TEMARIO

- **INTRODUCCION (4)**
- **EVOLUCION HISTORICA (5)**
- **CARACTERISTICAS GENERALES (15)**

3.1 CONCEPTOS BASICOS (16)

- **SESION UNIX (17)**
- **ESTRUCTURA DEL SISTEMA OPERATIVO (19)**
- **EL NUCLEO DEL SISTEMA OPERATIVO (22)**
- **SISTEMA DE ARCHIVOS (25)**
- **ADMINISTRACION DE ARCHIVOS Y DIRECTORIOS (26)**
- **MANEJO DE ARCHIVOS Y DE INFORMACION (28)**
- **CARACTERISTICAS DE LA VERSION 4 (29)**
- **ARCHIVOS Y DIRECTORIOS (30)**
- **TIPOS DE ARCHIVOS (31)**
- **PERMISOS Y PROPIETARIOS DE ARCHIVOS (33)**
- **DIRECCIONAMIENTO DE ENTRADA Y SALIDA (35)**
- **ESTRUCTURA DE LA LINEA DE ORDENES (35)**
- **ESTRUCTURA JERARQUICA DE FICHEROS (36)**
- **MONTAJE DEL SISTEMA DE FICHEROS (37)**

4.13 ARBOL DE DIRECTORIOS DE UNIX (38)

- **CONTROL DE PROCESOS (39)**

5.1 PRIORIDADES DE PROCESOS (41)

5.2 ORDENES DE TRATAMIENTO DE PROCESOS (42)

- **PROCESOS. MANEJO DEL PROCESADOR (42)**
- **GESTION Y MANEJO DE MEMORIA (45)**

6.1 MANEJO DE ENTRADAS Y SALIDAS (47)

6.2 LENGUAJE DE CONTROL DEL SISTEMA OPERATIVO (48)

6.3 ESTRUCTURA DE LA LINEA DE COMANDOS (52)

6.4 SISTEMA DE FICHEROS (53)

6.5 ORGANIZACIÓN DEL SISTEMA DE FICHEROS (53)

- **CACHÉ DE BLOQUES (55)**

6.7 DESCRIPCIÓN DEL MODO DE ACCESO A FICHEROS (56)

- **EL SHELL INTÉRPRETE DE COMANDOS (57)**

7.1 EL SHELL DE PRESENTACIÓN (58)

7.2 TIPOS DE SHELL (59)

7.3 SHELL DE KORN (60)

7.4 FICHEROS DE INICIALIZACIÓN (60)

7.5 VARIABLES DEL SHELL (61)

7.6 OBTENCIÓN DEL VALOR DE UNA VARIABLE (62)

7.7 DEFINICIÓN DE VARIABLES EN EL SHELL (62)

7.8 EXPORTACIÓN DE VARIABLES (62)

7.9 VARIABLES NOCLOBBER e IGNOREEOF (64)

7.10 HISTÓRICO DE ÓRDENES (64)

7.11 ALIAS DE ÓRDENES (64)

7.12 EJECUCION DE ORDENES (66)

7.12.1 EN MODO SUBORDINADO (66)

7.13 LA E/S EN MODO SUBORDINADO (66)

7.14 MANTENIMIENTO DE TRABAJOS ACTIVOS (67)

7.15 ÓRDENES DE CONTROL DE TRABAJOS (67)

7.16 EL SHELL DE TRABAJO (68)

- **HERRAMIENTAS DE DESARROLLO DE SOFT (69)**

8.1 LA PROGRAMACION DEL SHELL (70)

8.1.1 GUIONES DEL SHELL (70)

8.1.2 EJECUCIÓN DE UN GUIÓN (71)

8.1.3 COMENTARIOS EN LOS GUIONES DEL SHELL (71)

8.2 PARÁMETROS POSICIONALES (72)

8.3 OTRAS VARIABLES DEL GUIÓN (72)

8.4 SENTENCIAS DE CONTROL EN LA PROGRAMACIÓN SHELL (73)

8.5 ÓRDENES DE CONTROL (74)

8.6 OPERACIONES ARITMÉTICAS (77)

8.7 OTRAS ÓRDENES DEL SHELL (77)

8.8 OTRAS ÓRDENES (77)

8.9 MATRICES Y FORMA DE ACCESO (78)

8.10 LA HERRAMIENTA AWK (79)

8.11 PATRONES (79)

8.12 ACCIONES (81)

8.13 OPERADORES (82)

8.14 ARRAYS (82)

8.15 FUNCIONES DEFINIDAS POR EL USUARIO (82)

8.16 SENTENCIAS DE CONTROL (83)

8.17 FUNCIONES DE UNIX (83)

8.18 E/S EN AWK (83)

9. ADMINISTRACION DE SISTEMAS (87)

CONCLUSIONES (89)

BIBLIOGRAFIA (90)1. INTRODUCCION AL S.O UNIX

Se trata de un sistema operativo de los mas utilizados y con mas futuro debido a que son muchos organismos oficiales y particulares los que defienden su utilización, así como muchas firmas de fabricación y comercialización de computadoras que lo incorporan en sus productos. Podemos citar el ejemplo de la Comunidad Económica Europea, que impone el sistema operativo UNIX en todas las aplicaciones que se desarrollan bajo sus auspicios.

Comenzaremos la descripción del sistema operativo UNIX con una breve reseña histórica para llegar la situación actual e indicar los posibles motivos de su fuerte expansión. Seguiremos dando pequeñas indicaciones de su estructura interna para pasar a hablar del Interprete de Comandos (Shell), finalizando con una descripción de las herramientas de ayuda en el desarrollo de software.

2. HISTORIA DEL SISTEMA OPERATIVO UNIX

La historia del sistema operativo UNIX es la que se describe a continuación, en consonancia con la evolución representada en la figura siguiente.

1965: *Las empresas Bell Telephone Laboratories y General Electric Company intervienen en el proyecto MAC del Massachusets Institute Tecnology (MIT) para desarrollar un nuevo sistema operativo denominado MULTICS, cuyo objetivo fue el ofrecer un sistema multiusuario (acceso simultaneo de gran numero de usuario) de gran potencia de proceso, gran capacidad de almacenamiento y con grandes facilidades para*

compartir datos entre procesos.

1969: *Vistos los resultados poco satisfactorios del MULTICS, la Bell Telephone Laboratories se retira del proyecto y desarrolla un sistema de tiempo compartido con pagina por demanda para uso interno de la empresa sobre la computadora PDP − 7 de Digital, siendo los artífices del mismo un equipo encabezado por Ken Thompson y Denis Ritchie. Este sistema operativo constituyó la primera versión del UNIX, que solo permitía la explotación en monoprogramacion*

1971: *El resultado del sistema anterior tuvo tanto éxito que la compañía puso a disposición de Thompson y Ritchie una computadora mas potente, que fue la PDP − 11 de Digital. En ella, Thompson desarrollo el lenguaje de programación B inspirándose en BCPL y en el FORTRAN, y a continuación Ritchie creo el lenguaje C, con el que consiguió la generación de código de maquina, descripción de tipos de datos y de estructuras de datos.*

1973: *Sé reescribe en C la versión de UNIX desarrollada en ensamblador y que prácticamente es la que se ha mantenido hasta hoy. Aparece una versión de UNIX conocida como Programmer's Workbench (PWB).*

1974: *Se introduce el sistema operativo UNIX en las universidades norteamericanas con fines educativos.*

1977: *Se construye la primera versión comercial de UNIX, conocida como la versión 6, implantándose por primera vez en una computadora distinta de la PDP, que fue la INTERDATA 8/32.*

1979: *Aparece la versión 7 de UNIX para PDP y una versión para la computadora VAX de Digital (32 bits), conocida como 32V.*

1981: *Nace la primera versión de UNIX para computadoras personales con el nombre de XENIX.*

1982: *Para la distribución externa, los laboratorios Bell desarrollan el UNIX System III, que no es mas que el original con algunas pequeñas variantes. Por otra parte, la Universidad de Berkeley desarrolla una variante del UNIX 32V para computadoras VAX con mejoras en cuanto a comandos y gestión de la memoria virtual paginada, denominada 4.1 BSD.*

1983: *La empresa AT&T anuncia una nueva versión denominada UNIX System V, que es el sistema actual y que presenta importantes mejoras de rendimiento, comunicaciones, etc.*

1984: *La Universidad de Berkeley presenta la versión 4.2 BSD para computadoras VAX, que también se aplica en estaciones de trabajo SUN 2/3 de SUN MICROSYSTEMS.*

En la actualidad se utilizan fundamentalmente dos versiones del sistema operativo:

- *UNIX System V.*
- *XENIX System V*

En la mayoria de los casos, cada fabricante tiene su propia version de UNIX, entre ellas citaremos:

- *Digital Equipment Corporation ULTRIX*
- *IBM AIX*
- *Data General DG / UX*
- *National Semiconductor GENIX*
- *Gould Concept 32 / 6750 UTX*
- *Hewlett Packard HP − UX*

A partir de la 7ª edición de UNIX, AT&T que había adquirido los laboratorios BELL decidió hacerlo comerciable no distribuyendo el código fuente de las siguientes variaciones dándole el nombre de UNIX System V.

A partir de ahí, evolucionó como cualquier otro sistema y ahora pertenece a SANTA CRUZ que es de AT&T.

EVOLUCION DEL SISTEMA OPERATIVO UNIX

3. CARACTERÍSTICAS GENERALES

Es un sistema operativo de tiempo compartido, controla los recursos de una computadora y los asigna entre los usuarios. Permite a los usuarios correr sus programas. Controla los dispositivos de periféricos conectados a la máquina.

POSEE LAS SIGUIENTES CARACTERÍSTICAS:

- *Es un sistema operativo multiusuario, con capacidad de simular multiprocesamiento y procesamiento no interactivo.*
- *Está escrito en un lenguaje de alto nivel: C.*
- *Dispone de un lenguaje de control programable llamado SHELL.*
- *Ofrece facilidades para la creación de programas y sistemas y el ambiente adecuado para las tareas de diseños de software.*
- *Emplea manejo dinámico de memoria por intercambio o paginación.*
- *Tiene capacidad de interconexión de procesos.*
- *Permite comunicación entre procesos.*
- *Emplea un sistema jerárquico de archivos, con facilidades de protección de archivos, cuentas y procesos.*
- *Tiene facilidad para redireccionamiento de Entradas/Salidas.*
- *Garantiza un alto grado de portabilidad.*

El sistema se basa en un Núcleo llamado Kernel, que reside permanentemente en la memoria, y que atiende a todas las llamadas del sistema, administra el acceso a los archivos y el inicio o la suspención de las tareas de los usuarios.

