el área de reserva), sólo el superusuario podrá crear nuevos archivos y/o asignar bloques de datos a los archivos existentes.

La opción -i informa del número de inodos totales, usados y libres. La opción -v muestra los bloques totales, los usados, los libres y el porcentaje de bloques usados.

La orden *du*:

du muestra el número de kbytes de todos los archivos (recursivamente) de un directorio especificado o del archivo dado como <filename>. Por defecto sólo aparecen los directorios y subdirectorios. Si un archivo tiene varios enlaces, sólo se cuenta uno. Su sintaxis es la siguiente:

```
du [-s] [-a] [<filename> … ]
```

Las opciones realizan las siguientes modificaciones a la orden *du*:

-s Muestra una única entrada correspondiente al <filename> especificado.
-a Genera una entrada para cada archivo del directorio nombrado.

La orden *quot*:

quot muestra el número de bloques (de 1024 bytes) que pertenece a cada usuario del sistema de archivos especificado. Si no se especifica ninguno, se revisan todos los sistemas de archivos montados. Su sintaxis es como sigue:

```
quot [-f] [<filesystem>]
```

Con la opción -f se nos muestra el número de archivos y el espacio que ocupan para cada usuario.

Este comando sólo lo puede usar el superusuario.

ORGANIZACIÓN EFICIENTE DE LOS SISTEMA DE ARCHIVOS:

Existen dos aspectos del sistema que degradan la eficiencia de los sistema de archivos:

1. Fragmentación del disco: Cuando el archivo ha sido utilizado durante algún tiempo, la constante creación y borrado de archivos producen una fragmentación en el disco, la cual consiste en que los archivos son creados en varios pequeños pedazos en el disco duro. Es posible recuperar parte del disco (5 al 10%) realizando un backup de los archivos, removiéndolos y volviéndolos a restaurar.
2. Directorios muy grandes que incrementan el tiempo de búsqueda de un archivo: Se deben evitar los directorios más grandes de lo necesario. Se debe tener presente algunas limitaciones con respecto a tamaños. Un directorio que contenga hasta 30 entradas además de las entradas . y .. encaja en un bloque único del disco y es fácil de accesar. Un directorio puede tener hasta 286 entradas, y aún es aceptable si contiene solamente datos almacenados. Más entradas en un directorio de trabajo constituyen un desastre.

TEMA 4. ADMINISTRACIÓN DE LAS CUENTAS DE USUARIO.

INTRODUCCIÓN. ARCHIVOS ADMINISTRATIVOS:

Cada usuario que accede al sistema debe tener una cuenta de usuario (user account). El archivo que contiene información sobre cada cuenta del sistema y de los usuarios que pueden conectarse localmente a la máquina es el /etc/passwd. Las claves de acceso se almacenan en el archivo /etc/shadow. Si este archivo no existe, se almacenarán en el /etc/passwd. El archivo /etc/group almacena información acerca de los grupos de usuario existentes en el sistema.

El archivo /etc/passwd:

Contiene información sobre cada cuenta de usuario. Veamos un ejemplo de este archivo:

```
root:x:0:1:Superuser:/root:/bin/csh
daemon:x:1:1:System daemons:/etc:
bin:x:2:2:Owner of system commands:/bin:
sys:x:3:3:Owner of system files:/usr/sys:
adm:x:4:4:System accounting:/usr/adm:
uucp:x:5:5:UUCP administrator:/usr/lib/uucp:
nuucp:x:6:5:Anonymous UUCP site:/usr/spool/uucplogins/nuucp:/usr/lib/uucp/uucico
auth:x:7:21:Authentication administrator:/tcb/files/auth:
asg:x:8:8:Assignable devices:/usr/tmp:
cron:x:9:16:Cron daemon:/usr/spool/cron:
sysinfo:x:11:11:System information:/usr/bin:
dos:x:16:11:DOS device:/tmp:
mmdf:x:17:22:MMDF administrator:/usr/mmdf:
network:x:18:10:MICNET administrator:/usr/network:
nouser:x:28:28:Network user with no access privileges:/:/bin/false
listen:x:37:4:Network daemons:/usr/net/nls:
lp:x:71:18:Printer administrator:/usr/spool/lp:
audit:x:79:17:Audit administrator:/tcb/files/audit:
ingres:x:777:50:Database administrator:/usr/ingres:
games:x:42:50:Games Admin:/usr/games:/bin/rsh
julio:x:200:50:Julio Rey de Perea Campos:/home/julio:/bin/csh
german:x:201:50:G . M.  C.:/home/german:/bin/csh
pepe:x:202:50:Cuenta de prueba.:/home/pepe:/bin/csh
```

En este archivo se tendrá una entrada por cada cuenta que exista en el sistema conteniendo información de la siguiente forma:

```
username:password:uid:gid:comentario:directorio:shell
```

El significado de cada campo es el siguiente:

username	Nombre de conexión de la cuenta (nombre de usuario). Es una combinación de números y letras.
password	Palabra de paso (palabra clave) encriptada de la cuenta. Si este campo es nulo, la cuenta no se demanda password. Si existe el archivo /etc/shadow, entonces aparece el carácter 'x' y la clave encriptada junto con información adicional aparece en el /etc/passwd.
uid	Identificador de usuario de la cuenta. Es un identificador único para el sistema. Es un número entre 10 y 60000. Los identificadores del 0 al 9 se reservan para las cuentas del sistema. El uid del root es normalmente 0.
guid	Identificador del grupo al que pertenece la cuenta. Es un identificador numérico del grupo por defecto para ese usuario. Su valor puede estar entre 0 y 60000.
comentario	Nombre real del usuario y otra información que aparecerá en las cabeceras de mensaje del correo del usuario.
directorio	Nombre completo del home directory de la cuenta.
shell	Shell de ingreso de la cuenta.

El archivo /etc/shadow:

Contiene información sobre el password de cada cuenta de usuario. Veamos un ejemplo de este archivo:

```
root:jKemQiEWZNFic:9316:0:0
daemon:*::0:0
bin:*::0:0
sys:*::0:0
adm:*::0:0
uucp:*::0:0
nuucp:*::0:0
auth:*::0:0
asg:*LK**::0:0
cron:*::0:0
sysinfo:*::0:0
dos:*::0:0
mmdf:*::0:0
network:*::0:0
nouser:*:::
listen:*:::
lp:*::0:0
audit:*::0:0
ingres:*:7203:0:0
games:VQUevOs9/vJ/6iLntz0/OgMs:9318::
julio:wj214lv7C9hB2dT5UmNw6qTc:9316::
german:6KduXOtNpTNis:9316::
pepe:FsOWRXGNYqry.:9325::
```

En este archivo se tendrá una entrada por cada cuenta que exista en el sistema conteniendo información de la siguiente forma:

```
username:password:ultcambio:minimo:maximo
```

El significado de cada campo es el siguiente:

username	Nombre de usuario.
password siendo	Clave encriptada de la cuenta. Tendrá como prefijo la cadena "*LK*" cuando la cuenta está bloqueada, o la cadena "*RETIRED*" cuando la cuenta está retirada. Estará vacío este campo cuando la cuenta no posea password.
ultcambio	El número de días desde el 1 de Enero de 1970, y la fecha del último cambio en el password.
minimo	El número mínimo de días requerido entre cambios de password.
maximo	El número máximo de días en los cuales el password es válido.

El archivo /etc/group:

La base de datos que almacena información sobre todos los grupos del sistema es el archivo /etc/group. Un ejemplo de este archivo es el siguiente:

```
root::0:
other::1:root,daemon
bin::2:bin,daemon
sys::3:bin,sys,adm
adm::4:adm,daemon,listen
uucp::5:uucp,nuucp
mail::7:
asg::8:asg
network::10:network
sysinfo::11:sysinfo,dos
daemon::12:daemon
terminal::15:
cron::16:cron
audit::17:audit
lp::18:lp
backup::19:
mem::20:
auth::21:auth
mmdf::22:mmdf
sysadmin::23:
nogroup::28:nouser
group::50:ingres,julio,german,pepe
```

Cada entrada en este archivo tiene la sintaxis siguiente:

```
groupname:password:gid:listausuarios
```

siendo:

groupname	Nombre del grupo.
password	Clave encriptada del grupo (sin usar).
gid	Identificador de grupo. Debe ser único en la máquina local.
listausuarios	Lista de usuarios que pertenecen al grupo.

CREACIÓN/BORRADO/MODIFICACIÓN DE CUENTAS DE USUARIOS:

Con la utilidad administrativa *sysadmsh* se pueden realizar estas operaciones siguiendo una serie de menús. Sin embargo, vamos a ver como crear cuentas de usuario, como borrarlas o modificarlas de forma "manual".

Para crear una cuenta de usuario se deben seguir esta serie de pasos:

1. Crear una entrada con el nuevo usuario en el archivo /etc/passwd, asignándole un nombre de usuario (login name), un identificador de usuario no usado, un identificador de grupo, un directorio de trabajo (home directory) y un shell de conexión.
2. Añadir el usuario a su grupo en el archivo /etc/group.
3. Crear el directorio de trabajo.
4. Copiar los archivos de arranque correspondientes al shell de ingreso que se le haya asignado al usuario. Estos archivos se encuentran en los directorios ubicados en /usr/lib/mkuser.
5. Cambiar el propietario y el grupo del directorio de trabajo y de los archivos creados en él y asignárselo al usuario nuevo.
6. Crear password para la nueva cuenta.

Como ejemplo, vamos a crear la cuenta pepe:

- Añadir la siguiente entrada al archivo /etc/passwd:
  ```
  pepe::507:50:Cuenta de ejemplo:/home/pepe:/bin/csh
  ```

- Añadir el usuario a la entrada correspondiente en el archivo /etc/group:
  ```
  group::50:julio,german,pepe
  ```

- Crear el directorio de trabajo:
  ```
  # mkdir /home/pepe
  ```

- Copiar los archivos de arranque:
  ```
  # cp /usr/lib/mkuser/csh/cshrc /home/pepe/.cshrc
  # cp /usr/lib/mkuser/csh/login /home/pepe/.login
  ```

- Cambiar el propietario y el grupo de los archivos creados:
  ```
  # chown pepe /home/pepe
  # chown pepe /home/pepe/.cshrc
  # chown pepe /home/pepe/.login
  # chgrp group /home/pepe
  # chgrp group /home/pepe/.cshrc
  # chgrp group /home/pepe/.login
  ```

- Crear el passwd de la cuenta nueva:
  ```
  # passwd pepe
  ```

A partir de aquí, el nuevo usuario puede conectarse ya.

Para borrar un usuario, se siguen dos pasos:

1. Usar el comando *rmuser usuario* para borrar al usuario de las bases de datos.
2. Borrar el directorio de trabajo del antiguo usuario y todos los archivos creados por él.

Para modificar alguna característica de un usuario, se pueden editar los archivos correspondientes y hacer las modificaciones a mano.

CREACIÓN/BORRADO DE GRUPOS:

La forma de crear/borrar usuarios es editando el archivo /etc/group y hacer los cambios pertinentes.

La creación de grupos de usuarios perfectamente definidos, proporciona al sistema un grado de seguridad adicional permitiendo controlar el acceso a archivos y directorios de una forma más organizada.

LISTADO DE USUARIOS DEL SISTEMA:

La orden *finger*, sin especificar un nombre de usuario, muestra información sobre cada usuario que está conectado. Su sintaxis es:

finger [opciones] [<username> …]

La información que muestra es la siguiente:

- Nombre de conexión (login name).
- Nombre completo (full name).
- Nombre de terminal (terminal name).
- Tiempo de inactividad (idle time).
- Tiempo de conexión (login time).
- Posición (location). Comentario en el /etc/ttytab si se conecta localmente y hostname si se conecta remotamente.

Si se especifica un nombre de usuario aparecerá además la siguiente información.

- Directorio de trabajo y shell de conexión del usuario.
- Tiempo de la última conexión de usuario (si en ese instante no están conectados) o tiempo que llevan conectados (si están conectados).
- Terminal (e información del terminal obtenido en el /etc/ttytab) o host del que está conectado remotamente.
- La última vez que el usuario recibió y leyó el correo.
- Plan de trabajo contenido en el archivo .plan que puede encontrarse en el home directory del usuario.
- Proyecto en el que esté trabajando el usuario descrito en el .project del home directory si existe.

Opciones:

-l Fuerza el formato de salida largo.
-s Fuerza el formato de salida corto.

TEMA 5. DISPOSITIVOS.

CREACIÓN DE DISPOSITIVOS:

Como hemos visto en el bloque 1, el sistema UNIX trata a todos los dispositivos como archivos. Estos archivos están vacíos y únicamente contienen dos números, el número principal y el número secundario. El número principal identifica el tipo de dispositivo y el secundario identifica a un dispositivo concreto de ese tipo. También vimos que existen dos tipos de archivos especiales: tipo bloque y tipo carácter.

La orden *mkdev* permite crear archivos especiales asociados a dispositivos periféricos en caso de pérdida accidental o inexistencia de uno de ellos. Su sintaxis es la siguiente:

```
mkdev <dispositivo> [opciones]
```

Basándose en el dispositivo especificado, el comando *mkdev* llama a un script determinado que se encuentra en el directorio /usr/lib/mkdev. Algunos de los dispositivos que se pueden especificar son:

aio	Añade soporte para discos I/O asíncronos en el kernel.
cdrom	Añade soporte CD-ROM al kernel.
dos	Inicializa los dispositivos necesarios y configura el sistema para soportar sistemas de archivos DOS.
hd	Crea archivos especiales para usar discos duros.
lp	Añade o modifica la configuración de una impresora.
mouse	Inicializa los dispositivos necesarios y configura el sistema para usar cualquier ratón.

Esto es sólo un ejemplo de los posibles dispositivos que pueden crearse, existen más, como: cintas, puertos serie, puertos paralelo, terminales, …

TEMA 6. ADMINISTRACIÓN DE TERMINALES.

COMUNICACIÓN:

Generalmente, para la comunicación se utilizan conectores de 9 pines en el puerto del computador y 25 en las terminales. En comunicación asíncrona se tienen las siguientes líneas de comunicación:

Señal	Pin(DB-25)	Pin(DB-9)
Frame Gruond	1	3
Transmit Data	2	2
Receive Data	3	8
Clear to Send	4	7
Request to Send	5	7
Data set ready	6	6
Signal Ground	7	5
Data Carrier Detect	8	1
Data Terminal Ready	20	4
Ring Indicator	22	9

A nivel de comunicación de dispositivos, se tienen dos tipos:

- DTE - Data Terminal Equipment: Como por ejemplo los dispositivos seriales de los computadores y terminales.
- DCE - Data Communication Equipment: Como los modems y algunas impresoras seriales. La configuración de los dos tipos con respecto a pines es la misma. La única variación consiste en que lo que en el DCE es entrada, en el DTE es salida.

Al comunicar dispositivos del mismo tipo, el cable de comunicación en un tipo de protocolo debe construirse así:

DTE	DTE
1	1
2	3
3	2
4	5
5	4
6	20
7	7
20	6

Cuando se comunican dispositivos de diferente tipo, el cable de comunicación en un tipo de protocolo debe construirse así:

DTE	DTE
1	1
2	2
3	3
4	4
5	5
6	6
7	7
20	20

Los dispositivos seriales utilizan para su comunicación dos tipos de protocolo: XON y XOF.

- Protocolo XON: La terminal o el usuario envían un dato al computador con el fin de avisarle al la pronta saturación del buffer de comunicación , y en el momento de liberar dicho buffer, envía un dato para permitir que el computador vuelva a enviar el resto de comunicación , sólo se necesitan tres pines para comunicación: Transmit Data (Pin 2), Receive Data (Pin 3) y Signal Ground (Pin 7).
- En protocolo DTR, se utilizan señales lógicas a través de los cables de comunicación para avisarle al computador cuando debe transmitir y cuando debe dejar de hacerlo. En este tipo de comunicación, se necesitan casi todas las señales de comunicación presentes en los puertos seriales.

La configuración de las tarjetas de comunicación se realiza mediante el comando *mkdev serial*. Lo único que se debe tener en cuenta es el número de tarjeta (1 puerto, 2 puertos, 4 puertos, 5 puertos y 8 puertos) y la configuración de la misma (com1 o com2). No se utilizan los puertos de comunicación com3 y com4 puesto que estos dispositivos no generan interrupciones.

El sistema entonces generará los nodos correspondientes bajo el directorio /dev. Además actualiza los archivos/etc/inittab y /etc/ttytype que contiene los parámetros de cada terminal y el tipo de terminal asignado a cada puerto serial.

ARCHIVOS PARA LA GESTIÓN DE TERMINALES:

/etc/inittab:

Este archivo es leído por /etc/init, que es el proceso que atiende los dispositivos seriales. Este archivo se lee cuando:

a) Se arranca el sistema.
b) Se termina un proceso de arranque (el usuario sale del sistema).
c) El administrador del sistema lo manda a ejecutar.