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

− Ofrece las estructuras de control normales: secuenciación, iteración condicional, selección y otras.

− Paso de parámetros.

− Sustitución textual de variables y Cadenas.

− Comunicación bidireccional entre órdenes de shell.

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

Las entradas y salidas pueden ser redireccionadas o redirigidas hacia archivos, procesos y dispositivos;

Es posible interconectar procesos entre sí.

Diferentes usuarios pueden "ver" versiones distintas del sistema operativo debido a la capacidad del shell para configurar diversos ambientes de ejecución. Por ejemplo, se puede hacer que un usuario entre directamente a su sección, ejecute un programa en particular y salga automáticamente del sistema al terminar de usarlo.

3.1 CONCEPTOS BASICOS

Para abordar un pequeño recorrido a traves del sistema operativo UNIX, es necesario conocer ciertos conceptos y terminologia comun a todo el sistema.

La comunicación entre la computadora y el terminal teclado − pantalla que utilizara el usuario para trabajar bajo el control del sistema operativo es siempre Full − Duplex y podran ser tecleados todos los caracteres que deseen, mas los caracteres de control que tienen significado especial.

CARACTER DE CONTROL	*SIGNIFICADO*
RETURN	*Fin de E / S*
Ctrl − m	*Fin de E / S*
Ctrl − d	*Fin de la sesion*
Ctrl − g	*Acciona la campana*
Ctrl − h	*Retroceso (Backspace)*
Ctrl − i	*Tabulador*
DELETE	*Borra un caracter*
BREAK	*Produce una interrupcion*
Ctrl − c	*Produce una interrupcion*

3.2 SESION UNIX

El trabajo de un usuario en el sistema operativo UNIX se organiza por sesiones. Una sesion UNIX comprende todo el trabajo realizado por el usuario en la computadora, desde que se identifica hasta que se despide.

Vamos a exponer brevemente cuales deben ser las acciones que ha de llevar a cabo el usuario para dar comienzo a una sesion, operar a lo largo de ella y finalizarla correctamente.

La figura siguiente muestra un ejemplo de inicio y terminacion de una sesion de usuario. En ella puede observarse que el primer paso es el de identificacion del usuario, en el que ademas el sistema pide la correspondiente palabra o clave de usuario para proteger el acceso de otros usuarios. Una vez reconocida la clave por el sistema, dara comienzo a la sesion.

Aparece en el terminal en caracer que indica espera de mandato o comando, que sera interpretado por el interprete de comandos o Shell y que se denomina prompt, representandose en general por los caracteres $ o %. Cuando se termina el trabajo que se pretendia hacer, basta con despedirse de la sesion por medio del carácter de control Ctrl − d (D).

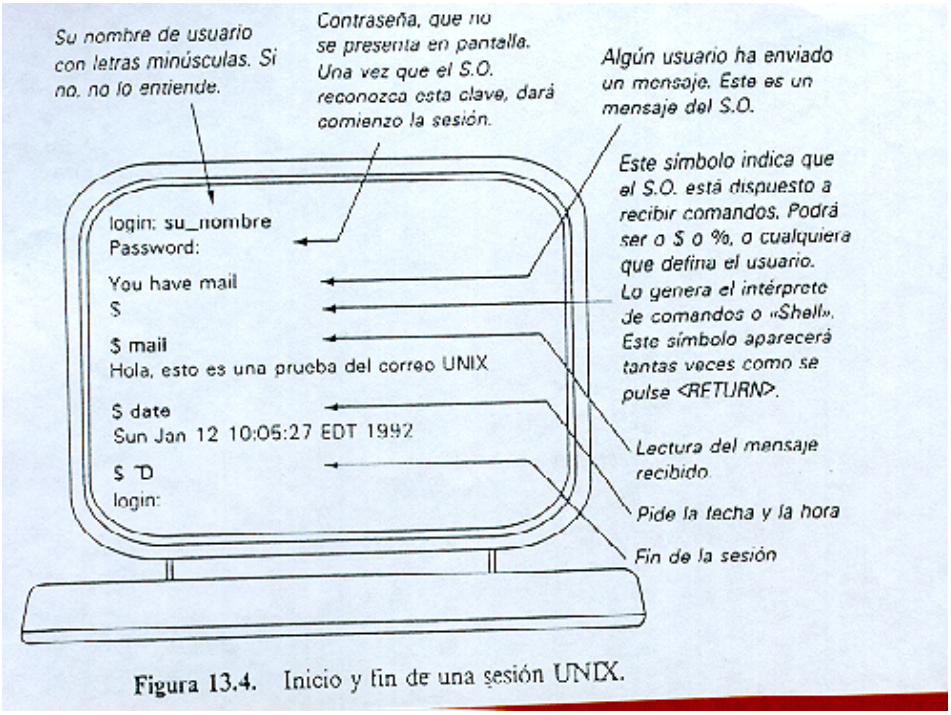


Figura 13.4. Inicio y fin de una sesión UNIX.

4. ESTRUCTURA

El sistema operativo UNIX como ya dijimos es un sistema operativo de tiempo compartido y por lo tanto, multiusuario, en el que existe la portabilidad para la implementacion de distintas computadoras.

Esta formado por una serie de elementos que pueden representarse en forma de capas concéntricas donde, en primer lugar, alrededor del hardware, aislando a este de los usuarios, además de adaptar el resto del sistema operativo a la maquina debido a la portabilidad que existe en el mismo.

COMPILADORES

INTERPRETE DE COMANDOS

KERNEL

HARDWARE

NUCLEO

SHELL

APLICACIONES DE USUARIO

En una segunda capa se encuentran los comandos, que no son otra cosa que el Interface entre los programas de aplicación y el núcleo del sistema operativo.

La ultima de las capas contiene los programas de aplicación.

El sistema operativo UNIX se compone de bloques funcionales que se encuentran representados en la Figura siguiente.

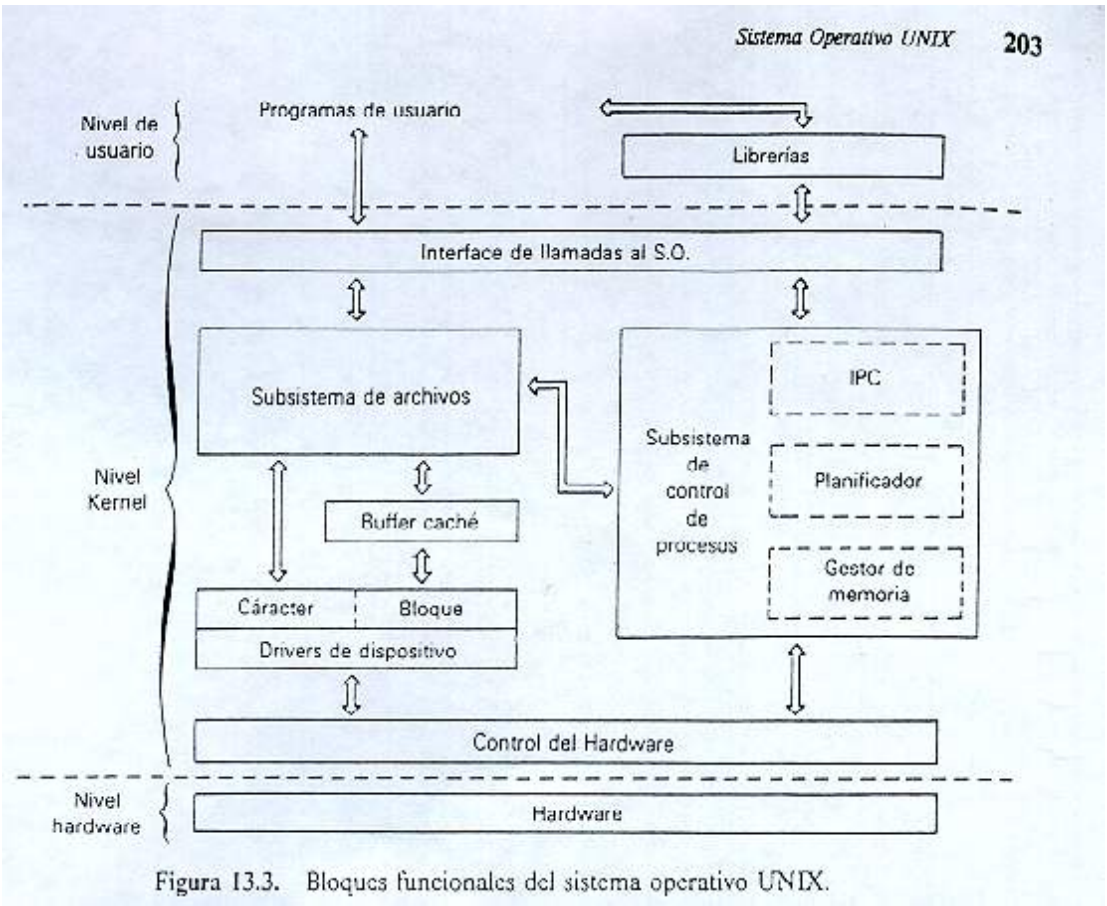


Figura 13.3. Bloques funcionales del sistema operativo UNIX.

4.1 EL NÚCLEO DEL SISTEMA OPERATIVO

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

Las funciones del núcleo son permitir la existencia de un ambiente en el que sea posible atender a varios usuarios y múltiples tareas en forma

concurrente, repartiendo al procesador entre todos ellos, e intentando mantener en grado óptimo la atención individual.

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

− Creación de procesos, asignación de tiempos de atención y sincronización.

− Asignación de la atención del procesador a los procesos que lo requieren.

− Administración de espacio en el sistema de archivos, que incluye: acceso, protección y administración de usuarios; comunicación entre usuarios v entre procesos, y manipulación de E/S y administración de periféricos.

− Supervisión de la transmisión de datos entre la memoria principal y los dispositivos periféricos.

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

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

Cuando se inicia la operación de la computadora, debe cargarse en la memoria una copia del núcleo, que reside en e] disco magnético (operación denominada bootstrap). Para ello, se deben inicializar algunas interfaces básicas de hardware; entre ellas, el reloj que proporciona interrupciones periódicas. El Kernel también prepara algunas estructuras de datos que abarcan una sección de almacenamiento temporal para transferencia de información entre terminales y procesos, una sección para almacenamiento de descriptores de archivos y una variable que indica la cantidad de memoria principal.

A continuación, el Kernel inicializa un proceso especial, llamado proceso 0. En general, los procesos se crean mediante una llamada a una rutina del sistema (fork), que funciona por un mecanismo de duplicación de procesos. Sin embargo, esto no es suficiente para crear el primero de ellos, por lo que el Kernel asigna una estructura de datos y establece apuntadores a una sección especial de la memoria, llamada tabla de procesos, que contendrá los descriptores de cada uno de los procesos existentes en el sistema.

Después de haber creado el proceso 0, se hace una copia del mismo, con lo que se crea el proceso 1; éste muy pronto se encargará de "dar vida" al sistema completo, mediante la activación de otros procesos que también forman parte del núcleo. Es decir, se inicia una cadena de activaciones de procesos, entre los cuales destaca el conocido como despachador, o scheduler, que es el responsable de decidir cuál proceso se ejecutará y cuáles van a entrar o salir de la memoria central. A partir de ese momento se conoce el número 1 como proceso de inicialización del sistema, init.

El proceso init es el responsable de establecer la estructura de procesos en UNIX. Normalmente, es capaz de crear al menos dos estructuras distintas de procesos: el modo monousuario y el multiusuario. Comienza activando el intérprete del lenguaje de control (Shell) en la terminal principal, o consola, del sistema y proporcionándole privilegios de "superusuario". En la modalidad de un solo usuario la consola permite iniciar una primera sesión, con privilegios especiales, e impide que las otras líneas de comunicación acepten iniciar sesiones nuevas. Esta modalidad se usa con frecuencia para revisar y reparar sistemas de archivos, realizar pruebas de funciones básicas del sistema y para otras actividades que requieren uso exclusivo de la computadora.

Init crea otro proceso, que espera pacientemente a que alguien entre en *sesión en alguna línea de comunicación. Cuando esto sucede, realiza ajustes en el protocolo de la línea y ejecuta el programa login, que se encarga de atender inicialmente a los nuevos usuarios. Si la clave del usuario, y la contraseña proporcionadas son las correctas, entonces entra en operación el programa Shell, que en lo sucesivo se encargará de la atención normal del usuario que se dio de alta en esa terminal.*

A partir de ese momento el responsable de atender al usuario en esa terminal es el intérprete Shell.

Cuando se desea terminar la sesión hay que desconectarse de Shell (y, por lo tanto, de UNIX), mediante una secuencia especial de teclas (usualmente. < CTL > − D). A partir de ese momento la terminal queda disponible para atender a un nuevo usuario.

4.2 SISTEMA DE ARCHIVOS

El concepto de archivo es básico en la organización de la información en UNIX.

Todo lo que puede tratar con información es un archivo.

Archivo será también una impresora, la pantalla...etc.

UNIX es un sistema dirigido a archivos.

UNIX se basa en tres puntos:

- *Simplicidad*
- *Generalidad*
- *Extensibilidad*

Cualquier tarea que se realice en UNIX está formada por la combinación de componentes simples.

Todas las componentes del sistema tienen un funcionamiento muy global.

Es un gran conjunto de elementos pequeños ; cada una de las tareas hace una función muy pequeña.

Es necesario que todas las ordenes formen una cosas global.

Las herramientas son muy genéricas.

Para programar en UNIX hay que prepararles en muchas partes pequeñas.

Como el código fuente era disponible todas las empresas investigadoras tienen su versión con sus mejoras.

4.3 ADMINISTRACIÓN DE ARCHIVOS Y DIRECTORIOS

El sistema de archivos de UNIX; esta basado en un modelo arborescente y recursivo, en el cual los nodos pueden ser tanto archivos como directorios, y estos últimos pueden contener a su vez directorios o subdirectorios. Debido a esta filosofía, se maneja al sistema con muy pocas órdenes, que permiten una gran gama de posibilidades. Todo archivo de UNIX está controlado por múltiples niveles de protección, que especifican los permisos de acceso al mismo. La diferencia que existe entre un archivo de datos, un programa, un manejador de entrada/salida o una instrucción ejecutable se refleja en estos parámetros, de modo que el sistema operativo adquiere características de coherencia y elegancia que lo distinguen.

La raíz del sistema de archivos (conocida como root) se denota con el símbolo /, y de ahí se desprende un conjunto de directorios que contienen todos los archivos del sistema de cómputo. Cada directorio, a su vez, funciona como la subraíz de un nuevo árbol que depende de él y que también puede estar formado por directorios o subdirectorios y archivos. Un archivo siempre ocupará el nivel más bajo dentro del árbol, porque de un archivo no pueden depender otros; si así fuera, sería un directorio. Es decir, los archivos son como las hojas del árbol.

Se define en forma unívoca el nombre de todo archivo (o directorio) mediante lo que se conoce como su trayectoria (path name): es decir, el conjunto completo de directorios, a partir de root (/), por los que hay que pasar para poder llegar al directorio o archivo deseado. Cada nombre se separa de los otros con el símbolo /, aunque tan sólo el primero de ellos se refiere a la raíz.

Por ejemplo, el archivo

u/gerencia/abril94l/carta2

tiene toda esta trayectoria como nombre absoluto, pero se llama gerencia/abril94l/carta2, sin 1ra diagonal inicial, si se observa desde el directorio /u. Para los usuarios que están normalmente en el directorio /u/gerencia, el archivo se llama abril94l/carta2. Así, también puede existir otro archivo llamado carta2, pero dentro de algún otro directorio y en caso de ser necesario se emplearía el nombre de la trayectoria (completa o en partes, de derecha a izquierda) para distinguirlos. UNIX ofrece medios muy sencillos para colocarse en diferentes puntos del árbol que forma el sistema de archivos, que para el ejemplo anterior podría ser el siguiente:

Como se dijo antes, desde el punto de vista del directorio abril94, que a su vez pertenece al directorio gerencia del directorio /u, basta con el nombre carta2 para apuntar al archivo en cuestión.

En esta forma se maneja el sistema completo de archivos y se dispone de un conjunto de órdenes de Shell (además de múltiples variantes) para hacer *diversas manipulaciones, como crear directorios, moverse dentro del sistema de archivos, copiarlos, etcétera.*

UNIX incluye, además, múltiples esquemas para crear, editar y procesar documentos. Existen varios tipos de editores, formadores de textos, macroprocesadores para textos, formadores de tablas, preprocesadores de expresiones matemáticas y un gran número de ayudas y utilerías diversas, que se mencionan más adelante.

A continuación se describe el modo de funcionamiento de UNIX, con base en un modelo de estudio de sistemas operativos que lo divide en "capas" jerárquicas para su mejor comprensión.

4.4 MANEJO DE ARCHIVOS Y DE INFORMACIÓN

Como ya se describió, la estructura básica del sistema de archivos es jerárquica, lo que significa que los archivos están almacenados en varios niveles. Se puede tener acceso a cualquier archivo mediante su trayectoria, que especifica su posición absoluta en la jerarquía, y los usuarios pueden cambiar su directorio actual a la posición deseada. Existe también un mecanismo de protección para evitar accesos no autorizados. Los directorios contienen información para cada archivo, que consiste en su nombre y en un número que el Kernel utiliza para manejar la estructura interna del sistema de archivos, conocido como el nodo−i. Hay un nodo−i para cada archivo, que contiene información de su directorio en el disco, su longitud, los modos y las fechas de acceso, el autor, etc. Existe, además, una tabla de descriptores de archivo, que es una estructura de datos residente en el disco magnético, a la que se tiene acceso mediante el sistema mencionado de E/S por bloques.

El control del espacio libre en el disco se mantiene mediante una lista ligada de bloques disponibles. Cada bloque contiene la dirección en disco del siguiente bloque en la cadena. El espacio restante contiene las direcciones de grupos de bloques del disco que se encuentren libres. De esta forma, con una operación de E/S, el sistema obtiene un conjunto de bloques libres y un apuntador para conseguir más.

Las operaciones de E/S en archivos se llevan a cabo con la ayuda de la correspondiente entrada del nodo−i en la tabla de archivos del sistema. El usuario normalmente desconoce los nodos−i porque las referencias se hacen por el nombre simbólico de la trayectoria. Los procesos emplean internamente funciones primitivas

(llamadas al sistema) para tener acceso a los archivos; las más comunes son open, creat, read, write, seek, close y unlink, aunque sólo son empleadas por los programadores, no por los usuarios finales del sistema.

Toda esta estructura física se maneja "desde afuera" mediante la filosofía jerárquica de archivos y directorios ya mencionada, y en forma totalmente transparente para el usuario. Además, desde el punto de vista del sistema operativo, un archivo es muy parecido a un dispositivo.

Las ventajas de tratar a los dispositivos de E/S en forma similar a los archivos normales son múltiples: un archivo y un dispositivo de E/S se tornan muy parecidos; los nombres de los archivos y de los dispositivos tienen la misma sintaxis y significado, así que a un programa que espera un nombre de archivo como parámetro puede dársele un nombre de dispositivo (con esto se logra interacción rápida y fácil entre procesos de alto nivel).

El sistema UNIX ofrece varios niveles de protección para el sistema de archivos, que consisten en asignar a cada archivo el número único de identificación de su dueño, junto con nueve bits de protección, que especifican permisos de lectura, escritura y ejecución para el propietario, para otros miembros de su grupo (definido por el administrador del sistema) y para el resto de los usuarios. Antes de cualquier acceso se verifica su validez consultando estos bits, que residen en el nodo−i de todo archivo. Además, existen otros tres bits que se emplean para manejos especiales, relacionados con la clave del superusuario.

Otra característica de UNIX es que no requiere que el conjunto de sistemas de archivos resida en un mismo dispositivo.

Es posible definir uno o varios sistemas "desmontables", que residen físicamente en diversas unidades de disco. Existe una orden (mkfs) que permite crear un sistema de archivos adicional, y una llamada al sistema (mount) con la que se añade (y otra con la que se desmonta) uno de ellos al sistema de archivos global.