Al modificar los parámetros del archivo, los parámetros son temporales. Si se quiere hacerlos permanentes, se deben hacer tanto en /etc/inittab, en los archivos del directorio /etc/conf/init.d o en /etc/conf/cf.d/init.base, los cuales son la base para la generación del archivo.

init.base:

Su estructura es la siguiente:

identificador: nivel de acción: acción: proceso.

identificador Es el identificador del dispositivo. Guarda información de cuando se utiliza este dispositivo.

nivel de acción Nivel que le va a asignar el proceso *init*. Estos niveles están en el rango del 1 al 6. Cuando a un identificador se le coloca más de una acción, deben escribirse a continuación, sin ninguna clase de separador. Niveles:

0: Invocado cuando se va a apagar el sistema.
1: Corresponde al modo monousuario o de mantenimiento.
2: Corresponde al modo multiusuario.
3: Pertenece al modo multiusuario y adicionalmente se ejecutan otros procesos pertenecientes a la compartición de recursos y procesos con otras máquinas con las cuales comparte una red de computadores.
4: Nivel alterno igual al 2.
5: Detiene el sistema.
6: Detiene el sistema y lo reinicializa en el estado indicado en la variable initdefault.
a,b,c: Procesa sólo aquellos estados en cuales se encuentran los y niveles a, b o c.
Q,q: Reexaminar el archivo.
S,s: Modo mantenimiento.

accion Puede ser de varios tipos, de acuerdo al nivel en el cual este corriendo Unix. Al arrancar el sistema puede contener cualquiera de los siguientes valores:

boot: Empieza el proceso y continua con los siguientes sin esperar su culminación.
bootwait: Empieza el proceso y antes de continuar espera que culmine el proceso.
initdefault: Determina en cual nivel se quiere entrar inicialmente el sistema. Si no tiene contenido, pregunta por el nivel.
sysinit: Empieza un proceso y espera terminarlo antes de continuar. Estos procesos se ejecutan antes de que se accese la consola.

Una vez se ha entrado a modo multiusuario los valores pueden ser:

off: Envía un mensaje de atención, espera 20 segundos y luego acaba con el proceso que esté corriendo. Si no hay procesos corriendo ignora esta línea.
once: Empieza el proceso y continúa con los demás procesos, cuando este proceso acaba, no reinicia uno nuevo.
ondemand: Exactamente igual al respawn. Funciona en otros niveles de acción.
respawn: Empieza el proceso y continúa con el siguiente en la tabla. Cuando termina, reinicia otro igual.
wait: Empieza un proceso y espera a que termine antes de seguir con la tabla.
proceso: Comando a ser ejecutado en dicho identificador.

Este es un ejemplo del contenido de este archivo

```
#       @(#)  init.base 22.3 90/01/15
#
#             UNIX es a registered trademark of AT&T
#               Prtions Copyright 1976-1989 AT&T
#       Portions Copyright 1980-1989 Microsoft Corporation
#    Portions Copyright 1983-1989 The Santa Cruz Operation, Inc.
#                   All Rights Reserved
#
# /etc/inittab on 286/386 processors is built by Installable
# Drivers (ID) each time the kernel is rebuilt. /etc/inittab is replaced
# by /etc/conf/init.d directory by the /etc/conf/bin/idmkinit command.
bchk::sysinit:/etc/bcheckrc </dev/console >/dev/console 2>&1
tcb::sysinit:/etc/smmck </dev/console >/dev/console 2>&1
ck:234:bootwait:/etc/asktimerc </dev/console >/dev/console 2>&1
ack:234:wait:/etc/authckrc </dev/console >/dev/console 2>&1
copy:2:bootwait:/bin/cat /etc/copyrights/* >/dev/console 2>&1
brc::bootwait:/etc/brc 1> /dev/console 2>&1
mt:23:bootwait:/etc/brc </dev/console >/dev/console 2>&1
is:S:initdefault:
r0:056:wait:/etc/rco  1>/dev/console 2>&1 </dev/console
r1:1:wait:/etc/rc1  1> /dev/console 2>&1 </dev/console
r2:2:wait:/etc/rc2 1> /dev/console 2>&1 </dev/console
r3:3:wait:/etc/rc3 1> /dev/console 2>&1 </dev/console
sd:0:wait:/etc/uadmin 2 0 >/dev/console 2>&1 </dev/console
fw:5:wait:/etc/uadmin 2 2 >/dev/console 2>&1 </dev/console
rb:6:wait:/etc/uadmin 2 1 >/dev/console 2>&1 </dev/console
co:12345:respawn:/etc/getty tty01 m
co2:2:respawn:/etc/getty tty02 m
co3:2:respawn:/etc/getty tty03 m
co4:2:respawn:/etc/getty tty04 m
co5:2:respawn:/etc/getty tty05 m
co6:2:respawn:/etc/getty tty06 m
co7:2:respawn:/etc/getty tty07 m
co8:2:respawn:/etc/getty tty08 m
co9:2:respawn:/etc/getty tty09 m
co10:2:respawn:/etc/getty tty10 m
co11:2:respawn:/etc/getty tty11 m
co12:2:respawn:/etc/getty tty12 m
po:2:off:/etc/getty ttyp0 m
p1:2:off:/etc/getty ttyp1 m
po:2:off:/etc/getty ttyp2 m
p2:2:off:/etc/getty ttyp3 m
p3:2:off:/etc/getty ttyp4 m
p4:2:off:/etc/getty ttyp5 m
p5:2:off:/etc/getty ttyp6 m
p6:2:off:/etc/getty ttyp7 m
p7:2:off:/etc/getty ttyp7 m
```

Lo primero que hace el sistema es ejecutar el comando *bcheckrc*, el cual es un script que evalúa el estado del sistema de archivos root, mediante el comando *fsck*. Después se asignan los diferentes comandos que se ejecutan en el momento de iniciar el sistema. Su modo default es monousuario, definido en la línea initdefault.

El archivo /etc/brc borra la antigua tabla del sistema de archivos que se encontraba en el sistema de archivos en caso de que haya sido mal apagado.

/etc/ttytype:

Este archivo define el tipo de terminal que se asignará a cada dispositivo del sistema. Cuando un usuario entra al sistema, el sistema busca que tipo de terminal se está utilizando, y lee sus características de este archivo. En XENIX lee del archivo /etc/termcap. Luego las sitúa en la variable TERMCAP. En cada archivo de arranque de los usuarios se hace una evaluación del tipo de terminal en el cual está trabajando y se asume como *ansi* la terminal por defecto. Dado el caso que no se quiera realizar la evaluación para que no haya necesidad de que el usuario escriba el tipo de terminal, se puede suprimir esta línea del archivo, pero debe definirse entonces correctamente en el archivo.

Su estructura es:

```
ansi     tty01
ansi     tty02
ansi     tty03
ansi     tty04
  :        :
wy60     tty1a
wy60     tty2a
wy60     ttyi00
```

El proceso de evaluación lee el tipo de terminal para dicho dispositivo y como tal lo maneja.

Archivos /etc/tercamp - /usr/lib/terminfo:

Estos archivos contienen las secuencias de manejo de un gran número de terminales que trabajan bajo UNIX/XENIX**.** XENIX utiliza el archivo termcap por defecto, UNIX utiliza terminfo. Cada parámetro define una característica de la terminal, como número de líneas y columnas, blink y video inverso.

Cada entrada en la base de datos, consiste de diferentes líneas que definen el tipo de terminal y el control de caracteres de secuencias que manipulan el despliegue. El archivo /etc/termcap es un archivo tipo ascii que puede ser editado. En las versiones de UNIX System V y versiones posteriores se utilizan archivos que se encuentran bajo el directorio /usr/lib/terminfo/* para el control de las terminales.

Por lo tanto, el archivo terminfo no puede ser editado, primero se debe editar el archivo terminfo.src o un archivo en el que se guarden las descripciones para las terminales, y posteriormente se debe compilar con el comando *tic*. Una vez compilado, el *tic* coloca los archivos en cada directorio.

/etc/gettydefs:

Contiene las definiciones para comunicarse con los puertos. Cada definición tiene un prefijo que corresponde al valor dado en /etc/ttys (XENIX) o en /etc/inittab (UNIX), y a continuación se tienen los parámetros bajo los cuales va a trabajar el dispositivo, como velocidad, echo, número de bits, entre otros. Contienen dos tipos de entrada:

- La primera se refiere a descripciones del protocolo DTR, utilizando descriptores numéricos.
- La segunda corresponde al manejo del protocolo Xon/Xoff y sus descriptores son letra del alfabético.

Por ejemplo:

m#etiq.ini#condi.ini.#Condi.fin.#Desp.ini.#Eti.Fin.#Programa

Etiq.inicial	Rótulo de entrada, concuerda con el archivo /etc/ttys o /etc/inittab.
Condi.ini	Utilizado mientras se intenta hacer contacto con una terminal, generalmente es la velocidad.
Condi.fin	Características operativas de la línea. SANE: Coloca algunas características de las terminales en valores estándar. HUPCL: Cierra la línea cuando se salga del sistema. TAB3: Expande los tabs con espacios, de lo contrario aparecerá el carácter ^I. IXANY: Permite que cualquier carácter reinicie la transmisión en protocolo Xon/Xoff. CS8: La transmisión se realiza a ocho bits de datos.
Desp.ini	Mensaje inicial presentado por el programa getty. Puede contener las mismas variables del comando echo para manejo de líneas, tabuladores, etc y utiliza el signo @ para separar el mensaje de la identificación que se encuentra en la primera línea del archivo /etc/systemid. Antes, el programa getty muestra la información del archivo /etc/issue.
Etiq.fin	Identifica el siguiente nivel que tomará el proceso de getty para tratar de establecer una comunicación correcta.
Programa	Es un campo adicional en el cual se puede obligar al sistema no ejecutar en el dispositivo que corresponde con esta descripción el comando /etc/login, sino cualquier otro comando, el cual debe ser precedido de la palabra AUTO.

TEMA 7. ADMINISTRACIÓN DE IMPRESORAS.

CREAR UN PUERTO PARALELO:

Para instalar una impresora es necesario tener creado un puerto paralelo. A continuación vamos a ver como crear este dispositivo. Para crear un puerto paralelo se usa el comando:

```
mkdev parallel
```

Aparecerá el siguiente menú:

1. Add a parallel port.
2. Remove a parallel port.
3. Show configuration.
4. Help.

Tenemos las opciones de añadir un puerto paralelo, eliminar un puerto paralelo, mostrar información sobre configuración y pedir ayuda. Eligiremos la opción 1 y entonces aparecerá otro menú, preguntando por el tipo de puerto paralelo que queremos crear:

1. Serial/Parallel adapter #1: address=378-37f
2. Parallel/Monochrome adapter : address=3bc-3be
3. Serial/Parallel adapter #2 : address=278-27a

Elegimos el puerto que queremos y nos preguntará a continuación por el IRQ con el siguiente mensaje:

```
Should this port use interrupt (default [5):
```

A continuación, el sistema asignará un archivo de dispositivo al puerto, por ejemplo el /dev/lp0. Por último nos informará de que el kernel a sido modificado y nos preguntará si queremos recompilarlo para almacenar los cambios.

CREAR UNA IMPRESORA:

Para crear y configurar un impresora usamos el comando siguiente:

```
mkdev lp
```

Nos aparecerá un shell parecido al *sysadmsh*: un sistema de menús y ventanas. Elegiremos la opción Configure y después la opción Add se introducirán los siguientes datos acerca de la impresora:

nombre	Nombre que queremos que tenga la impresora dentro del sistema.
comentario	Nombre completo de la impresora o cualquier comentario que se quiera hacer sobre la misma.
nombre de clase	Nombre del grupo al que se quiere añadir la impresora. Las impresoras que se vayan configurando se pueden agrupar siguiendo cualquier criterio.
usar interfaz de impresora	Se especifica el nombre de la interfaz que va a usar la impresora.
conexión	Tipo de conexión de la impresora. Puede ser directa (local) o de llamada (en red).
dispositivo	El archivo del dispositivo que va a utilizar la impresora (por ejemplo /dev/lp0).
banner	Se pregunta si se desea el servicio de banner en la impresora.

Si se quiere colocar esta impresora como la impresora por defecto, se elige la opción Configure, después la opción Default y por último se especifica el nombre de la impresora.

Para terminar de configurar la impresora hay que realizar las siguientes acciones, dentro del menú Schedule de la opción Configure:

1. Se elige la opción Begin.
2. Se elige la opción Accept y se indica el nombre de la impresora.
3. Se elige la opción Enable y se indica el nombre de la impresora.

Con todo esto queda ya creada una impresora. Los usuarios pueden usarla mediante el comando *lp* explicado en el bloque 2.

TEMA 8. COPIAS DE SEGURIDAD. RECUPERACIÓN.

INTRODUCCIÓN:

Las copias de seguridad son una de las tareas más usuales y necesarias a realizar en el mantenimiento de todo el sistema. El objetivo de toda copia de seguridad es el de poder recuperar toda la información en caso de que ésta se pierda, por ello, es aconsejable que se realicen con una cierta periodicidad y planificación. Por ejemplo, sería deseable que se hicieran copias diarias de aquellos directorios que contienen los datos de los usuarios.

Existen varios tipos de copia de seguridad:

- Copias imágenes:

Se copia toda la información de un sistema de archivos conservando toda su organización. Cuando se realiza una recuperación, el sistema de archivos quedará tal y como estaba cuando se realizó la copia. La recuperación de una copia imagen sólo se puede realizar sobre un disco o partición de disco del mismo tamaño que aquél del que se realizó la copia. Este tipo de copias de seguridad no es el utilizado en el trabajo diario. Pueden ser convenientes realizarlas tras el proceso de instalación y/o cuando el sistema se encuentra en un estado óptimo (bien configurado).

- Copias incrementales:

Las copias incrementales consisten en realizar una copia completa periódicamente (una vez a la semana o al mes) y diariamente, se hace una copia de los archivos que han sido modificados desde la última vez que se realizó la copia de seguridad. Se necesita conocer las fechas de grabación para cada archivo. Las copias incrementales no conservan la organización del disco. La recuperación se podría realizar en cualquier parte. Es recomendable que las copias incrementales se realicen con el sistema trabajando en modo monousuario. Se deben considerar los siguientes elementos cuando se planifica una estrategia de copias de seguridad:

  - Sistemas de archivos que necesitan que se les haga copias de seguridad.
  - Frecuencia con que se deben realizar las copias de seguridad.
  - Niveles de las copias incrementales de forma que se pueda recuperar la información con el menor daño.
  - Cantidad de cintas que se necesitan para implementar la planificación de copias de seguridad.

- Copias específicas:

Cuando se realizan copias de seguridad de los archivos que el operador desea guardar sin tener en cuenta la fecha de modificación u otro criterio.

COPIAS INCREMENTALES CON *backup/ restore*:

La orden *backup*:

La orden *backup* realiza copias de seguridad incrementales, de los archivos del home directory de un usuario especificado, o de un conjunto de archivos y directorios determinados. Las copias son restauradas con la orden *restore*. Su sintaxis es la siguiente:

```
backup [-h] [-t] [-p | -c | -f archivos | -u usuarios] -d dispositivo
```

Sus opciones son:

-h Produce una historia de ls Backups realizados.
-c Realiza un backup completo. Todos los archivos modificados desde la instalación del sistema son copiados.
-p Realiza un backup incremental. Copia sólo los archivos modificados desde la fecha del último backup.
-f Copia los archivos especificados en el argumento archivos.
-u Copia el home directory del usuario especificado.
-d Se usa para especificar el dispositivo de destino que se va a utilizar.
-t Se usa cuando el dispositivo especificado es una cinta.

La orden *restore*:

Restaura los archivos copiados con la orden *backup*. Su sintaxis es la siguiente:

```
restore [-c] [-i] [-o] [-t] [-d] dispositivo]
```

Sus opciones son:

-c Realiza una completa restauración. Todos los archivos de la copia son restaurados.
-i Obtiene una lista de los archivos existentes en la copia. Ningún archivo es restaurado.
-o Sobreescribe los archivos existentes. Si esta opción no se especifica, los archivos de la copia que ya existan no son restaurados.
-t Indica que el dispositivo donde está la copia es una cinta.
-d Especifica el dispositivo donde se encuentra la copia de seguridad.

COPIAS ESPECÍFICAS CON *tar* Y *cpio*:

La orden *tar*:

La orden *tar* crea archivos de cinta (tarfile) y añade o extrae archivos. Un tarfile es habitualmente una cinta magnética pero puede ser un archivo.

Su sintaxis básica es:

```
tar c|t|u|x[opciones] [tarfile] archivos
```

Letras de función:

c Crea un nuevo tarfile y escribe los archivos en él.
t Lista la tabla de contenidos de tarfile.
u Los archivos especificados son añadidos al tarfile si éstos no están aún o si han sido modificados desde la copia en este tarfile.
x Extrae los archivos nombrados desde el tarfile. Si no se especifican archivos, se extraerán todos los archivos.

Si un archivo especificado es un directorio, se copiará o se recuperará recursivamente, es decir, se tratarán todos los archivos y subdirectorios que hayan en él.

Opciones:

v Normalmente *tar* trabaja silenciosamente. Con esta opción, *tar* mostrará el nombre de cada archivo.
f Usa el próximo argumento como nombre del tarfile.

La orden *cpio*:

cpio realiza copias de archivos a un archivo de copia de cpio. Los archivos de copia de cpio contienen rutas de archivos e información de estado, junto con el contenido de los archivos almacenados. Su sintaxis es la siguiente:

```
cpio -i | -o [opciones]
```

Sus comandos son:

-o Copia archivos a la salida estándar. Normalmente se redirecciona a un dispositivo de copia de seguridad. Los archivos a copiar los obtiene de la entrada estándar.
-i Extrae archivos de la entrada estándar. Los archivos extraídos son copiados en el directorio actual. Se pueden especificar los archivos que se quieren recuperar. Cada etiqueta de archivo debe ir entre comillas dobles.