EI control de las impresoras de una computadora que funciona con el sistema operativo UNIX consiste en un subsistema (SPOOL) que se encarga de coordinar los pedidos de impresión de múltiples usuarios. Existe un proceso de Kernel que en forma periódica revise las colas de servicio de las impresoras para detectar la existencia de pedidos e iniciar entonces las tareas de impresión. Este tipo de procesos, que son activados en forma periódica por el núcleo del sistema operativo, reciben en UNIX el nombre de daemons (duendes), tal vez porque se despiertan y aparecen sin previo aviso. Otros se encargan de activar procesos en tiempos previamente determinados por el usuario, o de escribir periódicamente los contenidos de los buffers de memoria en el disco magnético.

4.5 CARACTERISTICAS DE LA VERSION 4 (MEJORAS)

- *Conjunto unificado de ordenes (no estándar)*
- *Ofrece un conjunto de Shell´s estándares. Todos ofrecen los mismos : ksh, csh, sh, jsh,rsh (lynux bash)*
- *Sistemas de archivos en red.*
- *Acceso al sistema más sencillo.*
- *Herramientas de administración.*
- *Procesamiento en tiempo real.*
- *Internacionalización del sistema.*
- *Conexiones más sencillas.*

4.6 ARCHIVOS Y DIRECTORIOS

Lestructura de archivos y directorios es jerarquica y esta cnstituida de forma arborescente, donde los nodos son directorios y las hojas son archivos normales. El árbol tiene un directorio raíz único root que se identifica por /.

El sistema de archivos UNIX gestiona varios tipos de archivos:

- ***Archivos :***

Pueden ser normales o regulares: *Son los archivos de usuario que contienen programas.*

Directorios: *Son particiones lógicas que a su vez son archivos que contienen la información necesaria para poder encontrar un archivo en el disco.*

Archivos especiales: *Se utilizan para designar periféricos de entrada y salida.*

- ***Pipes :*** *archivos que permiten la transferencia de datos entre procesos*
- *Dispositivos organizados por bloques.*
- *Dispositivos organizados por caracteres.*

El sistema de archivos, cada archivo tiene asociado un conjunto de permisos que determinan que puede hacerse con el y quienes tiene acceso.

(Estructura básica de almacenamiento de información)

Desde un punto de vista físico es una secuencia de bytes.

*Los nombres de los archivos pueden tener hasta **256** caracteres y pueden incluir una extensión.*

Tienen que ser una secuencia que no incluya el carácter ` / '.

Tampoco es recomendable incluir los siguientes caracteres:

` ! ', ` # ', ` ( `, ` ) ', ` ` ', ` ', ` ; ', `<', `>', Tab,

` $ ', ` ^ ', ` { `, ' } ', ` * ', ` ? ', ` \ `, Espace, Backspace

Solo admiten una extensión.

- ***Directorios :*** *Un directorio es un tipo de archivo que contiene información*

acerca de otros archivos. Se dice que estos otros archivos están dentro del directorio.

La información que contiene es sobre otros archivos. Un directorio puede contener subdirectorios.

El sistema de directorios de UNIX tiene forma de grafo acíclico.

- ***Metacaracteres :*** *El SHELL de UNIX posee un conjunto de operadores*

sobre caracteres que permite especificar más de un valor. Lo más habitual son: ` * '(combinación de caracteres) y ` ?' (cualquier único carácter)

Los metacaracteres son traducidos por el SHELL de forma que la orden no tiene constancia de su existencia.

Ej :

ls (lista el contenido de un directorio)

ls a* ls avion_adios_...etc si avion y adios están en el directorio

Shell ls

El asterisco es mucho más genérico que en MS−DOS :

- **n camión*
- **cion* vacaciones*
- ** nombre.ext*

4.7 TIPOS DE ARCHIVOS

Todo lo que trata con información, UNIX lo trata como un archivo.

Para distinguir unas clases de tratamiento de información con otras, UNIX establece los siguientes tipos de archivos :

- ***Archivos ordinarios :*** *Archivo convencional . Colección de palabras de memoria que contienen datos o código.*
- ***Vinculos :*** *Segundo nombre para un archivo ; permite la compartición de información. La existencia de vinculos es lo que convierte al arbos de UNIX un grafo acíclico.*

ARBOL MS−DOS ARBOL ACÍCLICO UNIX

El vínculo es un enlace con el fichero físico, para la compartición de un archivo para varios usuarios.

No se puede crear un archivo vínculo de la nada, es necesario que exista el archivo físico al cual está vinculado.

Ej.

Al realizar un **ls** *el vinculo viene representado como :*

clase − > / etc / passwd

- ***Vínculos SIMBÓLICOS:*** *Amplia el concepto de vinculo permitiendo vincularse a directorios o ficheros desde la red. Un vinculo simbólico desde el punto de vista físico no es más que un archivo que contiene el nombre de otro. No se implementa un vínculo, sino que viene el archivo vinculado. Al hacer un vinculo, crea una entrada más en la tabla de directorios, y al mismo tiempo un archivo al que está vinculado.*
- ***Directorios :*** *Archivo que contiene información sobre otros archivos(es un fichero). Se puede tratar con las ordenes de tratamiento de archivos.*
- ***Archivos especiales:*** *(De dispositivos) Representas dispositivos, hardware de la máquina y se encargan de traducir las ordenes o las llamadas de los comandos UNIX a las secuencias especificas del hardware instalado. Realiza la ocultación de hardware al sistema. Hacen la traducción, funcionan independientemente del hardware en el que está instalado.*

Para saber el tipo de un archivo: **ls −l** *Muestra información larga sobre ficheros.*

Indica al principio del todo con una letra, el tipo de archivo:

d Directorio

l linker (vinculo)

− Archivos ordinarios

s Archivos especiales

Y muestra más información: Tamaño...etc.

4.8 PERMISOS Y PROPIETARIOS DE ARCHIVOS

Es aplicable a los cinco tipos de archivos.

El borrado de directorios es diferente al borrado de archivos.

En UNIX todos los archivos pertenecen al propietario del proceso que los creó, los archivos del sistema se distribuyen entre varios usuarios del sistema, cada uno de ellos destinado a una labor distinta.

En vez de haber una única cuenta de administrador existen varias cuentas de sistema. Nos podemos conectar como diferentes usuarios que realizan una tarea diferente. Los archivos pertenecen a la cuenta en la han sido creados.

*Para cambiar el propietario de un archivo hay que ser el propietario y ejecutar **CHOWN** (chown) el cual regala el archivo. Un no propietario no puede regalar un archivo que no es suyo.*

*El cambio de propietario no supone cambio de grupo. El fichero sigue perteneciendo al grupo de su creador. Para cambiar el grupo se utiliza **CHGRP**(chgrp).*

Los permisos en UNIX son de tres tipos y se establecen a tres niveles :

Tipos : a. r lectura

- *w escritura*
- *x ejecución*

Niveles : a. Propietario

- *Grupo*
- *Resto de usuarios*

*El único que puede cambiar los permisos es el dueño y lo hace mediante la orden **CHMOD**(chmod).*

Existen 9 posibilidades distintas que se representan mediante 9 bits

r w x r w x r w x

Los permisos de un fichero se representan mediante tres números en octal.

Ej. 7 7 7 (Acceso a cualquier usuario)

*También es posible indicar los permisos existentes para un fichero en el momento de la creación. Esto se llama máscara de permisos y se establece mediante la orden **UMASK** (umask). UMASK establece la máscara hasta que termine la sesión de conexión.*

Si no hay UMASK 6 6 6 Directorios

7 7 7 Ficheros

Para que la orden UMASK este al iniciar una sesión debemos meterla en el ***.profile*** *(Archivo de configuración del SHELL) KSHELL CASHELL ? ?*

El sistema operativo UNIX permite redireccionar tanto la entrada como la salida de cualquier orden sobre ficheros. En realidad un redireccionamiento modifica los valores de los descriptores de archivos sobre los que escriben o leen las ordenes.

Los descriptores de archivos existentes son :

Número	Nombre	Definición
0	Entrada	/dev/tty
1	Salida	/dev/tty
2	Error	/dev/tty

Por defecto están asignados a /dev.

Un redireccionamiento asigna a 0,1,2 un archivo distinto.

Una orden en UNIX lee del descriptor 0, escribe en el descriptor 1, y manda los errores al 2.

Si queremos que escribe lea, o mande los errores a otro sitio se modifica la tabla de descriptores.

El encargado de gestionarlo es el SHELL. El SHELL se encarga de asignar cada número de descriptor al archivo que corresponda.

Los caracteres de redireccionamiento son los siguientes :

> **[fichero]** *Creo fichero y lo lleno con la salida.*

>> **[fichero]** *Añade la salida al archivo.*

< **[fichero]** *Toma la entrada del fichero.*

- **[fichero]** *Error en un fichero nuevo.*

2>> **[fichero]** *Añade la salida de error a un fichero.*

Ejercicio : *Si el comando who me indica quien está conectado y el comando sort ordena un fichero; ¿Cómo obtendríamos la lista ordenada de los usuarios conectados?*

UNIX permite también modificar de una sola vez el descriptor de salida de esa orden y encauzarlo hacia el descriptor de entrada de otra para ello se utiliza el `pipe' ` | '.

Encauzo una orden con otra mediante esta tubería. La tubería crea un archivo temporal que sigue un esquema y sobre el que se garantiza la exclusión mutua.

Ejercicio : *Sacar por pantalla el numero de archivos que contiene un directorio.*

wc −l Cuenta el numero de líneas que escribe.

ls | wc −l Encauzo un listado hacia un contar líneas.

4.9 DIRECCIONAMIENTO DE ENTRADA / SALIDA

El sistema operativo UNIX permite redireccionar tanto la entrada como la salida de cualquier orden sobre ficheros. En realidad un redireccionamiento modifica los valores de los descriptores de archivos sobre los que escriben o leen las ordenes.

Los descriptores de archivos existentes son :

Número	Nombre	Definición
0	Entrada	/dev/tty
1	Salida	/dev/tty
2	Error	/dev/tty

Por defecto están asignados a /dev.

Un redireccionamiento asigna a 0,1,2 un archivo distinto.

Una orden en UNIX lee del descriptor 0, escribe en el descriptor 1, y manda los errores al 2.

Si queremos que escribe lea, o mande los errores a otro sitio se modifica la tabla de descriptores.

El encargado de gestionarlo es el SHELL. El SHELL se encarga de asignar cada número de descriptor al archivo que corresponda.