Las opciones de “cpio -o” más importantes son:

a Actualiza las fechas de acceso de los archivos cuando se completa la copia.
c Escribe una cabecera en ASCII.
v Informa de los archivos que se están copiando.

Las opciones de “cpio -i” más importantes son:

d Crea los directorios necesarios.
r Renombra archivos interactivamente. Si da el carácter ‘.’, se copiará el nombre original. Si el usuario de una línea vacía, se salta el archivo.
t Muestra los archivos existentes del archivo de entrada especificado. No se recupera ningún archivo.

- u Copia incondicional. Normalmente no se recupera los archivos cuya fecha de modificación en disco sea más nueva que en el medio de recuperación. Con esta opción, se recupera sin tener en cuenta la fecha de modificación.

TEMA 9. EJECUCIÓN PERIÓDICA DE TAREAS. LA UTILIDAD *cron*.

INTRODUCCIÓN:

El UNIX System V permite la ejecución periódica de tareas. La utilidad *cron* es un proceso permanente, iniciado al arrancar el sistema, que se activa una vez cada minuto y consulta los archivos del directorio /usr/spool/cron/crontabs para hallar las tareas que hay que ejecutar y cuándo hay que hacerlas. Si el momento es el definido, *cron* iniciará las tareas pertinentes.

Los archivos contenidos en el directorio /usr/spool/cron/crontabs se denominan habitualmente archivos crontab. Existe un archivo para cada usuario que precise ejecutar trabajos periódicos. Cada línea en un archivo crontab consta de 6 campos separados por espacios o tabuladores y su significado es el siguiente:

```
minuto  hora  día_mes  mes  día_semana  orden
```

donde:

minuto	Minuto en el que se ejecutará la orden. Su valor será 0-59.
hora	Su valor será de 0-23.
día_mes	Su valor será de 1-31.
mes	Mes del año. Su valor será de 1-12.
día_semana	Su valor será de 0-6. El valor 0 corresponde al domingo. Este campo complementa al campo día_mes puesto que puede interesar que se ejecute una orden el día 1 y todos los lunes.
orden	Orden a ejecutar.

Los campos 1 al 5 pueden tener una lista de valores separados por comas, un rango de valores o un asterisco significando en esta caso que la tarea se ejecuta para todos los posibles valores del campo. Si se especifica un día de la semana (en el campo día_semana) y en el campo día_mes se pone un asterisco, se considerará sólo el día de la semana. Los resultados de la orden aparecerán en la salida estándar si ésta no se redirecciona. Ejemplos:

```
0     0     *     *     *     calendar
15    4     *     *     *     /usr/etc/sa -s > /dev/null
0     0     *     *     1-5   /usr/local/weekdays
40    0     *     *     0,6   /usr/local/weekends
```

La orden *calendar* se ejecuta a medianoche cada día. *sa* se ejecuta a las 04:15 horas de cada día. *weekdays* se ejecuta a medianoche de lunes a viernes de cada semana. *weekends* se ejecuta los sábados y domingos de cada semana a las 00:40 horas.

Si la orden va a ser ejecutada una sola vez, se puede usar para ese caso la orden *at* (explicada en el bloque 2).

LA ORDEN *crontab*:

El sistema dispone de una orden, llamada *crontab*, que permite crear, editar, borrar o listar un archivo crontab de usuario. Su sintaxis es la siguiente:

```
crontab [-u usuario] [-lr]
```

crontab toma los comando que se van a ejecutar en el tiempo de la entrada estándar. Opciones:

-u usuario	Permite especificar el usuario al que se le va a modificar su crontab. Sólo puede hacerlo el superusuario.
-l	Lista el archivo crontab del usuario.
-r	Elimina el archivo crontab del usuario.

Si existe el archivo /usr/lib/cron/cron.allow, sólo los usuario cuyo nombre de usuario aparezcan en éste (un nombre por línea), podrán usar la orden *crontab*. Si este archivo no existe, *crontab* comprobará el archivo /usr/lib/cron/cron.deny para determinar qué usuarios se les deberá denegar el uso de *crontab*. Si ninguno de los dos archivos existe, sólo el superusuario podrá utilizarlo. Si cron.allow no existe y cron.deny existe pero está vacío, podrán ejecutarlo todos los usuarios.

TEMA 10. COMUNICACIÓN CON LOS USUARIOS.

INTRODUCCIÓN:

Una de las responsabilidades del administrador del sistema es crear la mayoría de las noticias (news) del sistema. Si hay otros usuarios en la máquina, es cortesía anunciar los tiempos de parada planificados, la existencia de nuevo software y otra información actual. La orden *news* suele utilizarse con este fin, puesto que los usuarios obtienen generalmente los anuncios de noticias cuando se conectan a la máquina.

Otra característica que suele utilizarse para noticias de mayor prioridad es el *motd* (mensaje del día), que se visualiza automáticamente para cada usuario cuando éstos se conectan.

Para anuncios de muy alta prioridad se puede utilizar la orden *wall* (mensaje para todos). La orden *wall* lee un mensaje de su entrada estándar e inmediatamente lo escribe directamente al terminal de todos los usuarios que están actualmente presentes en la máquina. Esta orden debería ser utilizada para mensajes de valor inmediato, tal como una desconexión inminente del sistema.

LA ORDEN *news*:

La orden *news* permite a un usuario leer mensajes publicados por al administrador del sistema. Se puede ejecutar *news* para visualizar todas las noticias creadas desde la última vez que se ejecutó la orden, pero no las noticias antiguas que ya hayan sido examinadas. He aquí un ejemplo:

```
# news

junta (pat) Mon Jun 18 08:26:47 colombia 1990

El lunes a las 9:00 A.M. es la revisión de las estrategias comerciales.
Espero que todos estéis alli con comentarios y propuestas.   ¡Gracias!  -pat
#
```

Después se haber visto los mensajes una vez, no serán vueltos a visualizar si se ejecuta news de nuevo. La orden *news* incluye algunas opciones para modificar su comportamiento implícito, estas son:

-n Hace que *news* liste únicamente los nombres de las noticias que aún no han sido examinadas.

-s Proporciona un recuento de las noticias aún por leer, sin listar sus nombres. Una de estas dos opciones suele ser utilizada en el archivo de conexión (como el .profile o el .cshsr) de modo que las noticias por leer se visualicen cada vez que se efectúe una conexión ante la máquina.

-a Hace que *news* visualice todas las noticias disponibles, hayan sido o no leídas previamente.

Cualesquiera otras opciones para la orden *news* se supone que son los nombres de los items específicos que *news* va a visualizar, como en este ejemplo:

```
# news comida

comida (jim) Mon Jun 18 08:25:31 colombia 1990

No lo olvides, hoy es el  día del gran almuerzo y fiesta
Nos vemos alli…                                  =jim
#
```

La orden *news* funciona manteniendo un archivo de longitud cero de nombre .news_time en el home directory. El tiempo de modificación de .news_time toma como valor la hora y fecha en que la orden *news* se ha ejecutado. Cuando se ejecuta *news* de nuevo, sólo los archivos más recientes que este archivo son visualizados, excepto cuando se emplea la opción -a. Si no existe .news_time, se crea al ejecutar *news*.

Las noticias disponibles para la orden *news* se guardan en el directorio /usr/news. Cada noticia es un fichero separado en este directorio y el nombre del fichero es el nombre de la noticia, como se muestra aquí:

```
# ls -l /usr/news
total 3
-rw-r--r--   1  jim      other        74  Jun  18  08:25  comida
-rw-r--r--   1  pat      other       125  Jun  18  08:26  junta
-rw-r--r--   1  steve    other        32  Jun  18  08:24  bienvenida
```

El administrador del sistema es generalmente el responsable del mantenimiento de este directorio y el autorizado a suprimir las noticias obsoletas. En este caso el directorio de noticias sólo puede ser modificable por el superusuario, pero debería ser legible y ejecutable por todos los usuarios. Sin embargo, en algunos sistemas el directorio /usr/news es públicamente modificable, de modo que todos los usuarios pueden crear noticias dejando ficheros en el directorio. En cualquier caso, los archivos creados en el directorio deben ser públicamente legibles o si no la orden *news* fallará.

EL MENSAJE DEL DÍA:

Otra facilidad con una función análoga es el mensaje del día. Aunque no existe una orden específica, la mayoría de los sistemas incluyen la facilidad. El mensaje del día se utiliza generalmente para dar a conocer mensajes de prioridad mayor que las noticias, tales como la hora prevista de desconexión o los cambios de última hora en el sistema. En la práctica, el archivo de conexión incluye la siguiente línea:

```
cat /etc/mot
```

Como el archivo de conexión se ejecuta cada vez que un usuario abre una sesión en la máquina, el contenido del fichero motd se visualiza cada vez. A diferencia de *news*, este fichero se visualiza cada vez que el usuario se conecta, de modo que los mensajes fuera de fecha contenidos en este fichero tienden a ser tediosos muy rápidamente cuando los usuarios se presentan ante la máquina repetidamente. El fichero /etc/motd debería de ser legible por todos pero mantenido (y, por tanto, modificable) sólo por el superusuario.

LA ORDEN *wall*:

La orden *write* sólo permite comunicaciones con un sólo usuario designado como argumento de la línea de orden. Ocasionalmente puede ser necesario enviar un mensaje a todos los usuarios presentes en la máquina. Por ejemplo, las desconexiones del sistema deberían ser anunciadas para advertir a todos los usuarios que aseguren sus sesiones antes de que el sistema sea desconectado[10]. En vez de hacer un *write* a cada usuario individualmente, el sistema UNIX dispone de la orden *wall*. Esta orden sólo es utilizable por administrador del sistema. Al igual que *write*, *wall* lee de la entrada estándar el mensaje a enviar del modo siguiente:

```
# wall < fich.mensaje
```

Luego envía un mensaje a todos los usuarios que estén actualmente presentes, incluyendo a su autor. No hay utilidad que permita a los receptores de *wall* contestar al remitente.

[10] Esto lo realiza automáticamente el comando *shutdown*.

TEMA 11. SEGURIDAD DEL SISTEMA.

INTRODUCCIÓN:

El sistema operativo UNIX presenta avances en el aspecto de seguridad, pero es indudable que si no se combinan con una buena administración, puede presentar puertas abiertas que permitan errores ocasionales o intencionados con la información que allí se maneja.

Los delitos con los computadores ocurren a pesar de los mejores esfuerzos que se haga para prevenirlos, descubrirlos e impedirlos. Las víctimas, tanto individuos como organizaciones, tiene disponibles más estrategias de las que comúnmente conocemos y utilizamos para asegurar la eficiencia en alto grado de nuestros procedimientos de manejo de información.

DEFINICIÓN DE USUARIOS:

En un sistema UNIX, con un nivel de seguridad tradicional, existen tres archivos para la creación y control de usuarios y dada su importancia se deben tener en cuenta las siguientes recomendaciones:

/etc/passwd:

- El administrador del sistema con clave de superusuario debe ser el único que debe tener los permisos para modificar este archivo.
- Tener una copia de respaldo en disquete e impresora para verificar que no ha sufrido modificaciones o que sirva para procesos de recuperación.
- Es importante estandarizar los nombres de los usuarios.
- Definir grupos de usuarios con sus respectivos privilegios.
- Identificar y remover o bloquear usuarios que no han sido usados o que no son necesarios; estas cuentas que no son usadas pueden proveer fácilmente acceso a intrusos.
- Guardar en un directorio especial con protección contra lectura, escritura y ejecución.

/etc/shadow:

- El administrador del sistema con clave de superusuario debe ser el único que debe tener permiso para modificar este archivo.
- Tener una copia de respaldo en disquete e impresora para verificar que no ha sufrido modificaciones o que sirva para procesos de recuperación.
- Verificar que este archivo no tenga permisos de lectura para los usuarios.
- Dar los parámetros para obligar al usuario a que cambie regularmente el password.
- Detectar o identificar usuarios sin password y obligarlos a su definición.
- Guardar en un directorio especial con protección contra lectura, escritura y ejecución.

/etc/group:

- El superusuario debe ser el único que debe tener los permiso s para modificar este archivo.
- Tener una copia de respaldo en disquete e impresora para verificar que no ha sufrido modificaciones o que sirva para procesos de recuperación.
- Estandarizar los nombres de los grupos con nombres claros.
- Revisar los grupos que no tienen usuarios y depurarlos si es el caso.
- Definir perfiles de usuarios para clasificar los usuarios en los grupos.
- Identificar y reubicar los usuarios que tengan el mismo grupo de usuarios del sistema.

RECOMENDACIONES DE COMANDOS DE UNIX:

Es importante que los usuarios o auditores de sistemas tengan un conocimiento de algunos comandos básicos y de administración de UNIX, para lograr realizar una buena auditoría del sistema. Estas son algunas consideraciones a tomar en cuenta:

- Tener una copia de seguridad de los archivos de usuario para evitar pérdidas por el mal uso de los comandos.
- Capacitar al personal usuario de UNIX sobre la importancia de generar periódicamente un archivo con el comando *l -i > archivo_respaldo*, pues con él puede recuperarse información valiosa, frente a un accidente o problemas del sistema.
- Restringir el acceso a comandos como *ps* y *kill*.
- Para usuarios especiales, en el archivo de arranque (.profile, .login, etc.) debe tener la forma de que ubique al usuario en la aplicación que le corresponda, con el tipo de shell asignado por el administrador.
- Verificar que las variables de ambiente están bien definidas para cada usuario.
- Enviar mensajes por terminal o por correo para recordar a los usuarios que se deben salir correctamente del sistema antes de apagar la terminal.
- Capacitar al administrador para que haga buen uso del comando *ps*.
- Crear un shell script para generar reportes estadísticos producto del comando *ps* y realizar los seguimientos a que haya lugar.
- Chequear los procesos en background.
- Identificar y comparar los archivos ocultos.
- Comparar los archivos sensibles para identificar en que hora se produjo la última modificación.
- Limitarse el permiso de escritura a nivel de usuario, grupo y otros sobre los archivos de arranque para evitar su alteración por parte de los mismos. Para propósitos de auditoría, obtenerse copia impresa del contenido de estos archivos con alguna regularidad.
- Listar por impresora los archivos anteriores para respaldar papeles de trabajo de la auditoría y para facilitar la labor de análisis por parte del auditor de la información contenida en tales archivos.

SEGURIDAD EN ARCHIVOS:

Estas son las consideraciones que se deben tomar en cuenta:

- Definir qué usuarios pertenecen a un grupo y qué permisos deben tener sobre los archivos.
- Revisar archivos importantes y verificar sus privilegios.
- Crear directorios de trabajo con privilegios de lectura, escritura y ejecución solamente para el propietario, para que ningún otro usuario tenga acceso.
- Definir cuales deben ser los directorios protegidos contra borrado.
- Restringir el cambio de grupo a los usuarios.
- De los archivos básicos de las aplicaciones o del sistema, se debe establecer un procedimiento para verificar los siguientes atributos:
  - Cambio en el nombre del archivo o directorio.
  - Cambio en el tamaño de los archivos.
  - Cambio en la estructura de los permisos.
  - Cambio en el mtime y ctime.
  - Comparar con los permisos originales.
- Mantener un log de los archivos para evaluar si los permisos se encuentran autorizados o si las modificaciones temporales efectuados por el administrador del sistema se encuentran todavía, es decir, nos permite verificar que los permisos dados a los archivos de los diferentes usuarios correspondan al nivel de privilegios que estos posean.
- Mantener un respaldo del árbol de organización de los archivos y sus respectivos permisos, permitiendo así en caso de cualquier eventualidad una forma de mostrarnos cuál era el estado de los permisos para los diferentes archivos y directorios del sistema es una etapa precedente.
- No es aconsejable ceder ciertos archivos a determinados usuarios, ya que estos pueden cambiar los niveles de permisos y el usuario que cedió el archivo ya no tendría ninguna facultad de manipulación del archivo.
- El superusuario debe estar haciendo un permanente chequeo de los permisos otorgados a los usuarios sobre los archivos, para estar controlando las modificaciones efectuadas y los privilegios denominados públicos que un momento dado se le asigna a los archivos.

ADMINISTRACIÓN DEL SISTEMA:

Recomendaciones generales:

- Revisar periódicamente el encendido y apagado de la máquina.
- Restringir el acceso al archivo asktimer, que guarda la fecha y hora del sistema.
- Verificar qué usuarios llevan mucho tiempo ocioso conectado al sistema.
- Encriptar los archivos que contienen información crítica o van a ser transmitidos vía módem.
- Revisar periódicamente el espacio libre de los sistemas de archivos (no debe sobrepasar el 80%) y la tabla de inodos (no debe pasar del 90%).
- Cuando se sobrepase los límites establecidos de espacio en disco se debe:
  - Borrar los archivos core, logger, messages y archivos que no se requieran en el sistema.

  - Buscar los archivos grandes de usuarios y solicitar a los usuarios depurar los archivos grandes.
- Generar una lista de errores y las acciones tomadas para futuras situaciones.