Los caracteres de redireccionamiento son los siguientes :

> **[fichero]** *Creo fichero y lo lleno con la salida.*

>> **[fichero]** *Añade la salida al archivo.*

< **[fichero]** *Toma la entrada del fichero.*

- **[fichero]** *Error en un fichero nuevo.*

2>> [fichero] *Añade la salida de error a un fichero.*

Ejercicio : *Si el comando who me indica quien está conectado y el comando sort ordena un fichero; ¿Cómo obtendríamos la lista ordenada de los usuarios conectados?*

UNIX permite también modificar de una sola vez el descriptor de salida de esa orden y encauzarlo hacia el descriptor de entrada de otra para ello se utiliza el `pipe' ` | '.

Encauzo una orden con otra mediante esta tubería. La tubería crea un archivo temporal que sigue un esquema y sobre el que se garantiza la exclusión mutua.

Ejercicio : *Sacar por pantalla el numero de archivos que contiene un directorio.*

wc −l Cuenta el numero de líneas que escribe.

ls | wc −l Encauzo un listado hacia un contar líneas.

4.10 ESTRUCTURA DE LA LÍNEA DE ORDENES

La sintaxis general de una orden UNIX es :

<comando> [<opciones>] [<fichero_entrada>] [<fichero_salida>]

−letra ó +letra

Muchas ordenes UNIX no admiten ficheros de entrada o/y salida y todos escriben el fichero de entrada o de salida.

Se pueden redireccionar descriptores también con esto. (Todos menos el de errores).

4.11 ESTRUCTURA JERÁRQUICA DE FICHEROS

Como ya hemos dicho, el sistema de directorios de UNIX es de la forma grafo acíclico. Dentro del sistema de ficheros hay dos formas de referenciar un archivo. Llamamos nombre_absoluto al nombre único que se forma siguiendo la estructura de directorios desde la raíz hasta el archivo.

Llamamos nombre_relativo al recorrido desde el directorio actual.

.. Directorio superior

. Directorio actual

Para cambiar directorios utilizo la orden CD (cd) <directorio> tiene que existir un espacio entre ***cd*** *y el nombre del directorio..*

Para ver el directorio actual : PWD (pwd)

Dentro del tipo de fichero hay un tipo que se llama fichero tubería que es cuando el usuario gestiona el fichero temporal creado para comunicación en una tubería.

4.12 MONTAJE DEL SISTEMA DE FICHEROS

La estructura de directorios de UNIX gestiona los directorios de todos los sistema de ficheros en un único grafo acíclico (Algo del estilo de Windows 95)

El usuario cuando accede a un fichero no tiene constancia de la unidad física del sistema de ficheros físico donde se almacena ese archivo. La ruta de acceso es igual para archivos de todos los sistema de ficheros existentes.

En Windows 95 se conoce aún tratándose como un árbol, dónde se encuentra un fichero.

UNIX posee un sistema de ficheros raíz (root file system) que se corresponde con la unidad principal del sistema. Cuando desea añadir otro sistema de ficheros (administrador) selecciono un directorio dentro de ese sistema de ficheros raíz y sobre él monto mediante MOUNT(mount) el árbol de directorios del nuevo sistema de ficheros. Se superpone la raíz del nuevo árbol sobre el directorio seleccionado.

Se puede seleccionar un sistema de ficheros y montar algo sobre él.

Cuando se monta un sistema de ficheros sobre un directorio, los accesos al directorio se redirigen hacia el nuevo sistema de ficheros con lo que el directorio anterior queda oculto. Por ello es conveniente que el subdirectorio elegido esté vacío.

Para desmontar un sistema de ficheros se usa la orden UNMOUNT(unmount), la cual solo la puede utilizar el administrador.

No sabemos a que fichero estamos accediendo, para saber los ficheros y directorios a los que estamos accediendo utilizamos la orden UNMOUNT(unmount) sin parámetros.

4.13 ÁRBOL DE DIRECTORIOS DE UNIX

/

VAR DEV

USR

ETC HOME SPOOL TMP

BIN LIB SBIN INCLUDE SHARE

MAN

/ var *Contiene archivos que varían entre las diferentes versiones de UNIX y archivos cuyo tamaño cambia en función del instante. Ej. Archivo de correo, archivos temporales de usuario y archivos de registro y contabilidad (Contenidos en /mail, /tmp y /adm respectivamente).*

/ dev *Contiene los archivos especiales o de dispositivos, control de disco, impresoras, terminales...etc.*

/ etc *Contiene archivos de administración y bases de datos de configuración. Son archivos de texto, Ej. passwd, shadow..etc.*

/ home *A partir de este directorio, se encuentran los subárboles de usuario ; los directorios de conexión. En algunos sistemas se llama / users. Nosotros estaremos en / home / alumnos / FM−23 / 951081*

/ spool *Archivos temporales de gestión del sistema, Backup, impresión ...etc.*

/ tmp *Archivos temporales que necesitan las ordenes de UNIX. Los que se crean con tuberías (|)...etc.*

/usr *Contiene los directorios globales que utiliza el usuario en la máquina. Es todo lo que hemos visto hasta ahora /home. (Cualquier editor suele crear un temporal).*

/ usr / bin *Contiene los comandos del sistema y utilidades.*

/usr/sbin *Contiene comandos del sistema destinados a administración ; para el susario son solo de consulta.*

/ usr / lib *Bibliotecas de lenguajes programación.*

/usr/include*Contiene archivos de cabecera del lenguaje de programación C.(.h)*

/usr/share/man *Contiene los ficheros del manual.*

5. CONTROL DE PROCESOS

El núcleo del sistema operativo UNIX conoce la existencia de un proceso a través de su bloque de control de proceso, donde se describe el proceso y su entorno, constituyendo un contexto consistente en:

- ***Espacio de direccionamiento y entorno de ejecución:*** *Variables que utiliza el proceso*
- ***Contenido de los registros de hardware:*** *Contador de programa, registro de estado del procesador, puntero de la pila y registros de propósito general.*
- *Contenido de las estructuras del núcleo relacionadas con el proceso: Tablas de proceso, áreas, regiones, etc.*

Si congelamos el estado del procesador y del proceso que esta en ejecución en un determinado momento, obtendríamos lo que se conoce como imagen estática del programa. En caso de producirse una interrupción o cambio en el proceso, se almacena la imagen del que esta en ejecución en ese mismo instante.

Cada proceso se reconoce dentro del sistema por un numero que lo identifica unívocamente y que se conoce como INDENTIFICADOR DEL PROCESO (PID).

Todos los procesos excepto el proceso 0, son creados por otro proceso , es decir, el sistema de creación y gestión de procesos en el sistema operativo UNIX es jerárquico.

El proceso que se genera con el PID 0 es un proceso especial creado en el momento de arrancar el sistema. A continuación se genera un proceso INIT que será el antecesor de todos los procesos que se generen en el sistema. A partir de aquí se generan todos los procesos como terminales existan.

Los procesos que atienden a los terminales crearan otros con el fin de identificar y controlar el acceso de los usuarios al sistema, los cuales , una vez que inicien la sesión, ejecutaran un proceso interprete de comandos Shell que será el que genere el resto de los programas solicitados por el usuario.

Un proceso en UNIX es una instancia de programa en ejecución.

Cuando ejecutamos un programa, creamos uno o varios procesos.

Un proceso son varias estructuras de datos.

Desde un punto de vista más a bajo nivel un proceso es el resultado de la ejecución de una llamada al sistema; La llamada al sistema ***fork.*** *Cuando un proceso ejecuta la llamada fork, el sistema operativo crea una copia de la memoria virtual del proceso llamador. Se crea un clónico del primero.*

La diferencia entre las dos copias es el PID de cada proceso. Por esto es un proceso distinto.

En UNIX se establece una metáfora al considerar los procesos como seres vivos. Un proceso nace, muere, tiene hijos ..etc.

5.1 PRIORIDADES DE PROCESOS

El núcleo de UNIX reparte entre todos los procesos recursos, según sea el sistema de prioridades. Este es el que manda en los recursos. La prioridad de un recurso la determina el administrador en función del

propietario del mismo y el usuario normal solo puede influir ligeramente a este valor de prioridad. La prioridad de un proceso sigue una formula así:

PRIORIDAD = PRIORIDAD BASE + VALOR NICE

La prioridad base es la que viene marcada por el tipo de usuario. El valor nice es el valor modificador que puede emplear el usuario.

El usuario puede asignar valores nice desde −1 hasta −19 mediante la orden nice seguida del valor.

La orden nice solo permite disminuir la prioridad de un proceso con lo que aumenta el tiempo de CPU destinado a los demás procesos.

Si se pudiese aumentar la prioridad, sería un caos.

El administrador puede utilizar valores nice positivos poniendo dos signos menos seguidos.

Ej:

nice pr1 # −10 Usuario y administrador.

nice pr1 # − − 10 Solo administrador.

5.2 ORDENES DE TRATAMIENTO DE PROCESOS

Cuando nosotros le damos una orden al SHELL, este se encarga de ejecutarla.

Un proceso puede esperar la llegada de una señal de terminación de hijos en ejecución mediante la llamada al sistema wait. UNIX ofrece un comando wait que permite a un guión del SHELL realizar una espera equivalente a la llamada al sistema.

Para mantener información acerca de los procesos hacemos uso de la orden PS. Trabaja con el PID de procesos. El sistema operativo asigna los PID de proceso de forma consecutiva e incremental y cuando llegue al último número válido reutiliza los números de procesos que han dejado de existir. El único proceso que no carga es el 0.

Un modo sencillo de afectar a la planificación consiste en dormir un proceso durante un intervalo de tiempo. La orden ***sleep*** *recibe un parámetro, y luego duerme el proceso que lo ejecuta durante ese tiempo. Lo que hace es mandar una llamada al sistema para que se duerman. Para un proceso significa ayudar a los demás.*

Sleep permite dormir un proceso pero pierde precisión en intervalos de tiempo largo.

5.3 PROCESOS. MANEJO DEL PROCESADOR

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

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

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

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

Cuando se invoca una función del sistema, el proceso de usuario llama al Kernel como subrutina. Hay un cambio de ambientes y, como resultado, se tiene un proceso del sistema. Estos dos procesos son dos fases del mismo original, que nunca se ejecutan en forma simultánea.