Configuración del sistema:

- Mantener respaldo de los archivos que nos muestran los parámetros del kernel y la distribución de los sistemas de archivos del disco (mtune, sysdef, divvy).
- Establecer políticas claras sobre la distribución de los sistemas de archivos del disco.
- Mantener un respaldo en disco duro y un dispositivo magnético del archivo de configuración del sistema UNIX.

Archivos del sistema:

Los siguientes archivos se deben imprimir como papeles de trabajo y como soporte para una futura instalación y soluciones de problemas.

- group
- passwd
- hwconfig
- swconfig
- mtune
- sysdef
- divvy
- hosts
- inittab
- messages
- lpstat

Respaldo:

- Se deben generar los discos de emergencia que contengan:
  - Sistemas de archivos.
  - Boot.
  - Root filesystem only.
- Acerca de la creación de los discos de emergencia, se recomienda:
  - Formatear los discos previa copia.
  - Generar dos copias como mínimo de cada disco.
  - Tener disco de emergencia fuera de las instalaciones.
  - Realizar pruebas periódicas de las copias realizadas.
- Debe desmontarse el sistema de archivos del sistema, después de cumplida la función restauradora del mismo.
- Documentar las modificaciones del kernel.
- Restringir el acceso para algunos usuarios al *sysadmsh*.
- Imprimir los reportes de auditoría que UNIX nos ofrece.
- En los discos de emergencia, copiar el editor *vi*.

RECOMENDACIONES ADICIONALES:

- En UNIX se recomiendan los siguientes criterios para la definición de password:
  - No utilizar nombres de familiares o gente cercana, número de teléfono, códigos de empleados, nombres de hobbies, palabras populares del medio.
  - Se recomienda utilizar palabras no utilizadas en el medio en forma inversa, o abreviaciones de expresiones. Además se debe agregar caracteres no alfabéticos en la mitad o final del password, estas claves deben ser fáciles de recordar por el usuario.
- Cada vez que se efectúen modificaciones al kernel es importante generar copia impresa y magnética de los nuevos parámetros contenidos en el archivo mtune. Esto debería implementarse mediante un shell script que generará un archivo de salida a través de la ejecución del comando *sysdef*.
- Restringir el acceso físico al computador.
- Fijar los permisos de acceso a sus directorios y archivos, de tal manera que ellos puedan ser accesados solamente por el propietario o en otro caso por el grupo u otros usuarios. No dejar archivos públicos de escritura, sólo en algunos casos cuando exista una buena razón.
- Asignar password a todos los usuarios y cambiarlos periódicamente.
- Los usuarios deben chequear periódicamente la última vez que se accedió al sistema a través de su login, para identificar si ha habido un uso desautorizado de su login, y cuanto tiempo transcurrió.
- Los usuarios, quienes pueden hacer uso del comando *su*, pueden ingresar con clave de root, comprometiendo la seguridad de sus sistema, ya que pueden modificar archivos con el desconocimiento de su propietario. Por esta razón es recomendable bloquear este comando, o generar un log donde se guarde la utilización de este comando, se debe chequear el archivo /var/adm/sulog donde se guarda la información permitida.
- Colocar una apropiada máscara en el archivo de arranque para fijar un nivel de seguridad para la creación de archivos y directorios.
- No montar un medio (tal como un disquete, cinta, etc.) hasta que su contenido no se verifique. Estos dispositivos pueden contener archivos con identificadores de usuarios desconocidos, o pueden dañar información actualizada.
- No adicionar paquetes o programas de origen desconocido.
- Revisar todos los servicios de conexión de la red con el sistema.
- Debe conservarse como papeles de trabajo por parte del auditor todos aquellos listados a que hemos hecho referencia en los puntos anteriores.

BLOQUE 4: REDES EN UNIX.

TEMA 1. ARQUITECTURA DE TCP/IP.

VISIÓN GENERAL DE LAS REDES DE COMUNICACIONES.

Una red es un medio de transmisión de datos que permite a los ordenadores compartir recursos e información. Los medios de transmisión que se emplean en las redes de comunicaciones son:

- Cable de par trenzado:

Está formado por dos hilos de cobre recubiertos cada uno de aislante. El par se trenza para reducir interferencias. Existen distintos modelos. El mejor puede alcanzar velocidades de hasta 100Mbps en distancias cortas.

- Cable coaxial:

Consta fundamentalmente de un hilo central de cobre, rodeado de un dieléctrico, una malla de cobre y un aislante. Existen dos tipos de cable coaxial; el cable coaxial de 50 ohms y el de 75 ohms.

- Radioenlaces y satélites:

Estas dos técnicas se caracterizan por tener como medio de transmisión las diferentes capas de la atmósfera. Existen distintas bandas de frecuencia por las que se pueden transmitir señales de TV, radio, satélite, etc…

- Fibra óptica:

Es un delgado filamento de material dieléctrico (generalmente vidrio) envuelto por una cubierta plástica. El interior de la fibra está dividida en dos cilindros concéntricos de distinto índice de refracción. Por sus características, permite unas velocidades de transmisión muy altas y gran inmunidad a interferencias electromagnéticas.

A los computadores que forman parte de una red se les conoce como host o nodo. Una red formada por computadores con la misma arquitectura se conoce como red homogénea. Si por el contrario, los computadores tienen distintas arquitecturas, será una red heterogénea. Un protocolo de red es un conjunto de reglas para la comunicación en una red entre dos hosts. Un protocolo

puede definir cómo se identifican los hosts en una red; la forma que tomarán los datos en la transmisión; o bien, cómo esta información se debe procesar en el destino final

Las redes se pueden clasificar según su alcance en:

- Redes locales (LANs). Desde 4M-2Gbit/s. Se utilizan en edificios, campus, ...
- Redes metropolitanas (MANs). Desde 56k-100Mbit/s. Se utiliza en ciudades.
- Redes de área amplia (WANs). Desde 9,6-45kbit/s. Se utiliza a cualquier distancia.

También se pueden distinguir las redes según el tipo de conmutación usada. Existen tres tipos:

- Redes con conmutación de circuitos: Si dos ordenadores quieren comunicarse entre sí, deben establecer previamente una línea permanente o dedicada (canal físico), durante el tiempo de la transmisión de información, por la que circularán los mensajes (como cuando queremos hablar por teléfono). La información no se comprueba y por tanto, no hay retardos de tiempo.
- Redes con conmutación de paquetes: La información se fragmenta en unidades de información llamadas paquetes que son enviados a través de la red. En la red existe un control de flujo y de errores. La red proporciona a cada paquete para su transmisión un enlace físico. Dependiendo de que el enlace se mantenga o no durante la comunicación, para los paquetes que componen la información, hablaremos de técnica datagrama o circuito virtual:
  - Datagrama: Los paquetes se pueden transmitir por distintos canales físicos.
  - Circuitos virtuales: Los paquetes que corresponden a una misma comunicación están asignados al mismo enlace físico, o sea, se establece para cada comunicación un canal lógico.
- Redes con conmutación de mensajes: La información se fracciona en unidades de información, llamadas mensajes, de longitud variable. La red, al igual que en la conmutación de paquetes, provee un canal. Los mensajes se transfieren de forma completa de nodo a nodo, donde se comprueban y se corrigen los errores. Los mensajes en un determinado nodo no se destruyen hasta que no se haya comprobado que han sido recibidos correctamente en el posterior.

TIPOS DE REDES SOPORTADOS POR UNIX:

Una red de UNIX es un grupo de al menos dos estaciones de trabajo conectados por algún mecanismo como puede ser un cable. Hay varios tipos de estaciones de trabajo:

- Servidor: Es una estación con disco que suministra servicios (como correo, acceso a sistemas de archivos, acceso a bases de datos, …) a otras máquinas en una red.
- Cliente: Es una estación que solicita recursos de red a un servidor, como puede ser correo, almacenamiento en disco, etc.
- Standalone: Una estación con sus propios recursos, como puede ser su propio disco, sistema de archivos, correo, bases de datos, etc. Se pueden conectar a una red. En algunos casos, una estación standalone puede actuar como un servidor para otras máquinas cliente.

Algunos de los tipos de redes que soporta UNIX son:

- Redes de área local (LANs): Es un sistema de comunicaciones que permite que un número de dispositivos independientes se comuniquen directamente unos con otros, dentro de un área geográfica de tamaño moderado y sobre un canal físico de comunicaciones de velocidad también moderada. La forma de conectarse suele ser mediante cable. Dos tipos de redes LAN comunes a UNIX se basan en las tecnologías Ethernet y FDDI. Tales redes permiten la comunicación entre máquinas en tiempo real; con lo cual, se pueden realizar operaciones como acceder a un archivo que esté almacenado en el disco de otro ordenador, usar el disco de otro ordenador para almacenar sus propios archivos, imprimir trabajos tanto en una impresora local como en la de otro ordenador, enviar mensajes entre los usuarios o grupos de usuarios que forman parte de la red, trabajar con aplicaciones distribuidas basadas en el modelo cliente/servidor, etc.
- Redes basadas en ARPANET: En 1969 ARPA (Advanced Research Projects Agency) del Departamento de Defensa de los EE.UU. patrocinó el desarrollo de un protocolo de comunicación y red de banda ancha. Su descendiente se llama Internet, el cual está compuesto de un número de redes interconectadas que usan el protocolo Internet (IP). Los usuarios en máquinas que usan Internet, por cable u ondas; pueden conectarse en tiempo real a otras máquinas de la red.
- Redes UUCP: La red UUCP (Unix to Unix Copy Program) no es un sistema hardware permanente que conecta máquinas, pero permite a las máquinas usar teléfonos para transmitir datos. No es una red en tiempo real; por tanto, no podemos conectarnos a otra máquina con UUCP. Se puede enviar información a otra máquinas y recibir respuestas de ellas. Se puede usar redes UUCP para enviar correo sobre redes basadas en ARPANET.

INTRODUCCIÓN A TCP/IP:

TCP/IP fue desarrollado originalmente por el Dpto. de Defensa de los EE.UU. El término Internet describe a la familia de protocolos junto con la red; sin embargo se usa el término TCP/IP (por los dos protocolos más utilizados a nivel de transporte y físico) para referenciar a la colección de protocolos Internet. TCP/IP proporciona servicios para muchos tipos diferentes de hosts conectados a redes heterogéneas. Estas redes pueden ser redes de área ancha, como las basadas en X.25, o bien redes locales. TCP/IP se diseñó cumpliendo los siguientes requerimientos:

- Independencia de la tecnología usada (satélite, fibra óptica, línea conmutada, etc)
- Universalidad, es decir, cualquier ordenador de cualquier tipo y marca podrá usar esta red para el intercambio de información.
- Robustez respecto a fallos de los sistemas o conexiones individuales.
- Transporte fiable entre dos ordenadores finales.
- Protocolos de aplicaciones estándar.

Cada máquina UNIX inicia TCP/IP a través de los scripts /etc/rc.boot y /etc/rc.local. Para la gestión de Internet existe un proceso del sistema de servicios Internet, *inetd*, que se ejecuta normalmente en tiempo de arranque a través del script /etc/rc.local. Este proceso escucha las posibles conexiones en las direcciones Internet de los servicios que especifique su archivo de configuración. Cuando se detecta una conexión, llama al proceso servidor especificado para el servicio solicitado y, dependiendo del servicio, *inetd* seguirá escuchando en el buzón o esperará hasta que el servidor haya finalizado. Esto se hace para evitar un exceso de procesos ejecutándose a la vez.

HARDWARE EN UNA RED BASADA EN TCP/IP:

Las redes pueden ampliarse enlazando redes de área local a través de un computador llamado IP router. Se conoce como un internetwork a una configuración de red consistente de varias redes enlazadas por routers. Cuando queramos ampliar la red local o queramos crear un internetwork es posible que se necesite el siguiente hardware:

- Repetidor: Es un dispositivo usado a nivel físico de la red para conectar dos segmentos de red. Permite por tanto extender la longitud máxima de la red. Se encarga de copiar cada bit del paquete de un segmento o subred a otro; o dicho de otra forma, es un regenerador de señal a nivel físico.
- Puente: Es un dispositivo usado a nivel de enlace de datos de la arquitectura de red. Copia paquetes selectivamente desde una red a otra del mismo tipo (con el mismo MAC pero puede tener diferente medio de transmisión). Extiende la longitud máxima de la red (incluye las funciones de un repetidor).
- Enrutador: Trabaja a nivel de red. Es un dispositivo que envía paquetes desde una red lógica a otra; o sea, realiza operaciones de internetworking. Por tanto, puede conectar LANs con diferentes MAC (por ejemplo, IEEE 802.3 con Token Ring) e incluso con diferentes topologías. Una red lógica trabaja sólo con un protocolo particular. Normalmente existe una red lógica por cada red física.
- Gateway: Este dispositivo y su software asociado, permite que se puedan comunicar redes que usen diferentes protocolos; o sea, realiza conversiones de protocolos. Pueden llegar a diferir en todos los niveles de la arquitectura de red.

ARQUITECTURA DE TCP/IP. RELACIÓN CON EL MODELO OSI:

El modelo OSI:

Estudiemos brevemente, en primer lugar, el modelo de referencia OSI[11]. (Open System Interconnection) de ISO (Internacional Standard Organization). ISO es una organización que produce estándares internacionales. Otra organización similar es el IEEE (Institute of Electrical and Electronic Engineers). También se encuentra el CCITT (Comité Consultivo Internacional Telegráfico y Telefónico) que es un comité que hace recomendaciones sobre teléfono, telégrafo e interfaces de comunicación de datos. ISO es un miembro de CCITT.

El modelo OSI pretende simplemente definir un conjunto de mecanismos que hagan posible la interconexión de sistemas informáticos heterogéneos, utilizando medios de transmisión de datos. En el diseño, ISO ha tenido en cuenta de que su arquitectura permitiera la utilización de las diferentes normas emitidas por otros organismos internacionales, especialmente el CCITT.

OSI dispone de siete niveles que se han definido según el nivel de abstracción que implementen. Cada nivel ofrece una serie de servicios. Veamos estos niveles junto con algunos de los servicios más importantes de cada nivel:

[11] Existen otras arquitecturas de red que pueden ser de interés su estudio aunque el que se utiliza como referencia es normalmente la arquitectura OSI. Algunas de estas arquitecturas son la SNA, diseñada por IBM, la DNA de Digital y la DSA de Bull.

APLICACIÓN	ISO 8571 ISO 9040 ISO 8572 ISO 9041 ISO 8831 CCITT X.400 ISO 8832
PRESENTACIÓN	ISO 8822 ISO 8823
SESIÓN	CCITT X.125 / ISO 8326 CCITT X.225 / ISO 8327
TRANSPORTE	CCITT X.214 / ISO 8072 CCITT X.224 / ISO 8073
RED	CCITT X.25 (ISO 8208)
ENLACE	ISO 8802
FÍSICO	CCITT X.21 TOKEN RING RS-232-C TOKEN BUS ETHERNET

Figura 1.1: Arquitectura del modelo OSI.

- Nivel físico:

Es el encargado de la transmisión de series de bits por un canal de comunicación.

- Nivel de enlace:

Recoge los bits proporcionados por el nivel 1 y los transforma en un conjunto que aparece libres de errores de transmisión para pasarlos al nivel de red. Esto se consigue realizando la partición de los datos de entrada en unidades de información llamadas tramas de datos. En el nivel de enlace de las redes de área local se añaden normalmente dos responsabilidades, no consideradas en redes de área amplia, el control de acceso al medio (debido a que pueden existir varias comunicaciones simultáneas) y la capacidad de direccionamiento (para asignar las direcciones que especifican quiénes son los ordenadores fuentes y destino). Todo esto hace que el nivel de enlace se divida en dos subniveles: el subnivel de control de acceso al medio (MAC), situado por encima del nivel físico; y el subnivel de enlace lógico propiamente dicho (LLC), situado encima de MAC.

- Nivel de red:

Controla las operaciones de la red. Entre otras cosas, define las principales características del interfaz Host/Sistemas de tránsito, y cómo los paquetes, unidades de información a este nivel, son encaminados dentro de la red.

- Nivel de transporte:

Se conoce también como nivel Host/Host. La función básica del nivel de transporte consiste en aceptar los datos del nivel de sesión, partirlos en unidades más pequeñas si fuera necesario, pasarlos al nivel de red y asegurar que todas esas unidades de información lleguen correctamente al otro extremo. La conexión más corriente a este nivel de transporte es en circuito virtual que entrega los mensajes en el orden en que fueron recibidos.

- Nivel de sesión:

Es a través de este nivel con el que el usuario debe establecer la conexión con un proceso en otro sistema. Para establecer una sesión, el usuario debe proporcionar la dirección remota con la que se quiere conectar. En este nivel se convertirá la dirección de sesión en la dirección de transporte.

- Nivel de presentación:

Este nivel realiza varia funciones que se requieren para solucionar problemas de los usuarios. Algunas de estas funciones van orientadas a transformaciones criptográficas, compresión de texto, compatibilidad de terminales y transferencia de archivos.

- Nivel de aplicación:

Corresponde a este nivel las utilidades o servicios que se ofrecen a los usuarios. Normalmente se busca conseguir la transparencia de red, ocultación de recursos, etc…

La arquitectura de TCP/IP y su relación con el modelo OSI es la siguiente:

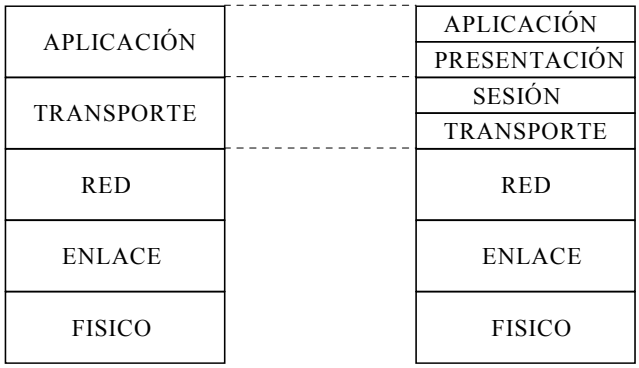


Figura 1.2: Relación entre la arquitectura de TCP/IP y el modelo OSI.

Servicios de TCP/IP:

Los servicios principales que se tienen en los distintos niveles de TCP/IP son los siguientes:

APLICACIÓN	FTP TELNET DNS RCP TFTP SMTP
TRANSPORTE	TCP UDP
RED	IP
ENLACE	ETHERNET ARP RARP
FISICO	ETHERNET TOKEN RING IEEE 802.5 FFDI

Figura 1.3: Servicios de TCP/IP.

- Nivel físico:

Es el nivel de hardware. Los protocolos a nivel físico transmiten datos en forma de paquetes. En este nivel, TCP/IP soporta varios protocolos entre los que se incluye Ethernet, IEEE 802.5, FDDI, Token Ring, etc.

- Nivel de enlace:

El controlador Ethernet es un ejemplo de estándar soportado por este nivel. Existen dos protocolos, ARP y RARP, que se sitúan entre el nivel de red y el nivel de enlace. ARP es el protocolo de resolución de direcciones Ethernet (Ethernet Address Resolution Protocol). Se encarga de asignar direcciones IP (32 bits) a direcciones Ethernet (48 bits). Esta dirección se convierte porque para poder mandar la información necesitamos la dirección física. RARP o Reverse ARP es el protocolo de resolución de direcciones IP. Asigna direcciones Ethernet a direcciones IP. Se utiliza en estaciones sin disco. ARP y RARP no se usan si no se utiliza Ethernet

- Nivel de red:

IP es el protocolo presente en el nivel de red. IP es un servicio de datagrama sin conexión. Cada datagrama es direccionado independientemente a su dirección IP destino, aunque se tiene en cuenta direccionamiento y enrutamiento. IP no puede asegurar la llegada de un datagrama al destino final. Si hay problemas en la red, como rutas incorrectas, un gateway debe anunciar el error al host original. Los errores se generan porque cada máquina trabaja independientemente de las otras (un ordenador puede estar desconectado, puede tener una ruta mal configurada, etc...). El protocolo ICMP permite que un host o gateway mande informes de errores o mensajes de control a otros gateways o hosts. Los mensajes de ICMP lo usa el software de Internet y no los procesos de usuario. Algunos errores son por fallos de enrutamiento, vencimiento de tiempo de vida del datagrama, información de rutas mejores, excesivas pérdidas de paquetes, etc.

- Nivel de transporte:

Los protocolos a nivel de transporte establecen comunicaciones entre procesos ejecutándose en máquinas separadas. Los protocolos a este nivel son TCP y UDP. TCP (Transmision Control Protocol) es un producto orientado a la conexión (necesita dos puntos finales para comunicarse con significado). Es el responsable de proveer a los usuarios de circuitos virtuales para la comunicación. UDP proporciona un servicio de datagrama sin conexión. No garantiza el orden de entrega.

- Nivel de aplicación:

En este nivel existe una gran variedad de protocolos TCP/IP. Básicamente todos estos servicios se pueden descomponer en cuatro grandes grupos:

  - Servicios de conexión remota; como telnet, rlogin y remsh.
  - Servicios de transferencia/compartición de archivos; como rcp, ftp, bftp, etc.
  - Servicios de correo electrónico; como smtp.
  - Servicio de nombre de dominios (DNS).

DIRECCIONES INTERNET:

Para que los hosts puedan comunicarse entre sí es necesaria la siguiente información:

- Nombre del host.
- Dirección del host.
- Ruta para poder llegar hasta donde se encuentra el host.

El esquema de direccionamiento de TCP/IP fue diseñado con la idea de poder procesar la información con eficiencia, ya que cada máquina por la que pasa un paquete tiene que tomar la decisión de mandarlo a otra máquina, quedárselo y pasarlo a la aplicación o descartarlo.

Los hosts en una red TCP/IP se identifican por tres direcciones:

- Hostname: Nombre que identifica a un host.
- Dirección Internet: Dirección de red que es entendida por los protocolos inferiores.
- Dirección física: Dirección de las tarjetas de red que se utiliza en conexiones Ethernet. No puede ser modificado.

Cada host en una red TCP/IP está identificado, generalmente, por una única dirección Internet. Esta dirección es independiente de la dirección física.

Cada dirección Internet, también conocida como dirección IP, está formada por 32 bits. La dirección se divide en dos partes: una corresponde a la red y otra al host. El organismo encargado de gestionar las direcciones, NIC (Network Information Center), ha establecido cinco clases de direcciones; aunque sólo tres de ellas son las básicas (A, B, C).

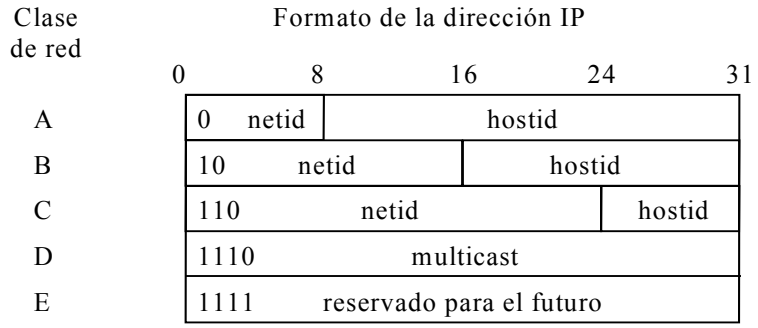


Figura 1.4: Formato de las cinco clases de direcciones IP.

Dicho de otra forma, dividiendo la dirección Internet en bytes, los valores posibles del primer byte para la clase A estarán comprendidos entre 1-126, para la clase B entre 128-191 y para la clase C entre 192-223. Los tres primeros bits de una dirección determinan su clase. Estas máscaras de bits lo usan los enrutadores para determinar que bits de una dirección Internet pertenecen a la dirección de red.

Los ordenadores que están conectados a más de una red tendrán tantas direcciones IP como número de conexiones de red tengan. Además, existen direcciones especiales:

- 0 en el campo netid significa “esta red”.
- Todos 1 en el campo hostid significa broadcast, es decir, mensaje para todos.

Además hay esquemas de direccionamiento para multicasting (valor 224), es decir, mensajes dirigidos a un grupo de ordenadores.

La ventaja de esta forma de direccionamiento es que es un esquema muy eficiente; el ordenador tiene que hacer pocos cálculos para saber qué es lo que ha de hacer con el paquete. Los inconvenientes son: si se cambia el ordenador de un sitio a otro, hay que cambiar la dirección IP, cambiar un conjunto de redes de una clase a otra implica realizar muchos cambios y el enrutamiento no es claro ya que pueden existir ordenadores con más de una dirección IP.

Normalmente las direcciones IP serán de tipo decimal, donde cada byte es separado por un punto decimal; por ejemplo, 128.10.2.30. En la actualidad se usan nombres en lugar de números y luego se traducen a direcciones IP.

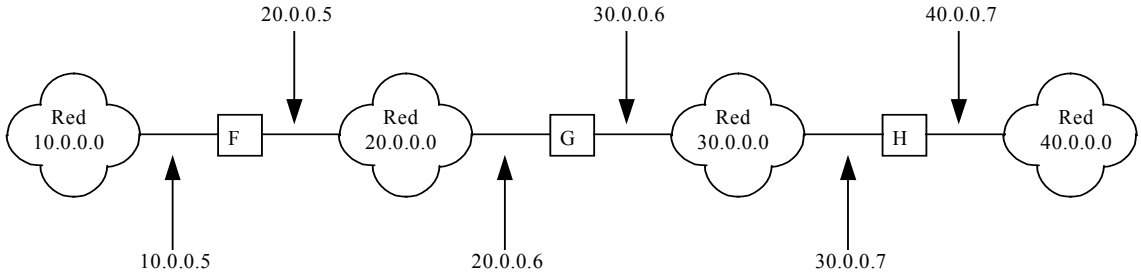
Dentro de una red, los hosts pueden estar agrupados en redes físicas más pequeñas llamados subredes. Para ello se utiliza la máscara de subred (subnet mask). Esto nos permite cambiar redes de clase A a redes de clase B o C o bien, redes de clase B a redes de clase C. Por ejemplo, si una empresa tiene dos redes físicas Ethernet compartiendo una dirección de clase A, 91.0.0.0, los hosts de una de las redes tendrán las direcciones 91.1.X.X y los otros hosts, de la otra red, las direcciones 91.2.X.X. Este particionamiento de la red no es visible para los hosts que no estén en la red subdividida (sólo ven una única red).

ENRUTAMIENTO:

El enrutamiento (routing) se define como la elección de un camino desde un origen a un destino. Una red puede estar formada por varias redes físicas interconectadas por equipos denominados gateways (pórticos).

Existen dos tipos de rutas: rutas directas y rutas indirectas. Se utiliza una ruta directa cuando se quieren comunicar dos hosts en una misma red; la cual se realiza sin la necesidad de usar gateway. En este caso, el enrutamiento no se necesita ya que sólo se debe especificar la dirección de destino.

Cuando el host destino no está en la misma red, el host emisor debe identificar un gateway al cual enviar los datos. El gateway enviará a su vez los datos hacia la red destino. En este caso, se estará utilizando una ruta indirecta. Este enrutamiento se realiza mediante tablas. En cada máquina se guarda una tabla de parejas de direcciones (red_destino, gateway) donde la dirección indicada por red_destino identificará a la red destino a la cual va dirigida la información y la otra dirección, indicada por gateway, indica el siguiente gateway al cual se hace la entrega. Si no se requiere otro gateway intermedio, se pondrá una marca especial para indicar la entrega directa.



Entrega a hosts de la red	Enrutar a dirección
20.0.0.0	entrega directa
30.0.0.0	entrega directa
10.0.0.0	20.0.0.5
40.0.0.0	30.0.0.7

Gateway G

Figura 1.5: Ejemplo de enrutamiento.

Nótese que un gateway sólo conoce la dirección del gateway siguiente y no el enrutamiento completo, y que la siguiente estación del datagrama se encuentra en la misma red a la que pertenece el gateway. Mirando la figura 1.5, las direcciones inferiores se corresponden con la comunicación a la derecha y las superiores, con la izquierda.

El enrutamiento puede realizarse a hosts específicos o a redes específicas. Para ello de utiliza la orden *route*. Si se quiere guardar la información del enrutamiento se debe actualizar el archivo /etc/gateways.eth para cada ruta que se quiera añadir.

BLOQUE 5: PROGRAMACIÓN CON LA INTERFAZ DEL SISTEMA

TEMA 1. EL COMPILADOR DE C.

COMPILACIÓN Y EJECUCIÓN DE PROGRAMAS EN C:

Los pasos a seguir para compilar un programa en C son:

1. Con un editor, escribir y guardar el código fuente del programa en C. Utilizar un nombre con extensión .c (Por ejemplo prueba.c).
2. Para compilarlo se utiliza el comando *cc*. Si no hay errores, el resultado de la compilación se guardará en un archivo que por defecto se llama *a.out*.
3. Para ejecutarlo basta con escribir el nombre del archivo:

```
% a.out
```

EL COMPILADOR cc:

Sintaxis:

cc [opciones] <archivo> … [librerías]

cc traduce programas escritos en C a módulos ejecutables. Opcionalmente los enlaza con otros archivos objeto generados por *cc* o por otros procesadores de lenguajes. Admite una serie de archivos fuente en C y de archivos objeto en la lista de archivos <archivo>. *cc* permite que se compilen y enlacen cualquier combinación de archivos de los siguientes tipos:

- Archivos fuente en C (extensión .c).
- Archivos fuente preprocesados en C (extensión .y). Son archivos fuentes sin comentarios.
- Archivos con código objeto (extensión .o).

- Archivos fuente en ensamblador (extensión .s).

Después de haber realizado satisfactoriamente el enlace de los diferentes archivos, el resultado que da en un archivo (*a.out*), salvo que se indique otro archivo mediante la opción -o en la línea de compilación.

Algunas opciones de compilación:

-c	Indica que se suprima la llamada a *ld* (enlazador) y produce un archivo objeto .o por cada archivo fuente. Puede especificar el nombre del archivo objeto con la opción -o nombre.
-C	Llama al preprocesador de C, *cpp*, para borrar los comentarios.
-fsingle	Indica a *cc* que utilice sólo aritmética en simple precisión y no convierta
nada	a doble precisión y como lo hace por defecto.
-g	Produce tablas de información adicional para *dbx* y *dbxtool*.
-go	Produce tablas de información adicional para *adb*.
-help	Visualiza información acerca de *cc*
-I camino	Proporciona el camino del directorio donde se ha de buscar los archivos #include. El compilador busca estos archivos primero en el
directorio donde	esta el fuente, luego los indicados con -I y por último en
/usr/include.	
-l	Indica a *ld* el enlace con las librerías objeto. El orden de las librerías en la línea de compilación es importante, son resueltas de izquierda a derecha. Esta opción ha de seguir a los argumentos del archivo fuente.
-L directorio	Añade directorio a la lista de directorios en donde se han de buscar rutinas de librerías objeto para ser enlazadas con *ld*.
-o archivo	Nombre del archivo de salida.
-w	No imprime mensajes de aviso.

TEMA 2. HERRAMIENTAS PARA LA DEPURACIÓN.

INTRODUCCIÓN:

Existen tres depuradores:

- *dbx*
- *dbxtool*
- *adb*

***dbx*:**

Depurador interactivo, orientado a líneas, a nivel de programas fuentes. Permite determinar cuando un programa se aborta, visualiza el valor de las variables y expresiones, incluye puntos de ruptura en el código y ejecuta un programa paso a paso ofreciendo información sobre esta ejecución.

***dbxtool*:**

Depurador basado en una interface de ventanas para *dbx*. Similar a *dbx* salvo que es más cómodo de manejar porque permite usar el ratón. Incluye una ventana para ejecutar cualquiera de los comandos existentes en *dbx*.

***adb*:**

Depurador interactivo orientado a líneas; depura a nivel de ensamblador. Se usa para examinar los archivos *core* (que contienen una copia del estado del programa en momento de romperse) y determinar así las causas de la ruptura. Ofrece un entorno controlado para la ejecución del programa.

dbx y *dbxtool* son mucho más fáciles de utilizar que *adb* y además, contienen todos los aspectos necesarios para la depuración. *adb* es más útil para la observación interactiva de archivos binarios sin símbolos ni código objeto. Se utiliza para depurar código objeto cuando no disponemos del fuente y para depurar el kernel.

TIPOS DE DEPURACIÓN SOPORTADOS POR *dbx* Y *dbxtool*:

Soportan tres tipos de depuración:

- Post-mortem.
- Proceso vivo.
- Multiproceso.

Post-mortem:

Se pueden depurar un programa en post-mortem si éste ha generado un archivo *core*, *dbx* toma el valor de las variables de dicho archivo. La operación más corriente es conseguir la pila de ejecución con el comando *where* y ver el valor de las variables con *print*.

Proceso vivo:

En un proceso vivo el depurador puede:

- Establecer un punto de arranque del proceso.
- Fijar y borrar puntos de ruptura.
- Reinicializar un proceso parado.

Multiproceso:

La depuración multiproceso es útil cuando depuramos dos procesos que están estrechamente ligados. Por ejemplo en una red puede haber un programa cliente y otro servidor (que intercambian información, llamadas a procedimientos, etc.). Para depurar ambos es necesario que cada uno haya iniciado la depuración con *dbx*. No se permiten depuraciones remotas sino que éstas deben hacerse sobre la misma máquina.

dbx:

Estudiando primero el funcionamiento de la herramienta *dbx*, será más fácil comprender luego el de *dbxtool*. Ambos tienen las mismas órdenes pero *dbxtool* incorpora un interface gráfico (ventanas selección con ratón, etc.). Tenemos que trabajar con *dbx* desde el shell mientras que para *dbxtool* necesitamos el entorno X-Windows. Así, para terminales que no tengan posibilidades gráficas, la única opción es *dbx*.

Preparación de un archivo para la depuración:

Para que un archivo sea depurable, el proceso de compilación debe realizarse incluyendo el argumento -g en la línea de llamada al compilador *cc*. Este parámetro indica al compilador que incluya cierta información en el archivo objeto, que será imprescindible para la depuración.

Llamada al depurador *dbx*:

La sintaxis de la llamada al depurador es la siguiente:

```
dbx [-r] [-kbd] [-I dir] [f_objeto] [f_core | Id_proces]
```

Las opciones son:

-r Ejecuta el archivo objeto al entrar.
-I dir Añade el directorio a la lista donde se pueden buscar el archivo fuente.
f_objeto Nombre del archivo que contiene el código objeto resultante de la compilación con la opción -g.
f_core | Id_proces
Si se incluye el nombre de un archivo core o del identificador de proceso, el depurador utiliza esta información para descubrir el estado del proceso cuando se produjo el fallo.