Existe una tabla de procesos que contiene una entrada por cada uno de ellos con los datos que requiere el sistema:

identificación, direcciones de los segmentos que emplea en la memoria, información que necesita el scheduler y otros. la entrada de la tabla de procesos se asigna cuando se crea el proceso y se libera cuando éste termina.

Para crear un proceso se requiere la inicialización de una entrada en la tabla, así como la creación de segmentos de texto y de datos. Además, es necesario modificar la tabla cuando cambia el estado del proceso o cuando recibe un mensaje de otro (para sincronización, por ejemplo). Cuando un proceso termina, su entrada en la tabla se libera y queda otro disponible para que otro nuevo la utilice.

En el sistema operativo UNIX los procesos pueden comunicarse internamente entre sí, mediante el envío de mensajes o señales. El mecanismo conocido como interconexión (pipe) crea un canal entre dos procesos mediante una llamada a una rutina del Kernel, y se emplea tanto para pasar datos unidireccionalmente entre las imágenes de ambos, como para sincronizarlos, ya que si un proceso intenta escribir en un pipe ocupado, debe esperar a que el receptor lea los datos pendientes. Lo mismo ocurre en el caso de una lectura de datos inexistentes: el proceso que intenta leer debe esperar a que el proceso productor deposite los datos en el canal de intercomunicación.

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

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

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

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

En un sistema de tiempo compartido se divide el tiempo en un determinado número de intervalos o fracciones y se asigna cada una de ellas a un proceso. Además UNIX toma en consideración que hay procesos en espera de una operación de E/S y que ya no pueden aprovechar su fracción. Para asegurar una distribución

adecuada del procesador entre los procesos se calculan dinámicamente las prioridades de estos últimos, con el fin de determinar cuál será el proceso que se ejecutará cuando se suspenda el proceso activo actual.

6. GESTION Y MANEJO DE MEMORIA

La gestión de memoria en el sistema operativo UNIX se basa en el intercambio (swapping) y paginación. La paginación de la memoria se lleva a cabo si el hardware de la computadora la soporta. La política de carga y descarga de un proceso en la memoria depende del tiempo que lleve en la misma, de su actividad y del tamaño.

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

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

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

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

Hay una pregunta que surge entonces es ¿cuál de los posibles procesos que están cargados será desactivado y cambiado a memoria secundaria? Los procesos que se eligen primero son aquellos que están esperando operaciones lentas (E/S), o que llevan cierto tiempo sin haberse movido al disco. La idea es tratar de repartir en forma equitativa las oportunidades de ejecución entre todos los procesos, tomando en cuenta sus historias recientes y sus patrones de ejecución.

Otra pregunta es ¿cuál de todos los procesos que están en el disco será traído a memoria principal?. La decisión se toma con base en el tiempo de residencia en memoria secundaria. El proceso más antiguo es el que se llama primero, con una pequeña penalización para los grandes.

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

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

6.1 MANEJO DE ENTRADAS Y SALIDAS

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

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

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

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

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

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

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

6.2 LENGUAJE DE CONTROL DEL SISTEMA OPERATIVO

Entre los rasgos distintivos de UNIX está el lenguaje de control que emplea, llamado Shell. Es importante analizar dos funciones más de Shell, llamadas redireccionamiento e Interconexión.

Asociado con cada proceso hay un conjunto de descriptores de archivo numerados 0, I y 2, que se utilizan para todas las transacciones entre los procesos y el sistema operativo. El descriptor de archivo 0 se conoce como la entrada estándar; el descriptor de archivo 1, como la salida estándar, y el descriptor 2, como el error estándar. En general, todos están asociados con la terminal de vídeo, pero, debido a que inicialmente son establecidos por Shell, es posible reasignarlos.

Una parte de la orden que comience con el símbolo ? se considera como el nombre del archivo que será abierto por Shell y que se asociará con la entrada estándar; en su ausencia, la entrada estándar se asigna a

la terminal. En forma similar, un archivo cuyo nombre está precedido por el símbolo > recibe la salida estándar de las operaciones.

Cuando Shell interpreta la orden

califica < examen > resulta

llama a ejecución al programa califica (que ya debe estar compilado y listo para ejecutar) y detecta la existencia de un archivo que toma el lugar de la entrada estándar y de otro que reemplaza a la salida estándar. Después, pasa como datos de lectura los contenidos del archivo examen recién abierto (que debe existir previamente) al programa ejecutable. Conforme el programa produce datos como salida, éstos se guardan en el archivo resulta que Shell crea en ese momento.

*En la teoría de lenguajes formales desempeñan un importante papel las gramáticas llamadas de tipo 3 (también conocidas como regulares), que tienen múltiples aplicaciones en el manejo de lenguajes. Existen unas construcciones gramaticales conocidas como expresiones regulares, con las que se puede hacer referencia a un conjunto ilimitado de nombres con estructura lexicográfica similar; esto lo aprovecha Shell para dar al usuario facilidades expresivas adicionales en el manejo de los nombres de los archivos. Así, por ejemplo, el nombre carta * se refiere a todos los archivos que comiencen con el prefijo carta* y que sean seguidos por cualquier subcadena, incluyendo la cadena vacía; por ello, si se incluye el nombre carta* en alguna orden, Shell la aplicará a los archivos carta, carta1, carta2 y cualquier otro que cumpla con esa especificación abreviada. En general, en lugares donde se emplea un nombre o una trayectoria, Shell permite utilizar una expresión regular que sirve como abreviatura para toda una familia de ellos, y automáticamente repite el pedido de atención para los componentes. Existen además otros caracteres especiales que Shell reconoce y emplea para el manejo de expresiones regulares, lo que proporciona al lenguaje de control de UNIX mayor potencia y capacidad expresiva.*

En UNIX existe también la posibilidad de ejecutar programas sin tener que atenderlos en forma interactiva, sino simulando paralelismo (es decir, atender de manera concurrente varios procesos de un mismo usuario). Esto se logra agregando el símbolo & al final de la línea en la que se escribe la orden de ejecución. Como resultado, Shell no espera que el proceso "hijo" termine de ejecutar (como haría normalmente), sino que regresa a atender al usuario inmediatamente después de haber creado el proceso asincrónico, simulando en esta forma el procesamiento por lotes (batch) Para cada uno de estos procesos Shell proporciona, además, el número de identificación, por lo que si fuera necesario el usuario podría cancelarlo posteriormente, o averiguar el avance de la ejecución.

La comunicación interna entre procesos (es decir, el envío de mensajes con los que los diversos procesos se sincronizan y coordinan) ocurre mediante el mecanismo de interconexiones (pipes) ya mencionado, que conecta la salida estándar de un programa a la entrada estándar de otro, como si fuera un conducto con dos extremos, cada uno de los cuales está conectado a su vez a un proceso distinto. Desde Shell puede emplearse este mecanismo con el símbolo | en la línea donde se escribe la orden de ejecución.

Así en el ejemplo:

(califica < tarea | sorte > lista) &

se emplean las características de interconexión, redireccionamiento y asincronía de procesos para lograr resultados difíciles de obtener en otros sistemas operativos. Aquí se pide que, en forma asincrónica (es decir, dejando que la terminal siga disponible para atender otras tareas del mismo usuario), se ejecute el programa califica para que lea los datos que requiere del archivo tareas; al terminar, se conectará con el proceso sort (es decir, pasará los resultados intermedios) para que continúe el procesamiento y se arreglen los resultados en orden alfabético; al final de todo esto, los resultados quedarán en el archivo lista.

Con esta otra orden, por ejemplo, se busca obtener todos los renglones que contengan las palabras "contrato" o "empleado" en los archivos en disco cuyos nombres comiencen con la letra "E" (lo cual se denota mediante una expresión regular). Para lograrlo, se hace uso de una función llamada egrep, especial para el manejo de patrones y combinaciones de expresiones regulares dentro de los archivos:

*egrep−n 'contrato' 'empleado' E **

Los resultados aparecen así:

Emple1: 5: en caso de que un empleado decide hacer uso de la facilidad,

Emple1:7: y el contrato así lo considere las obligaciones de la

Emple2:9: Cláusula II: El contrato colectivo de trabajo

Emple2:15: Fracción llI: El empleado tendrá derecho, de acuerdo con lo

El tercer renglón, por ejemplo, muestra el noveno renglón del archivo Emple2, que contiene una de las palabras buscadas.

Como UNIX fue diseñado para servir de entorno en las labores de diseño y producción de programas, ofrece − además de su filosofía misma − un rico conjunto de herramientas para la creación de sistemas complejos, entre las que destaca el subsistema make. Este último ofrece una especie de lenguaje muy sencillo, con el cual el programador describe las relaciones estructurales entre los módulos que configuran un sistema completo, para que de ahí en adelante make se encargue de mantener el sistema siempre al día. Es decir, si se modifica algún módulo, se reemplaza o se añade otro, las compilaciones individuales, así como las cargas y ligas a que haya lugar, serán realizadas en forma automática, por esta herramienta. Con una sola orden, entonces, es posible efectuar decenas de compilaciones y ligas predefinidas entre módulos, y asegurarse de que en todo momento se tiene la última versión de un sistema, ya que también se lleva cuenta automática de las fechas de creación, modificación y compilación de los diversos módulos. De esta manera, se convierte en una herramienta casi indispensable al desarrollar aplicaciones que requieren decenas de programas que interactúan entre sí o que mantienen relaciones jerárquicas.

Otras herramientas interesantes son ar, diseñado para crear y mantener bibliotecas de programas (que serán luego utilizadas por otros programas para efectuar las funciones ya definidas sin tener que duplicar el código); awk, un lenguaje para reconocimiento de patrones y expresiones regulares (es decir, generadas por una gramática regular o de tipo 3), útil para extraer información de archivos en forma selectiva; lex, un generador de analizadores lexicográfico, y yacc, un compilador de compiladores. Estos dos últimos se emplean como herramientas en la creación de compiladores y procesadores de lenguajes.

La lista complete de funciones, órdenes de subsistemas que forman parte de las utilerías del sistema operativo Unix es realmente grande, e incluye más de un centenar, que se pueden agrupar en los siguientes rubros:

Compiladores de compiladores.

Ejecución de programas.

Facilidades de comunicaciones.

Funciones para control de status.

Funciones para control de usuarios.

Funciones para impresión.

Herramientas de desarrollo de programación.