El prompt (dbx) nos indica que espera algún comando *dbx*. Para salir del depurador basta con indicar:

```
(dbx) quit
```

Operaciones en dbx:

Listado de código fuente:

```
(dbx) list linea_inicial,linea_final
```

Listado de procedimiento activo (o post-mortem):

Se mueve dentro de la pila y nos dice el procedimiento indicado con n.

```
(dbx) where [n]
(dbx) up [n]
(dbx) down [n]
```

Nombre y visualización de datos:

```
(dbx) print expresion
```
Imprime el valor de la expresión.

```
(dbx) display expresion
```
Visualiza el valor de la expresión cada vez que se detiene la ejecución del programa.

```
(dbx) undisplay expresion
```
Detiene la visualización de la expresión especificada.

```
(dbx) whatis identificador
      whatis tipo
```
Imprime el tipo del identificador o la estructura del tipo indicado.

```
(dbx) which identificador
```

Imprime el bloque de salida asociado al identificador.

(dbx) whereis identificador
Imprime un informe completo sobre las posiciones del identificador en el archivo.

(dbx) assign variable=expresion
set variable=expresion
Asigna valores a las variables.

(dbx) dump [funcion]
Visualiza el nombre y valor de todas las variables y parámetros en la función.

Establecer puntos de ruptura:

(dbx) stop at n_linea [if condicion]
Se detiene en la línea si la condición es correcta o siempre si no hay condición.

(dbx) stop in procedimiento [if condicion]
Se detiene en la primera línea del procedimiento si la condición es verdadera o siempre si no la hay.

(dbx) stop variable [if condicion]
Detiene la ejecución cuando cambia el valor de la variable si la condición es cierta.

(dbx) stop if condicion
Se detiene cuando la condición sea cierta.

(dbx) when in procedimiento {comando;…}
Ejecuta el comando *dbx* cuando entra al procedimiento o función indicada.

(dbx) when at n_linea {comando;…}
Ejecuta el comando al llegar a la línea indicada.

(dbx) when condicion {comando;…}
Ejecuta el comando cuando la condición es cierta.

Para ver y quitar los puntos de ruptura:

(dbx) status [>nombre_archivo]
Visualiza los comandos *trace*, *stop* y *when* activados y da un número de comando a cada uno.

(dbx) delete numero_comando [[,] numero_comando]
delete all
Borra el comando indicado o todos ellos.

(dbx) clear n_linea
Borra los puntos de ruptura de esta línea.

Ejecución de programas:

(dbx) run [argumentos]
Inicia la ejecución del archivo objeto.

(dbx) rerun
Igual que el anterior salvo que utiliza los argumentos de la última ejecución.

(dbx) cont [at n_linea]
Continúa la ejecución desde donde estaba parado o desde la línea indicada.

(dbx) trace n_linea
Visualiza la línea justo antes de ejecutarla.

(dbx) trace procedimiento
Cada vez que se llame al procedimiento visualiza información sobre qué rutina se llama y el valor de los parámetros y cuando vuelve visualiza el valor de salida de la función.

(dbx) trace expresion
Visualiza el valor de la expresión cuando la línea de código es alcanzada.

(dbx) trace variable
Visualiza el nombre y valor de la variable cuando cambia.

(dbx) step [n]
Ejecuta las siguientes n líneas y luego para. Entra en los procedimientos y funciones.

(dbx) next [n]
Ejecuta las siguientes n líneas. Ejecuta los procedimientos y funciones en un paso.

(dbx) call procedimiento parametros
Ejecuta el procedimiento.

Acceso a archivos y directorios:

Permite acceder a archivos y directorios sin salir del *dbx*.

(dbx) edit nombre_archivo
edit procedimiento
Edita con *vi*.

(dbx) file nombre_archivo
Cambia el archivo actual.

(dbx) func procedimiento
Cambia la función actual.

(dbx) use [dir]
Incluye el directorio en la lista de directorios donde se buscará un archivo fuente.

(dbx) cd [dir]

(dbx) pwd

(dbx) sh comando
Ejecuta un comando del shell.

(dbx) /cadena [/]
Busca cadena desde la posición actual hacia el final del archivo.

(dbx) ?cadena [?]
Busca la cadena desde la posición actual hacia el comienzo del archivo.

PRÁCTICA:

Para practicar con el compilador de C (*cc*) y el depurador (*dbx*), construya un programa que lea un archivo de texto que se le pase como parámetro y que cuente el número de caracteres, de palabras y de líneas del mismo.

TEMA 3. INTRODUCCIÓN A LOS SERVICIOS DEL SISTEMA.

INTRODUCCIÓN:

La única forma que existe en el sistema operativo UNIX para acceder a los servicios del kernel es mediante las llamadas al sistema. Algunos de los servicios que se pueden obtener son los correspondientes a los sistemas de archivos, a los mecanismos d multitarea y a las primitivas de comunicación entre procesos.

En definitiva, las llamadas al sistema definen la interfaz entre el kernel y los programas de usuario que se ejecutan en el nivel superior; o dicho de otra forma, definen lo que es UNIX.

Otros servicios de más alto nivel que podemos utilizar en el entorno de programación son aquellas que nos ofrece las rutinas de las librerías del compilador.

Veamos algunos conceptos básicos que pueden ser necesarios cuando se programe en UNIX:

- Descriptor: Es un entero asignado por el sistema que nos permite referenciar unívocamente a un archivo, buzón (socket), grupo de semáforos, etc. para poder realizar operaciones con éstos.
- Grupo de procesos tty (Tty Process Group): Cada proceso activo puede ser miembro de un grupo de terminal que se identifica por un entero positivo llamado ID de grupo de procesos tty. Esta agrupación se usa para arbitrar entre múltiples trabajos que se encuentren en el mismo terminal y para dirigir señales al grupo de procesos apropiado.
- ID de usuario efectivo, ID de grupo efectivo y Grupos de acceso: El acceso a los recursos del sistema se controla mediante tres valores, el ID de usuario efectivo, el ID de grupo efectivo y los IDs de grupo suplementarios. El ID de usuario efectivo e ID de grupo efectivo son normalmente el ID de usuario real e ID de grupo real del proceso respectivamente. Pero ambos pueden ser modificados mediante el bit setuid o el bit setgid del proceso ejecutado.
- ID del proceso padre (Parent Process ID): Un nuevo proceso se crea a partir de un proceso *fork* activo. El ID del proceso padre de un proceso es el ID de proceso de su creador.
- ID de grupo de procesos (Process Group ID): Cada proceso activo es miembro de un grupo de procesos que se identifica por un entero positivo llamado ID de grupo de procesos. Esta agrupación permite la señalización de procesos relacionados y los mecanismos de control de trabajos.
- ID de usuario real e ID de grupo real: Cada usuario en el sistema se identifica por un entero positivo conocido como el ID de usuario real. Cada usuario pertenece al menos a un grupo, conocido como ID de grupo real.
- Procesos especiales (Special processes): Los procesos con ID de proceso 0, 1 y 2 son especiales. El proceso 0 es el scheduler. El proceso 1 es el proceso de inicialización *init*, y por tanto el ascendiente de cualquier otro proceso del sistema. Se usa para el control de la estructura de procesos. El proceso 2 es el proceso de swapping.
- Señal (Signal): Las señales se usan para notificar sucesos asíncronos. Las señales se pueden dirigir a procesos, grupos de procesos y a otras combinaciones de procesos. Las señales las pueden enviar un proceso o el propio sistema operativo.

- Sesión (Session): Cada proceso es miembro de una sesión. Con cada terminal de control se asocia una sesión en el sistema, como ocurre con los shells de conexión.
- Terminal de control (Controlling Terminal): Un terminal que se asocia con una sesión. Cada sesión puede tener al menos un terminal de control. Un terminal puede ser el terminal de control de al menos una sesión. El terminal de control se usa para dirigir señales (como interrupciones o señales de control de trabajos) a los procesos apropiados.

Estudiemos como ejemplo las llamadas al sistema que se emplean en la creación y ejecución de procesos, en el manejo de semáforos, en la compartición de memoria y en la estructura básica de un programa cliente/servidor.

TEMA 4. CREACIÓN Y EJECUCIÓN DE PROCESOS.

CONCEPTOS BÁSICOS:

Un proceso es un programa en ejecución que puede ejecutarse concurrentemente con otros procesos. El sistema operativo UNIX permite a los usuarios crear varios procesos a la vez, los cuales pueden sincronizarse, comunicarse e incluso compartir información.

La única forma que tiene el sistema para identificar a un proceso es mediante un identificador de proceso (PID), el cual es único en el sistema. Asociado con cada proceso se encuentra el contexto de un proceso.

El contexto de un proceso está formado por el contenido de su espacio de direccionamiento, el contenido de los registros hardware y las estructuras de datos del kernel asociado con el proceso. Más formalmente, el contexto de un proceso es la unión del contexto a nivel de usuario (user-level context), del contexto del registro (register context) y del contexto a nivel del sistema (system-level context). El contexto a nivel de usuario consta de código, datos y pila así como de la memoria compartida que pueda tener. El contexto del registro consta del contador de programa, registro de estado del procesador, punteros de la pila y registros de propósito general. El contexto a nivel del sistema consta de la entrada en la tabla de procesos, la u-area (formada por los archivos abiertos, directorio actual, …), la pila del kernel, …

Algunas de las llamadas al sistema que nos permite manejar a los procesos son:

fork	crea un nuevo proceso.
exec	permite a un proceso ejecutar un nuevo programa.
wait	permite a un proceso padre (parent pocress) sincronizar su ejecución con la llamada *exit* de un proceso hijo (child process).
exit	termina la ejecución de un proceso.

CREACIÓN DE PROCESOS:

La única forma que existe para que un usuario pueda crear un nuevo proceso en UNIX es ejecutando la llamada al sistema *fork*. Esta llamada hace que el proceso que la ejecute se divida en dos procesos. Al proceso que ejecuta *fork* se le conoce como proceso padre y al nuevo proceso creado se le llama proceso hijo. La sintaxis de *fork* es:

```
int fork ();
```

Tras ejecutarse esta llamada al sistema, los dos procesos tendrán copias idénticas de su contexto a nivel de usuario. La única diferencia será que el valor entero que devuelve *fork* (correspondiente al PID) para el proceso padre el ID del proceso hijo mientras que para el proceso hijo es 0. Estos valores se utilizan para determinar el código que ejecutará cada proceso (ver ejemplo más adelante). En caso de error, devuelve el valor -1. Además, ambos procesos compartirán los archivos que el proceso padre tenía abiertos hasta el momento de la llamada al sistema. Y por último, el proceso hijo recién creado heredará los IDs de usuario (real y efectivo)

del proceso padre, el grupo de proceso padre, el directorio actual y el valor "nice" del padre; el cual de usa para calcular la prioridad de planificación.

Todos los procesos en el sistema, excepto el proceso 0, se crea mediante esta llamada al sistema[12].

El kernel asocia dos IDs de usuario con cada proceso (heredados por el proceso padre, como se ha dicho anteriormente), independiente del ID del proceso; el ID de usuario real (RUID) y el ID de usuario efectivo (EUID) o setuid (Set User ID). El ID de usuario real identifica al usuario responsable del proceso que se va a ejecutar mientras que el usuario efectivo se obtiene cuando se hereda el bit setuid del modo de permisos de un archivo y consiste en que un usuario (sólo durante la ejecución del programa) tendrá los privilegios del propietario de un archivo ejecutable, como si el usuario que está ejecutando el programa fuera el propietario del archivo. Con esto conseguiremos acceder a archivos de datos que pueda manejar el programa y para los cuales no se tenía permiso. Un ejemplo se tiene con la orden *passwd*, el cual accede a un archivo de datos (/etc/passwd) que no podría ser modificado directamente por un usuario normal.

El sistema impone (aunque es configurable) un límite en el número de procesos que un usuario puede ejecutar simultáneamente. De esta forma evitamos que se sature la tabla de procesos ya que podría impedir que otros usuarios pudieran crear procesos. Estas limitaciones no afectan al superusuario.

Considere el siguiente ejemplo. Consiste en un método para compartir el acceso a archivos mediante la llamada al sistema *fork*. Así, ambos procesos (padre e hijo) leerán la información del archivo existente y luego lo escribirán en un archivo nuevo. Para ejecutarlo, un usuario deberá escribir el nombre del programa con dos parámetros: (1) el nombre de un archivo existente y (2) el nombre del nuevo archivo a crear. Cada proceso podrá acceder a su copia privada de las variables pero ninguno podrá acceder a las variables del otro proceso.

```
/*      hijpad.c      */

#include <fcntl.h>

int fdrd,fdwt;
char c;

main (argc,argv)
   int argc;
   char *argv[];
{
int status;
if (argc!=3)
        {
        printf ("\nFaltan parametros.\n");
        exit (1);
        }
if ((fdrd=open(argv[1],O_RDONLY))==-1)
        {
        printf ("\nNo se puede leer el archivo de entrada.\n");
      exit(1);
        }
if ((fdwt=creat(argv[2],0666))==-1)
        {
```

[12] En algunas versiones modernas de UNIX, desaparece el proceso 0 y así, todos los procesos descienden del proceso *init* (proceso 1).

```
        printf ("\nNo se puede crear el archivo de salida.\n");
        exit(1);
        }
printf ("Status=%d",status);

switch (fork()) {
        case -1:  /* Gestión del error */
                  exit (1);
                  break;
        case 0:   /* Proceso hijo */
                  rdwrt();
                  exit (0);
                  break;
        default: /* Proceso padre */
                  rdwrt();
                  wait (&status);      /* espera a que termine el proceso hijo */
                  break;
  }  /* fin del switch */
}  /* fin de main */

rdwrt ()
{
for (;;) {
        if (read(fdrd, &c,1)==0=
                return;
        write (fdwt, &c, 1);
  }
}  /* fin de rdwrt */
```

Note que ambos procesos ejecutan el mismo código para poder así compartir el acceso al archivo.

EJECUCIÓN DE PROCESOS:

La llamada al sistema *execve* ejecuta un programa, colocándose el código de éste en el espacio de memoria del proceso que ejecutó la llamada. Su sintaxis es la siguiente:

```
execve (filename, argv, envp);
  char *filename, *argv[], *env[];
```

donde filename es el nombre del archivo a ejecutar, argv es un puntero a una lista de cadenas que corresponderán con los parámetros del programa ejecutable y envp es un puntero a una lista de punteros a cadenas que corresponden con el entorno del programa a ejecutar. Las cadenas se caracteres en el entorno son de la forma "nombre=valor" y pueden contener información útil para los programas como pueden ser el directorio de trabajo del usuario, caminos de directorios para buscar programas ejecutables, etc. Los procesos pueden también acceder a su entorno a través de la variable global environ (su declaración es de la forma siguiente: extern char **environ) inicializada por las rutinas del C.

Hay varias funciones de las librerías del lenguaje C que permiten también ejecutar programas como son *execl*, *execc*, *execlp* y *execvp*. Se diferencian en el tipo de información que se

pasa como parámetros. Estas funciones, junto con la llamada al sistema *execve*, forman la familia *exec*. Muchas de estas funciones son más sencillas de usar que la llamada la sistema; como podemos apreciar en el próximo ejemplo.

En el ejemplo siguiente, el programa crea un proceso hijo que ejecuta la función *execl*:

```
/*     crear.c      */

main()
{
int status;

if (fork()==0)
        execl ("/bin/date","date",0);
wait (&status);
}
```

ESPERA Y FINALIZACIÓN DE PROCESOS:

La llamada al sistema *wait* espera a que un proceso finalice. Normalmente lo ejecuta un proceso padre para esperar a que un proceso hijo termine su ejecución. Su sintaxis es:

```
wait (estado);
  int *estado;
```

Tras su ejecución, devuelve el PID del proceso hijo en cuestión y coloca en estado el estado de salida proporcionado por éste. Devuelve -1 si no existe ningún proceso hijo del proceso que envía la llamada. Podemos decir que *wait()* se utiliza para la sincronización de los dos procesos (padre e hijo) que se están ejecutando concurrentemente.

La llamada al sistema *exit()* provoca la finalización de un proceso. Los 8 bits menos significativos del argumento quedan a disposición del proceso padre.

Un proceso puede finalizar bien cuando se ejecuta *exit* o bien cuando termina la ejecución de su código.

PRÁCTICA:

Construya un pequeño shell ayudándose de las llamadas al sistema que se acaban de ver. El programa debe mostrar un prompt y ejecutará los comandos que se le especifiquen. El shell terminará su ejecución cuando se le introduzca una cadena determinada.

TEMA 5. SEMÁFOROS.

CONCEPTOS BÁSICOS:

Las llamadas al sistema de semáforos permite a los procesos sincronizar la ejecución haciendo un conjunto de operaciones atómicamente sobre un conjunto de semáforos. Gracias a esta sincronización, los procesos pueden cooperar correcta y eficientemente sobre recursos no compartibles que estén usando conjuntamente.

Dijkstra publicó el algoritmo de Dekker que describe una implementación de los semáforos según la cuál, eran objetos con valores enteros que tienen dos operaciones atómicas: P y V[13]. La operación P decrementa el valor de un semáforo si su valor es mayor de 0; la operación V incrementa su valor. Ya que las operaciones son atómicas, no se puede realizar más de una operación P o V a la vez sobre un semáforo. Las llamadas al sistema de semáforos en UNIX son una generalización de las operaciones P y V de Dijkstra, en el que varias operaciones se pueden hacer simultáneamente y las operaciones de incremento y decremento pueden ser con valores mayores de 1. El kernel hace todas las operaciones atómicamente; ningún otro proceso puede ajustar los valores de los semáforos hasta que todas las operaciones son hechas.

Las llamadas al sistema de semáforos son:

semget	Crea y accede a un conjunto de semáforos.
semctl	Hace varias operaciones de control sobre el conjunto.
semop	Manipula los valores de los semáforos.

La llamada al sistema *semget* crea una lista de semáforos (también se habla de conjunto de semáforos) o devuelve un identificador (ID) a una lista si ésta ya existía. Su sintaxis es la siguiente:

```
int semget (key, nsem, semflg)
        key_t key;
        int nsem, semflg;
```

La forma de obtener un ID al conjunto de semáforos es mediante una clave o key (numero entero sin signo) que será igual para todos los procesos que trabajen con el mismo conjunto. En la figura 5.1 podemos apreciar como se enlazan todas las listas de semáforos existentes a una Tabla de semáforos (Semaphore Table). El ID que devuelve *semget* se utiliza en la Tabla de semáforos para localizar una determinada lista.

[13] En alguna bibliografía, a estas operaciones se les conoce como *wait* y *signal*, pero en el entorno UNIX existen dos llamadas al sistema con estos mismos nombres y no tienen el mismo significado; por lo que utilizaremos P y V.

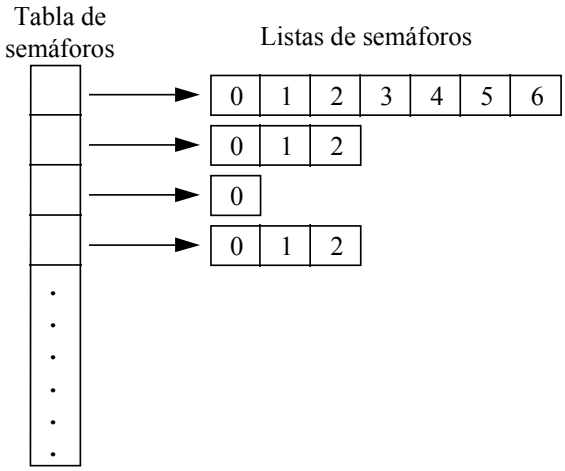


Figura 5.1: Relación entre las listas de semáforos y la tabla de semáforos.

Si el bit IPC_CREAT del parámetro *semflg* está activo, el conjunto se creará a no ser que éste existiera ya. Otros banderines que se pueden establecer son los permisos de la región (al igual que los archivos). Habrá *nsem* semáforos en el conjunto, numerados a partir de 0. El valor que devuelve (lo llamaremos *semid*) es de tipo entero y corresponde al identificador del conjunto de semáforos; en caso de error, devuelve -1.

Los procesos manipulan los semáforos con la llamada *semop*. Su sintaxis es la siguiente:

```
int semop (semid, semops, nsemops)
        int semid;
        struct sembuf *semops;
        int nsemops;
```

semop se usa para realizar atómicamente una lista de operaciones de semáforos en el conjunto de semáforos asociados con el identificador de conjunto de semáforos *semid*. *semops* es un puntero a la lista de estructuras de operación de semáforos. El contenido de cada estructura es el siguiente:

```
struct sembuf {
        unsigned short sem_num;   /* número de semáforo dentro de la lista */
        short          sem_op;    /* operación de semáforos */
        short          sem_flg;   /* banderines de operación */
};
```

nsemop es el tamaño de la lista. El valor de tipo entero que devuelve la función es el valor del último semáforo operado en el conjunto antes de hacer la operación.

El kernel lee la lista de operaciones de semáforos, *semops*, desde el espacio de direccionamiento del usuario y verifica que los números de semáforos son legales y que el proceso tiene los permisos necesarios para leer o cambiar los semáforos. En caso contrario, la llamada al sistema fallará.

Podría ocurrir situaciones peligrosas si un proceso realiza una operación sobre un semáforo, presumiblemente apoderándose o cerrando (locking) algún recurso, y después salir (*exit*) sin restablecer el valor del semáforo. Estas situaciones pueden ser debidas como consecuencia del resultado de un error del programador u otra circunstancia que cause la terminación de un proceso. Para evitar tales problemas, un proceso puede establecer el banderín SEM_UNDO en la llamada al

sistema *semop*; así, cuando éste sale, el kernel invierte[14] (reverse) el efecto de cada operación sobre el semáforo que el proceso haya hecho. El efecto neto es dejar el semáforo con el valor con que se inició.

La llamada al sistema *semctl* permite realizar operaciones de control sobre los semáforos. Su sintaxis es:

```
int semctl (semid, semnum, cmd, arg);
        int semid, semnum, cmd;
        union semun arg;
```

Las operaciones a realizar se especifican con *cmd* y afectan bien a un semáforo o bien a todos los semáforos que forman parte del conjunto de semáforos especificados con *semid*. En caso de que el comando vaya dirigido a un semáforo, éste estará identificado con *semnum*; y si va dirigido a todos, *semnum* especificará el número de semáforos sobre el que se va a actuar. *arg* se declara como una unión de la forma siguiente:

```
union semun {
        int val;
        struct semid_ds *semstat;
        unsighned short *array;
};
```

El kernel interpreta *arg* según el valor de *cmd*.
Algunas de las órdenes que se pueden especificar con *cmd* son:

GETVAL	Devuelve el valor del semáforo.
SETVAL	Establece el valor del semáforo a *arg.val*.
GETALL	Obtiene los valores de los semáforos y los deposita en la lista apuntada por *arg.array*.
SETALL	Establece los valores de los semáforos a partir de la información obtenida de la lista apuntada por *arg.array*.
IPC_RMID	Elimina del sistema el identificador de un conjunto de semáforos especificado por *semid* y destruye el conjunto de semáforos y estructuras de datos asociados con éste.

Veamos un ejemplo de cómo se pueden utilizar las llamadas al sistema de semáforos:

```
/*    semaforo.c    */

#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>

#define SEMKEY 76

main ()
{
```

[14] La forma de conseguir esta inversión es complicada de ver y se impone la claridad del estudio frente a la complejidad que supondría su explicación ya que no es imprescindible su conocimiento.

```
  /* Declaracion de variables */
  int semid;                          /* ID de la lista de semaforos */
  struct sembuf sem_oper;  /* Se usa para las operaciones P y V sobre
                                        semaforos */
  union semun arg;


  /* Creamos una lista de semaforos formada por dos semaforos */
  semid = semget (SEMKEY, 2, 0700 | IPC_CREAT);

  /* Inicializamos los semaforos */
  arg.array = (unsigned short *) malloc (sizeof (short)*2);
  arg.array [0] = arg.array [1] = 1;
  semctl (semid, 2, SETALL, arg);

  /* Veamos los valores iniciales de los semaforos */
  semctl (semid, 2, GETALL, arg);
  printf ("\nLos valores iniciales de los semáforos son %d y %d.\n",
          arg.array [0], arg.array [2];

  /* Operemos sobre los semáforos */
  sem_oper.sem_num = 0;     /* Actuamos sobre el semáforo 0 de la lista */
  sem_oper.sem_op = -1;       /* Vamos a decrementar en 1 el semaforo */
  sem_oper.sem_flg = SEM_UNDO;   /* Para evitar interbloqueos si un proceso acaba
                                    inesperadamente */
  semop (semid, &sem_oper, 1);

  sem_oper.sem_num = 1;     /* Actuamos sobre el semáforo 1 de la lista */
  sem_oper.sem_op = 1;       /* Vamos a incrementar en 1 el semaforo */
  sem_oper.sem_flg = SEM_UNDO;   /* No es necesario porque ya se ha hecho
                                        anteriormente */
  semop (semid, &sem_oper, 1);

  /* Veamos los valores de los semáforos */
  semctl (semid, 2, GETALL, arg);
  printf ("\nLos valores finales de los semaforos son %d y %d.\n",
          arg.array [0], arg.array [1]);

  /* Eliminar lista de semaforos */
  semctl (semid, 2, IPC_RMID, 0);

}  /* Fin de main */
```

PRÁCTICA:

Implemente el problema del productor/consumidor utilizando semáforos.

TEMA 6. MEMORIA COMPARTIDA.

CONCEPTOS BÁSICOS:

UNIX proporciona un sistema de memoria virtual. El espacio de direccionamineto de cada proceso está compuesto por una lista de páginas de memoria cada una de las cuales puede ser asignada y manipulada independientemente.

La memoria compartida (shared memory) es una forma implícita de comunicación entre procesos (IPC). Es por tanto un método altamente eficiente para compartir información entre procesos.

Los procesos pueden comunicarse directamente entre sí compartiendo partes de su espacio de direccionamiento virtual, con lo cual, se podrá leer y/o escribir datos en la memoria compartida. La forma de conseguir esto consiste en primer lugar en crear una región o segmento fuera del espacio de direcciones de un proceso y después, cada proceso que quiera acceder a dicha memoria, incluirá esa región como parte de su espacio de direccionamiento virtual.

El sistema permite que existan varias regiones compartibles, cada una con un subconjunto de procesos, y además, cada proceso puede acceder a varias regiones.

Para poder trabajar con memoria compartida se usan básicamente las siguientes llamadas al sistema:

shmget	Crea una nueva región de memoria compartida o devuelve una existente.
shmat	Une lógicamente una región al espacio de direccionamiento virtual de un proceso.
shmdt	Separa una región del espacio de direccionamiento virtual de un proceso.
shmctl	Manipula varios parámetros asociados con la memoria compartida.

Los procesos leen y escriben en la memoria compartida usando las mismas instrucciones que cuando utilizan la memoria ordinaria; una vez que dicha memoria compartida ha sido incluida como parte del espacio de direccionamiento del proceso.

Para la gestión de la memoria compartida, el kernel manipula tres tablas:

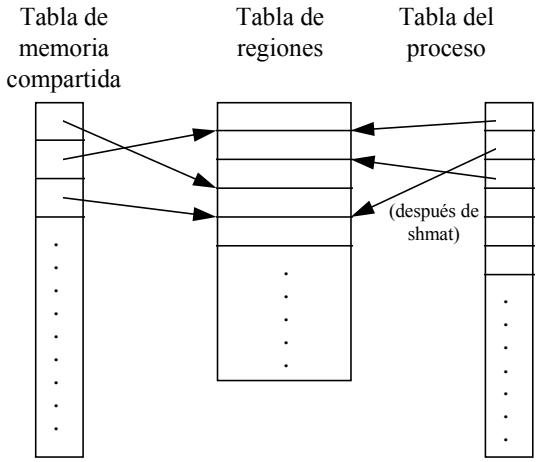


Figura 6.1: Tablas usadas por el kernel para gestionar la memoria compartida.

La tabla de memoria compartida (shared memory table) contiene una entrada por cada región distinta que se está compartiendo en el sistema; la tabla de regiones (region table) contiene una entrada por cada región y por último, la tabla del proceso o tabla de regiones por proceso (process table - per process region table) contiene las entradas correspondientes a las regiones que forman el espacio de direccionamiento del proceso. Existirá una tabla del proceso por cada proceso del sistema.

La sintaxis de *shmget* es la siguiente:

```
int shmget (key, size, flag);
        key_t key_;
        int size, shmflg;
```

donde *size* es el número de bytes de la región. El kernel busca en la tabla de memoria compartida por la clave (*key*) dada. La clave será un valor entero único para todos los procesos que comparten la misma región de memoria. Si existe una entrada, y los permisos lo permiten, devuelve un identificador a la región (en adelante *shmid*). Si no lo encuentra y el usuario había establecido el banderín (flag) IPC_CREAT, el kernel creará una nueva región. Otros banderines que se pueden establecer son los permisos a la región.

En caso de que se produzca un error, se devuelve -1. La región de memoria quedará reservada y sólo se asignará al espacio de direcciones de los procesos (tablas de páginas) cuando éstos se unan a la región, mediante la correspondiente llamada a *shmat*. La sintaxis de esta llamada al sistema es la siguiente:

```
char *shmat (shmid, addr, flags);
        int shmid;
        char *addr;
        int flags;
```

donde *shmid*, devuelto por la función *shmget*, identifica a la región de memoria compartida; *addr* es la dirección virtual donde el usuario quiere unir la memoria compartida; y *flag* especifica si la región es de sólo lectura y si el kernel debería completar la dirección especificada por el usuario. El valor que devuelve (*virtaddr*) es la dirección virtual donde el kernel a unido la región, no necesariamente el valor solicitado por el proceso. En caso de error, se devuelve -1.

Cuando se ejecuta *shmat*, el kernel verifica que el proceso tiene los permisos necesarios para acceder a la región. Si la dirección especificada por el proceso es 0, el kernel elige una dirección virtual conveniente.

Es necesario tener en cuenta que los procesos pueden incrementar el tamaño de su región de datos y pila. Cuando se hace, se asignan regiones nuevas contiguas virtualmente con las existentes; por ello, no se debe de utilizar *shmat* con direcciones que pudieran impedir esto.

Un proceso separa una región de memoria compartida desde su espacio de direcciones virtuales mediante:

```
int shmdt (addr)
        char *addr;
```

donde *addr* es la dirección virtual devuelta por *shmat*. Aunque parezca más lógico pasar a esta llamada un identificador, se usa una dirección para poder identificar entre varias instancias de la región de memoria compartida que estén unidas a sus espacios de direcciones.

Un proceso usa la llamada al sistema *shmctl* para buscar el estado y establecer parámetros de la región de memoria compartida. Su sintaxis es:

```
shmctl (shmid, cmd, shmststbuf)
        int shmid, cmd;
        struct shmid_ds *shmstatbuf;
```

donde *shmid* identifica a la entrada correspondiente en la tabla de memoria compartida, *cmd* especifica el tipo de operación y *shmststbuf* es la dirección de una estructura de datos a nivel de usuarios que contiene información de estado de la entrada de la tabla de memoria compartida cuando buscamos o establecemos su estado. El kernel utiliza los comandos para buscar el estado y cambiar al propietario y los permisos asociados con la región. Cuando se elimina una región de memoria compartida, el kernel libera la entrada y mira en la tabla de región; si no hay procesos que tengan la región unida a su espacio de direcciones, el kernel la liberará; en caso contrario, los procesos podrán seguir usándola pero ningún otro proceso podrá ser unido a la región.

Algunas de las operaciones que se pueden especificar con *cmd* son:

IPC_STAT Sitúa el valor actual de cada elemento de la estructura de datos asociada con *shmid* en la estructura apuntada por *shmststbuf*.

IPC_SET Establece el valor de los siguientes elementos de la estructura de datos asociada con *shmid*, obtenidos a partir de la estructura de datos apuntada por *shmstatbuf*.

IPC_RMID Elimina del sistema el ID de memoria compartida especificada con *shmid*.

Veamos un ejemplo de cómo se puede compartir una región de memoria.

```
/*                      memoria.c
  Descripcion: Sincronizacion entre dos o mas procesos; es decir,
               este proceso no avanza hasta que otro proceso no
               ponga un valor distinto de cero en una posicion de
               memoria compartida
*/

#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/sem.h>

#define SHMKEY 70
#define SEMKEY 60
#define PRINC 0

int semid,shmid;
char *addr;
int *pint;
struct sembuf sem_oper;
union semun arg;
struct shmid_ds *shmstatbuf;

main ()
{
```

```
 semid = semget (SEMKEY, 1, 0777);

 /* Obtenemos la direccion de la region de memoria */
 shmid = shmget (SHMKEY, 1, 0777);
 printf ("Proceso memoria.exe -> shmid=%d\n",shmid);

 /* Nota: Existira otro proceso que habra creado la region
        de la forma "shmget (SHMKEY, 1, 0777 | IPC_CREAT);" */

 /* Enlazamos la region al espacio de direcciones del proceso */
 addr = (char *) shmat (shmid, 0, 0);

 pint = (int *) addr;

 printf ("Ya he obtenido la region de memoria.\n");
printf ("%d",*pint);
 /* Esperamos a que el contenido de la primera direccion tenga un valor distinto
    de cero. Esto lo realizara el proceso conmem.exe */
 while (*pint == 0)
  ;  /* hacer nada */

/* resto del programa */

/* Separamos la region del espacio de direccionamiento */
shmdt (addr);

printf ("El proceso memoria.exe va ha terminar.\n");
printf ("%d",*pint);

sem_oper.sem_num = PRINC;
sem_oper.sem_op = 1;
sem_oper.sem_flg = SEM_UNDO;
semop (semid, &sem_oper, 1);
} /* fin de main */



/*                      prinm.c
   Descripcion: Este proceso crea una lista de semaforos y una region de memoria
            para el proceso memoria.c. Una vez que este ultimo termina,
            el proceso prinm.exe elimina la lista de semaforos y la region
            de memoria.
*/

#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <sys/shm.h>

#define SHMKEY 70
#define SEMKEY 60


main ()
{
```

```
int semid,shmid;
struct sembuf sem_oper;
union semun arg;
struct shmid_ds *shmstatbuf;

/* Creacion de la lista de semaforos */
semid = semget (SEMKEY, 1, 0777 | IPC_CREAT);

/* Inicializacion del semaforo */
arg.array = (unsigned short *) malloc (sizeof (short));
arg.array [0] = 0;
semctl (semid, 1, SETALL, arg);

/* Creacion de la region de memoria de 1Kb de tamano*/
shmid = shmget (SHMKEY, 1024, 0777 | IPC_CREAT);

sem_oper.sem_num = 0;
sem_oper.sem_op = -1;
sem_oper.sem_flg = SEM_UNDO;
semop (semid, &sem_oper, 1);

sem_oper.sem_num = 0;
sem_oper.sem_op = -1;
sem_oper.sem_flg = SEM_UNDO;
semop (semid, &sem_oper, 1);
/* Con esta operacion espero a que termine el proceso memoria.exe para
   eliminar tanto la lista de semaforos como la region de memoria */

semctl (semid, 1, IPC_RMID, 0);   /* Elimina la lista de semaforos */
shmctl (shmid, IPC_RMID, shmstatbuf);   /* Elimina la region de memoria */

}   /* Fin de main */



/*                    conmem.c                    */

/* Este proceso permite que el proceso memoria.c termine, ya que lo que
   hace este proceso es colocar un valor distinto de cero en la primera
   direccion de la region de memoria.   */

#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>

#define SHMKEY 70

main ()
{
int shmid,*pint;
char *addr;

shmid = shmget (SHMKEY, 1, 0777);
addr = (char *) shmat (shmid, 0, 0);
```

```
pint = (int *) addr;
*pint = 1;
shmdt (addr);

printf ("\nFin del proceso conmem.");
}   /* Fin de main */
```

Se debe ejecutar primero el programa prinm.c antes del programa memoria.c. Por último se ejecuta el programa conmem.c. Todos estos programas deben ejecutarse en modo background.