Lenguaje C, funciones y bibliotecas asociados.

Macroprocesamiento.

Manejo de directorios y archivos.

Manejo de gráficas.

Manejo de información.

Manejo de terminales.

Mantenimiento y respaldos.

Otros lenguajes algorítmicos integrados.

Preparación de documentos.

6.3 ESTRUCTURA DE LA LINEA DE COMANDOS

Una vez iniciada de una sesion UNIX y estando presente el prompt $, el interprete de comandos Shell esta preparado para recibir un comando, cuya estructura es la siguiente

Nombre: Nombre del comando

Calificador: Posibles variaciones de actuacion del comando

Argumentos: Nombre del elemento (archivo, directorio...) sobreel que se quiere aplicar el comando.

Terminador: Delimitador que sirve para separar comandos (;)

Cc = comando

−0 = calificador

ejemplo.c = argumento

; = separador

who = otro comando

6.4 SISTEMA DE FICHEROS

- **DESDE EL PUNTO DE VISTA DEL SISTEMA OPERATIVO UNIX**

El sistema de ficheros interacciona con los procesos de usuario y con el gestor de memoria para realizar su labor.

El diseño de un sistema de ficheros requiere el planteamiento de lassiguientes partes :

- *Intercambio de mensajes entre los procesos implicados.*
- *Organización interna del sistema de ficheros. Cachos de bolques para mejorar las prestaciones y descripción del modo de acceso a ficheros.*

INTERCAMBIO DE MENSAJES :

Una petición comienza con una solicitud por parte del proceso de usuario (open / close ..etc) cualquier tipo de llamada que accede al sistema.

Llega un petición, el sistema de ficheros se comunica con el núcleo para ver como es la petición, el núcleo con el sistema de ficheros para devolver el resultado, el sistema de ficheros con el gestor de memoria y el gestor de memoria de nuevo al sistema de ficheros... un tratamiento complejo en el que intervienen hasta 29 procesos diferentes.

El sistema de ficheros realiza un bucle de espera de mensajes, y cuando se produce alguna solicitud comienza el tratamiento de las mismas.

6.5 ORGANIZACIÓN DEL SISTEMA DE FICHEROS

El sistema de ficheros está formado por bloques de organización, mapas de bits, nodos−i, directorios y bloques de datos. Estos elementos se organizan según el siguiente esquema :

AUTO ARRANQUE 1 BLK	SUPER BLOQUE 1 BLK	Mapa de bits de nodos−i N BLK´S	NODOS−I N BLK´S	BLOQUES DE DATOS N BLK´S

Autoarranque *El bloque de autoarranque contiene software encargado de cargar el sistema operativo en memoria y empezar a ejecutarlo ; este bloque es al que accede la ROM del ordenador para comenzar a funcionar.*

Super Bloque *Contiene una descripción de la estructura del sistema de ficheros. Sus campos son :*

- *nº de nodos−i.*
- *nº de zonas de datos.* −−>[Author:FCCÖ˜z]
- *nº de bloque del mapa de nodos−i.*
- *Posición de la primera zona de datos.*
- *Máximo tamaño de fichero(nº mágico).*
- *Punteros al bolque del mapa de nodos−i.*
- *nº dispositivo superbloque.*
- *nodo−i del sistema de ficheros montado.*
- *nodo−i del directorio en que se monta.*
- *Última actualización.*
- *Si es de solo lectura.*

Mapa de bits (n−i) *Un nodo−i representa un fichero en UNIX. Existe uno por cada archivo. El mapa de bits de nodos−i, indica si la entrada para ese archivo está libre u ocupada. Cuando creo un archivo se busca un*

nodo−i cuyo bit esté a 0 y se le asigna ese archivo.

El borrado de archivos se hace de forma lógica poniendo a 0 el bit del nodo−i del archivo que acabamos de borrar.

Nodos−i *Hay un nodo−i por archivo, y puesto que el número de nodos−i es limitado, también lo es el de archivos. Puedo no llenar el disco, pero quedarme sin nodos−i, y puedo tener mas nodos−i de lo que permite mi disco. Cada nodo−i tiene una estructura de 64 bytes de 16 bits cada uno ; de tal forma que tendremos una estructura de 16x64 bits.*

Los campos contienen :

- *Modo.(Tipo de fichero y bit de protección)*
- *Uid. (Identificador de usuario)*
- *Tamaño.(En bytes)*
- *Última modificación.(nº de dias transcurridos desde 1/1/1970)*
- *Enlaces | GID(Enlaces a directorios que contienen nodo−i)*
- *nº de zona 0*
- *...*
- *nº de zona 9*
- *Puntero indirecto. Punteros a zonas de datos.*
- *Puntero indirecto doble.*
- *Puntero indirecto triple.*

6.6 CACHÉ DE BLOQUES

UNIX mantiene una caché con los últimos bloques accedidos que se organizan en forma de tabla hash y las colisiones se tratan mediante listas doblemente enlazadas.

La salida de bloques de la caché sigue un algoritmo LRU para cada entrada de la tabla hash.

Los bloques de la caché se escriben en disco en dos momentos :

- *Se escriben cuando son expulsador por la CPU.*
- *Cuando se ejecuta la orden SYNC(sync) que copia todos los bloques de la caché a disco. La orden sync se ejecuta aproximadamente cada 5 segundos (o más).*

Existen bloques especiales que se copian a disco directamente.

6.7 DESCRIPCIÓN DEL MODO DE ACCESO A FICHEROS

¿Cómo se realiza una búsqueda de un fichero en una ruta ?

Accederemos a los nodos−i de los directorios existentes en la ruta y a los bloques de datos que contienen información sobre cada directorio.

Un directorio en sus bloques de datos contiene una lista de entradas de 16 bytes, de los cuales 2 corresponden al número de nodo−i, y los otros 14 al nombre de fichero correspondiente a ese nodo−i ; cada entrada de la lista se corresponde con un fichero contenido en el directorio.

Ej. Para acceder a /users/practicas.txt

En cualquier acceso se comienza leyendo el nodo−i número 1 que se corresponde con el directorio origen del árbol de directorios. En este nodo−i seleccionamos el puntero a zona de datos que nos indica la zona donde está la información del directorio.

Una zona tiene como mínimo 1024 bytes, si cada entrada es de 16 bytes, nos salen 64 entradas de directorio en cada bloque o zona. Los directorios que tengan menos de 64 fichero ocuparán un bloque(zona).

Supongamos que 120 es un bloque de datos de directorios :

120

Para acceder a los archivos, accedemos a los nodos−i.

Para saber si una zona de datos es un directorio o un fichero hay que ver el modo en el nodo−i.

7. INTÉRPRETE DE COMANDOS

7.1 EL SHELL DE PRESENTACIÓN

Se conoce como Shell al programa que interpreta las peticiones del usuario para ejecutar comando, utilidades o programas, es decir, es el interprete de comandos del sistema operativo UNIX y es la Interface entre el usuario y el sistema.

El Shell es un programa del sistema operativo, pero no forma parte del núcleo del mismo. Se ejecuta cada vez que el usuario se identifica ante el sistema y comienza una sesión.

Es también un lenguaje de programación que soporta dadas las estructuras propias de los lenguajes modernos. Además permite la utilización de todas las primitivas del sistema operativo de control de procesos, interrupciones y utilidades para diseñar programas de comandos por el usuario.

Existen varios tipos de Shell con diferentes características:

- *Bourne Shell: es el interprete de comandos básico*
- *C − Shell : Es el interprete de comandos creado por Berkeley para el sistema operativo BSD y ara eñ XENIX, un poco mas completo que el anterior. Su programación es prácticamente lenguaje C.*
- *Korn Shell: Se basa en los dos anteriores, siendo compatible con el Bourne en un 95%. Añade posibilidades de programación avanzada, facilidades aritméticas y mayor rapidez en la ejecución.*

Veremos mas adelante un desarrollo puntual de cada uno de los tipos de Shell mencionados.

El SHELL de UNIX lo podemos ver desde dos puntos de vista diferentes:

- **Intérprete de ordenes:**
  - *Lee la línea de ordenes(desde el prompt hasta el RTM)*
  - *Evalúa los posibles caracteres especiales que contiene, dispone de lo necesario para la ejecución de los procesos de la línea de ordenes; de tal forma que el SHELL no es el encargado de ejecutarlas.*
- **Lenguaje de programación:**

Ofrece estructuras de alto nivel para crear programas que llamaremos guiones del SHELL (SHELL scripts) que indican una serie de pasos para realizar una labor.

Cuando nos presentamos en el sistema UNIX se inicia automáticamente el SHELL incluido en `/ etc / passwd' para el usuario que se conecta; este SHELL es el SHELL de presentación, se crea como hijo del proceso ***login*** *y como padre de todos los demás procesos que ejecute el usuario. El SHELL de presentación no termina hasta que el usuario se despide y su terminación provoca el fin de la sesión.*

Nosotros estamos conectados mientras exista el SHELL de presentación.

Una vez que el usuario está conectado, la mayor parte de las interacciones con el sistema toman la forma de dialogo con el SHELL. Este diálogo sigue una secuencia simple y repetitiva :

- *El SHELL presenta un prompt de solicitud de ordenes.*
- *El usuario introduce una orden por teclado.*
- *El SHELL evalúa la orden e indica al núcleo los procesos a crear.*
- *El SHELL espera a que terminen todos los procesos creados y vuelve a 1.*

1 2

4 3

En general la línea de ordenes contiene nombre, argumento y un RTM el SHELL no comienza a procesar la orden hasta que recibe RTM. Si es necesario se pueden realizar varias ordenes con un solo RTM separador por un ` ; '.

7.2 TIPOS DE SHELL

El UNIX sistema V versión 4, proporciona tres tipos de SHELL de presentación, que son :

- *Standar SHELL (Bourne SHELL) / bin / sh*
- *SHELL C / bin / csh*
- *SHELL de Korn / bin / ksh*

El csh y el ksh se desarrollaron para añadir capacidades adicionales al SHELL estándar, entre ellos la edición de la línea de ordenes, los alias de ordenes y el histórico de comandos.