PRÁCTICA:

Crear dos procesos que utilicen memoria compartida para comunicarse. Un proceso introduce texto en la memoria compartida y el otro la lee de allí y la introduce en un archivo.

TEMA 7. EL MODELO CLIENTE/SERVIDOR.

CONCEPTOS BÁSICOS:

El modelo cliente/servidor consiste básicamente en que uno o varios procesos clientes solicitan un determinado servicio a un proceso servidor que realiza la petición solicitada. Una gran ventaja de este modelo es su adaptación a las redes de computadores ya que los procesos clientes o servidor pueden ejecutarse en el mismo host o pueden encontrarse en hosts distintos; pero en cualquier caso, se debe saber en que host se encuentra el proceso servidor.

La forma de comunicar procesos cliente y servidor es mediante buzones (sockets). Un buzón es un punto final para la comunicación entre procesos; o sea un lugar donde se deposita y se separa un cierto número de mensajes. Cada buzón tiene colas para enviar o recibir datos.

Un proceso servidor necesita realizar las siguientes operaciones para poder aceptar peticiones por parte de los procesos cliente:

Obtener la dirección del host local:

La dirección internet se obtienen mediante la función *gethostbyname* la cual busca en el /etc/hosts a partir del nombre oficial. La función devuelve un puntero a una estructura (*hostent*) que contiene la dirección requerida. Su formato es el siguiente:

```
struct hostent {
        char *h_name;           /* nombre oficial */
        char **h_alias;  /* lista de alias */
        int h_addrtype;  /* tipo de dirección */
        int h_lenght;           /* longitud de la dirección */
        char **h_addr_list;     /* lista de direcciones */
        long h_addr;            /* dirección IP */
};
```

Obtener la dirección del puerto local:

La función *getservbyname* permite obtener el número del puerto local de un servicio a partir del nombre del servicio. Para ello busca en el archivo /etc/services. Los argumentos que se le pasan a la función son el nombre del servicio y la familia de protocolos.

La función devuelve un puntero a una estructura (*servent*) que contiene el número del puerto requerido. Su formato es el siguiente:

```
struct servent {
        char *s_name;           /* nombre oficial del servicio */
        char **s_aliases;       /* lista de alias */
        int s_port;             /* número del puerto */
        char s_proto;           /* familia de protocolos */
        char **h_sddr_list      /* lista de direcciones */
};
```

Crear buzón:

Para poder crear los buzones se utiliza la función *socket*. Los argumentos que se le pasan son la familia de protocolos (AF_INET), el tipo de comunicación (SOCK_STREAM ó SOCK_DGRAM) y el protocolo específico que se va a usar (tcp o udp). La función devuelve un entero que identifica al buzón recién creado.

Enlazar dirección al buzón:

La función *bind* enlaza una dirección a un buzón. Los argumentos que se le pasan son el identificador del buzón que va a ser enlazado, un puntero a una estructura de dirección y el tamaño de dicha estructura. El formato de una estructura de dirección es el siguiente:

```
struct sockaddr_in {
        short sin_family;       /* familia de dirección */
        short sin_port;         /* número de puerto */
        long sin_addr;          /* dirección IP */
        …
};
```

Escuchar solicitudes:

La función *listen* solamente indica que la aplicación desea aceptar solicitudes de conexión. Los argumentos que se le pasan son el identificador del buzón y el número máximo de solicitudes pendientes en la cola (límite "backlog").

Aceptar solicitudes:

La función *accept* obtiene la primera conexión de la cola de conexiones pendientes, crea un nuevo buzón con las mismas propiedades que tiene el buzón que se utiliza para aceptar solicitudes y asigna un nuevo descriptor para el buzón recién creado. Los argumentos que se le pasan a la función son el identificador de buzón, un puntero a una estructura de dirección y un puntero a un entero que contiene el tamaño de dicha estructura.

Enviar/recibir información:

La función *send* se utiliza para enviar datos a un buzón conectado. Los argumentos que se le pasan so el identificador del buzón, un puntero al buffer desde donde se va a enviar los datos y la longitud del buffer de recepción.

La función *recv* se utiliza para leer datos desde un socket conectado. Los argumentos que se le pasan son el identificador del buzón devuelto por la función *accept*, un puntero al buffer donde se van a recibir los datos y la longitud del buffer de recepción.

Un proceso cliente necesita realizar las siguientes operaciones para poder solicitar servicios por parte de un proceso servidor:

Obtener la dirección remota:

Se utiliza la función *gethostbyname*, al igual que ocurría con el proceso servidor.

Obtener la dirección del puerto remoto:

Se utiliza la función *getservbyname*, al igual que ocurría con el proceso servidor.

Crear un buzón:

La función *socket* permite crear buzones, de igual forma a como ocurría con el proceso servidor.

Conectar con el servidor:

La función *connect* permite a un cliente iniciar una conexión con el buzón de un servidor. Los argumentos que se le pasan a la función son el identificador del buzón, un puntero a una estructura de dirección y un puntero a un entero que contienen el tamaño de dicha estructura.

Enviar/recibir información:

Se utilizará las funciones *send* o *recv* según sea el flujo de información que interese.

Veamos un programa ejemplo que nos permita comprobar como se usan las llamadas al sistema que permiten crear un modelo cliente/servidor.

```
/*   cliente.c        */
/*   PROCESO CLIENTE   */

/* declaracion de archivos de inclusion */
#include <stdio.h>
#include <signal.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <netinet/in.h>
#include <arpa/inet.h>

main ()
{
 /* declaracion de variables y estructuras de datos */
```

```
 int retcode;
 int sock;
 int errno;
 struct sockaddr_in name;
 struct sockaddr_in addr;
 struct servent *servrec;
 struct hostent *hostrec;
 char buf[40];
 char host[30];
 int flag=0;

 /* Obtener host remoto donde esta el proceso servidor */
 if ((hostrec=gethostbyname ("almeria"))==NULL)
   error (1);

 /* Obtener puerto remoto del servicio */
 if ((servrec=getservbyname ("prueba", "tcp"))==NULL)
   error (2);

 /* Crear un socket */
 if ((sock=socket (AF_INET, SOCK_STREAM, 0))==-1)
   error (3);

 /* Actualizar nombre de socket */
 name.sin_port = servrec->s_port;
 name.sin_family = hostrec->h_addrtype;
 name.sin_addr.s_addr = INADDR_ANY;

 /* Contactar con el servidor */
 if ((retcode=connect (sock, &name, sizeof(name)))==-1)
   error (4);

 /* Enviar informacion al servidor */
 if ((retcode=send (sock, "hola", sizeof ("hola"), flag))==-1)
   error (5);

 /* Cerrar conexion */
 if (close (sock)==-1) error (6);

} /* fin de main */

error (n)
  int n;
{
 switch (n) {
  case 1: printf ("Error en gethostbyname \n");
          break;
  case 2: printf ("Error en getservbyname \n");
          break;
  case 3: printf ("Error en socket \n");
          break;
  case 4: printf ("Error en connect \n");
          break;
  case 5: printf ("Error en send \n");
          break;
  case 6: printf ("Error en close \n");
          break;
```

```
 } /* fin de switch */
 exit (1);
} /* fin de error */



/*   servidor2.c        */
/*   PROCESO SERVIDOR   */

/* declaracion de archivos de inclusion */
#include <stdio.h>
#include <signal.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <netinet/in.h>
#include <arpa/inet.h>

main ()
{
 /* declaracion de variables y estructuras de datos */
 int retcode;
 int sock;
 int newsock;
 int len=0;
 int flag=0;
 struct sockaddr_in name;
 struct sockaddr_in addr;
 struct servent *servrec;
 struct hostent *hostrec;
 char buf[40];
 char host[30];

 /* Obtener nombre del host local */
 if ((hostrec=gethostbyname ("almeria"))==NULL)
   error (1);

 /* Obtener puerto local */
 if ((servrec=getservbyname ("prueba", "tcp"))==NULL)
   error (2);

 /* Crear un socket */
 if ((sock=socket (AF_INET, SOCK_STREAM, 0))==-1)
   error (3);

 /* Actualizar nombre de socket */
 name.sin_port = servrec->s_port;
 name.sin_family = hostrec->h_addrtype;
 name.sin_addr.s_addr = INADDR_ANY;

 /* Enlazar nombre al socket */
 if ((retcode = bind (sock, &name, sizeof (name))) == -1 ) {
   close (sock);
   error (4);
 }
```

```
 /* Escuchar solicitudes */
 if ((retcode=listen (sock, 1))==-1)
   error (5);

 /* Aceptar solicitudes */
 memset (&addr, 0, sizeof (addr));
 len = sizeof (addr);
 if ((newsock = accept (sock, &addr,&len)) == -1)
   error (6);

 /* Recibir un mensaje */
 if (retcode = recv (newsock, buf, sizeof (buf), 0) == -1)
   error (7);

 /* Visualizar informacion transferida */
 printf (" %s \n", buf);

 /* Mandar un mensaje al cliente */
 retcode = send (newsock, "adios", sizeof ("adios"), 0);

 /* Cerrar conexion */
 if (close (newsock)==-1) error (8);
 if (close (sock)==-1) error (8);

} /* fin de main */

error (n)
  int n;
{
 switch (n) {
  case 1: printf ("Error en gethostbyname \n");
          break;
  case 2: printf ("Error en getservbyname \n");
          break;
  case 3: printf ("Error en socket \n");
          break;
  case 4: printf ("Error en bind \n");
          break;
  case 5: printf ("Error en listen \n");
          break;
  case 6: printf ("Error en accept \n");
          break;
  case 7: printf ("Error en recv \n");
          break;
  case 8: printf ("Error en close \n");
          break;
  case 9: printf ("Error en send2 \n");
          break;
 } /* fin de switch */
 exit (1);
} /* fin de error */
```

BIBLIOGRAFÍA

Brian W. Kernigham
El Entorno de Programación UNIX.
Prentice-Hall Hispanoamericana, S.A.

Maurice J. Bach
The Design of the UNIX Operating System.
Prentice-Hall Software Series.

Paul W. Abrahams y Bruce R. Larson
UNIX para impacientes.
Addison-Wesley Iberoamericana.

Introducción al UNIX Sistema V
Rachel Morgan y Henry McGilton
McGraw-Hill

UNIX. Sistema V versión 4. Manual de referencia.
Stephen Coffin
McGraw-Hill

Curso Introducción a la Administración y Programación en el entorno SunOs.
Departamento de Lenguajes y Computación
Universidad de Almería-España.

SCO UNIX Operating System. Manual del usuario.
SCO Open System Software.

INDICE

BLOQUE 1: DISEÑO DEL UNIX.

Tema 1: Introducción.
- Características principales.
- Historia de UNIX.
- El papel de C en UNIX.

Tema 2: Estructura de UNIX.
- Estructura del sistema.
- Arquitectura del sistema operativo UNIX.
- El núcleo o kernel de UNIX.

Tema 3: Estructura del Sistema de Archivos (File System).
- El sistema de archivos.
- Conceptos básicos sobre archivos.
- Estructura interna del sistema de archivos.
- Descripción del inodo.
- Directorios.
- Dispositivos y archivos especiales.
- Permisos de un archivo.

Tema 4: Procesos.
- Entorno de procesamiento.
- Modos de ejecución.
- Detención de procesos.

BLOQUE 2: UNIX A NIVEL DE USUARIO.

Tema 1: Usuarios y grupos.
- Tipos de usuario.
- Grupos.

Tema 2: Conexión al sistema.

- Inicio de sesión.
- Fin de sesión.

Tema 3: El Shell.

- ¿Qué es el Shell?
- El shell Bourne (sh).
- El shell C (csh).
- El shell Korn (ksh).
- El shell C mejorado (tcsh).
- Procedimientos de shell o shell scripts.

Tema 4: Comandos más comunes.

- Sintaxis de las órdenes.
- Metacaracteres.
- Ordenes de manipulación de directorios.
- Ordenes de manipulación de archivos.
- Modificación de permisos y propietarios.
- Impresión.
- Ordenes diversas.
- Procesos.

Tema 5: Programación con el Shell.

- ¿Por qué un shell programable?
- Creación de nuevos comandos.
- Argumentos y parámetros en los comandos.
- La salida de programas como argumentos.
- Variables de shell o de entorno.
- Iteraciones en los programas de shell.
- Filtros.
- Selección en los programas de shell.
- Selección múltiple en los programas de shell.
- El ciclo *while*: a la espera de sucesos.
- El ciclo *repeat*.

Tema 6: El editor de pantalla vi.

- Introducción.
- Modos de funcionamiento.
- Ordenes de entrada al editor *vi*.
- Comandos más usuales.

Tema 7: Comunicación entre usuarios.

- Correo electrónico con *mail*.
- La orden *write*.

Tema 8: La documentación del sistema UNIX.

- El Manual del usuario UNIX.
- Estructura del Manual del usuario.

- La orden *man* en-línea.

BLOQUE 3: ADMINISTRACIÓN DEL SISTEMA.

Tema 1: El superusuario y sus funciones.
- El superusuario.
- Tareas administrativas básicas.

Tema 2: Arranque y parada del sistema.
- Arranque del sistema.
- Parada del sistema.

Tema 3: Administración de los sistema de archivos.
- Montar un sistema de archivos.
- Crear un sistema de archivos.
- Integridad de los sistemas de archivos.
- Utilización del disco.
- Organización eficiente de los sistema de archivos.

Tema 4: Administración de las cuentas de usuario.
- Introducción. Archivos administrativos.
- Creación/borrado/modificación de cuentas de usuario.
- Creación/borrado de grupos.
- Listado de usuarios del sistema.

Tema 5: Dispositivos.
- Creación de dispositivos.

Tema 6: Administración de terminales.
- Comunicación.
- Archivos para la gestión de terminales.

Tema 7: Administración de impresoras.
- Creación de un puerto paralelo.
- Creación de una impresora.

Tema 8: Copias de seguridad. Recuperación.
- Introducción.
- Copias incrementales con *backup*/*restore*.
- Copias específicas con *tar* y *cpio*.

Tema 9: Ejecución periódica de tareas. La utilidad *cron*.
- Introducción.
- La orden *crontab*.

Tema 10: Comunicación con el usuario.

- Introducción.
- La orden *news*.
- El mensaje del día.
- La orden *wall*.

Tema 11: Seguridad del sistema.
- Introducción.
- Definición de usuarios.
- Recomendaciones de comandos UNIX.
- Seguridad en archivos.
- Administración del sistema.
- Recomendaciones adicionales.

BLOQUE 4: REDES EN UNIX.

Tema 1: Arquitectura de TCP/IP.
- Visión general de las redes de comunicaciones.
- Tipos de redes soportados por UNIX.
- Introducción a TCP/IP.
- Hardware en una red basada en TCP/IP.
- Arquitectura de TCP/IP. Relación con el modelo OSI.
- Direcciones Internet.
- Enrutamiento.

BLOQUE 5: PROGRAMACIÓN CON EL INTERFAZ DE USUARIO.

Tema 1: El compilador de C.
- Compilación y ejecución de programas en C.
- El compilador *cc*.

Tema 2: Herramientas para la depuración.
- Introducción.
- Tipos de depuración soportados por *dbx* y *dbxtool*.
- *dbx*.
- Práctica.

Tema 3: Introducción a los servicios del sistema.
- Introducción.

Tema 4: Creación y ejecución de procesos.
- Conceptos básicos.
- Creación de procesos.
- Ejecución de procesos.
- Espera y finalización de procesos.

- Práctica.

Tema 5: Semáforos.
- Conceptos básicos.
- Práctica.

Tema 6: Memoria compartida.
- Conceptos básicos.
- Práctica.

Tema 7: El modelo cliente/servidor.
- Conceptos básicos.