***El SHELL C** fue desarrollado por Bill Joy como una ampliación al sistema UNIX de Berkley. Proporciona todas las características del SHELL estándar y un amplio numero de extensiones. Su sintaxis se ajusta al lenguaje de programación C y es bastante diferente de la del SHELL estándar. Resulta muy cómodo para programar en C. Es incompatible con el SHELL estándar.*

7.3 SHELL DE KORN

El Korn SHELL fue desarrollado por David Korn en 1982 en los laboratorios Bell , incorpora la mayor parte de las ampliaciones del SHELL C y conserva la sintaxis del SHELL estándar. Amplia el SHELL estándar para mantenerlo la misma sintaxis. Los programas de SHELL estándar funcionasen en Korn. Lo del Korn no funcionan en Korn (debido a ampliaciones).

7.4 FICHEROS DE INICIALIZACIÓN

Cuando se inicia el SHELL de presentación, tanto en SHELL estándar como en SHELL de Korn, se busca un

archivo llamado .profile en el directorio de conexión. Este archivo es un fichero de texto que funciona como un guión del SHELL y por tanto contiene instrucciones que se ejecutarán en el momento de la conexión. Los guiones del SHELL son ficheros de texto. El SHELL ejecuta ese guión antes de hacer cualquier cosa. Tiene sentencias de información acerca del sistema y de personalización de variables de entorno.

Nuestra terminal va a ser vt220.

El SHELL indica que está listo para recibir la entrada, visualizando el prompt. Por defecto el prompt principal del SHELL es $ (# root).

El SHELL de korn utiliza un archivo adicional al .profile. Este archivo es le que se encuentra en la variable de entorno ENV y se ejecutan después del .profile. Se da el valor a ENV en el .profile. El nombre más habitual es .kshrc ; los fichero que acaban en rc son de configuración. El punto es un archivo de limitación de acceso. El .kshrc es otro fichero de texto.

El fichero de configuración .profile se ejecuta únicamente al iniciar la sesion. El fichero secundario se ejecuta cada vez que se inicializa el SHELL de presentación. Determinados programas tienen la capacidad de generar un escape del sistema con lo que anulan el SHELL de presentación y lo reinicializan al terminar. Ej. editor VI

En esta inicialización es cuando se utiliza el .kshrc pero no el .profile.

7.5 VARIABLES DEL SHELL

El SHELL dispone de un mecanismo para definir variables que pueden contener información para otros procesos o para si mismo. Ej. TER lo utilizarán muchos procesos. Se pueden definirán conjunto de variables estándar y también unas variables personales para almacenar cualquier tipo de información.

La variables del SHELL estándar más comunes son :

- *HOME : (Nombre del camino absoluto hasta el directorio de conexión del usuario). Se define automáticamente a partir de ../passwd y tanto el SHELL como otros ficheros pueden hacer usos de ellos (no modificarlo).*
- *PATH : Contiene en orden los directorios en los que el SHELL busca una orden a utilizar , cada nombre de directorio está separado del siguiente mediante dos puntos (:) y el directorio actual se representa con una posición vacía. UNIX busca los archivos en los directorios que vienen en el path en orden (si el directorio actual no se pone en el path, el SHELL no busca en él).*
- *CDPATH : Lista en orden los directorios en los que busca el SHELL para cambiar de directorio con la orden CD(cd) (como un PATH, pero para cambiar de directorio).*
- *PS1, PS2 : Definen respectivamente los prompt principal y secundario ; por defecto PS1 $ y PS2 >*
- *LOGNAME : Contienen el nombre de presentación del usuario.*
- *MAIL : Contiene el nombre del directorio donde se introduce el correo nuevo para el usuario.*
- *SHELL : Contiene el nombre del SHELL de presentación ; se usa para saber que SHELL actúa cuando se produce un escape del sistema.*
- *TERM : Contiene el nombre del terminal empleado y no se fija automáticamente (con los emuladores de terminal actuales a veces se da por defecto).*

Variables más comunes del KSHELL :

- *ENV : Contiene el nombre del fichero secundario de inicialización.*
- *EDITOR : Contiene el nombre del editor de línea de ordenes a emplear.*
- *HISTORY : Contiene el nombre del fichero de histórico de comandos.*

- *HISTSIZE : Contiene el tamaño dedicado al histórico de comandos.*
- *IFS : Contiene el carácter empleado como separador de campos.*

7.6 OBTENCIÓN DEL VALOR DE UNA VARIABLE

Para obtener el valor de una variable del SHELL debemos indicar el nombre dela variable y anteponer el símbolo $. Cuando el SHELL encuentra una palabra que comienza por $ supone que es una variable, y sustituye su valor por la parición de la variable.

Ej.

echo $LOGNAME Sustituye $LOGNAME por F951081 echo

ls $HOME sutituye $HOME por /home/alumnos/... ls

Esta forma de acceder a la variable es válida tanto para variables del sistema como para variables creadas por el usuario. Es decir, mediante este tipo de acceso podemos acceder a todos las variables predefinidas como creadas por el usuario.

Para visualizar todas las variables declaradas actualmente en el SHELL ejecutamos la orden SET(set).

7.7 DEFINICIÓN DE VARIABLES EN EL SHELL

Aunque existen variables cuyo valor se fija automáticamente hay otras a las que necesitamos asignar un valor. Para definir una variable, empleamos la siguiente sintaxis :

nombre=valor

Se puede cambiar el valor de una variable ya definida.

Si la variable no tiene un valor, la variable no existe. Esta es una orden de asignación y definición al mismo tiempo.

El valor puede contener más de una palabra, en esta caso se pondrá entre comillas. Entre el nombre, el valor y el signo =, no pueden existir espacios en blanco.

7.8 EXPORTACIÓN DE VARIABLES

Las variables propias o internas al SHELL se distinguen de las que se pueden utilizar en otros programas por el entorno al que pertenecen. En una sesión se crean tantos entornos distintos como procesos se estén ejecutando y cada entorno se divide en entorno propio y entorno global.

Las variables que pertenecen al entorno propio de un proceso solo son accesibles desde ese proceso, mientras que las de entorno global son accesibles desde ese proceso y desde todos los procesos hijos de ese.

Una variable propia del SHELL pasa al entorno cuando es exportada.

Para exportar una variable utilizamos:

export nom_var

Ej.

<Estamos en el SHELL de conexión>

P = 3

echo $P {Sale por pantalla un 3}

ksh {Ejecutamos otro entorno K SHELL}

P = 5

echo $P {Sale por pantalla un 5}

export P {Pasa la variable P al entorno del Ksh de ahora}

ksh {Ejecutamos otro entorno K SHELL}

echo $P {Sale por pantalla un 5}

P = 7

exit {Salgo del 2º Ksh}

echo $P {Sale por pantalla un 5}

{Las modificaciones no afectan al padre}

exit {Salgo del 1er Ksh}

echo $P {Sale por pantalla un 3}

ksh {Ejecuto de nuevo un Ksh}

echo $P {ERROR ! variable no declarada}

Existen variables del sistema que por defecto están exportadas (Pertenecen al entorno global).

Ej : HOME, LOGNAME, PS1, PS2...etc.

7.9 VARIABLES *NOCLOBBER* e *IGNOREEOF*

(EN MINÍSCULAS)

Tanto el Csh como el Ksh utilizan variables especiales llamadas de conmutación para activar o desactivar características. Las variables de conmutación solo pueden tener dos valores : Activado o desactivado. En el SHELL de Korn se gestionan estas variables mediante la orden set con el parámetro −o(activar) y +o(desactivar).

noclobber : *Cuando está activada impide sobreescribir en archivos existentes mediante redireccionamiento de salida. Para confirmar la sobreescritura, utilizamos el carácter |.*

Ej. ls −l > temp {temp existe} No lo hace ERROR.

ls −l >| temp {temp existe} Se fuerza la sobre escritura.

ignoreeof : *La pulsación de CTRL + D equivale a abortar la ejecución. En muchas ocasiones para abortar un proceso hay que pulsar esta combinación varias veces. Esta repetición de teclas puede hacer que además de abortar el proceso, nos salgamos del entorno SHELL de conxión(nos vamos fuera del sitema). La variable ignoreeof cuando está activahace que el SHELL de conexión ignore la pulsación de CTRL + D. Solo protege al SHELL de conexión.*

7.10 HISTÓRICO DE ÓRDENES

El Ksh mantiene un registro de todas las líneas de ordenes introducidas en una sesión (depende del tamaño que se le de a HISTSIZE). Este registro permite desarrollar varias características de este SHELL.

Visualización y ejecución :

Para ver en cualquier instante las últimas ordenes ejecutadas, empleamos la orden history ; también permite acceder a una orden en particular dándole como parámetro el código de esa línea de órdenes.

Se puede reejecutar una orden mediante el comando r, como parámetros admite tanto la orden a buscar, como el numero de línea en el que se encuentra la orden.(Sin parámetros ejecuta la última orden).

Ej.

r vi Retrocede hasta encontrar vi, y lo ejecuta.

El nombre de archivo que suele contener el histórico de comandos, es .sh_history

Edición de línea de ordenes :

El ksh, permite elegir un editor de línea de ordenes para modificar y reejecutar ordenes anteriores.

El editor elegido se encuentra en la variable EDITOR y debe ser asignado por el usuario.

Cuando está activo un editor de línea de ordenes podemos remplazar los comandos de ese editor para movernos por ese fichero histórico. El editor más usual es el VI. Otro importante es EMACS.

Cuando está activado el VI como editor disponemos de una ventana de edición de tamaño una línea sobre el fichero histórico.

7.11 ALIAS DE ÓRDENES

Un alias de una orden es una palabra que es sustituida por la orden cuando se usa como comando.

Es similar a una variable, pero usada como comando.

Un ALIAS se ejecuta.

Un ALIAS puede servir para dar otro nombre a ordenes existentes, para simplificar la escritura de ordenes o para sustituir a líneas de ordenes muy largas.

Un ALIAS puede hacer referencia solo a un comando, a un comando como parámetro, e incluso a una sucesión de comandos encadenados unos a otros.

Ej.

alias m = mailx

alias lsc = ls −lt

alias cuenta = who | wc−l

Para acceder a un ALIAS, basta con escribir su nombre y pulsar <ENTER>.

La vida de los alias alcanza hasta el final de la sesión.

Para que tengamos un alias desde que se inicia la sesión, hay que meterlo en el .profile.

7.12 EJECUCION DE ORDENES

7.12.1 EN MODO SUBORDINADO

Normalmente cuando el SHELL ejecuta una orden permanece bloqueado en espera de que finalice.

Durante este tiempo, no atiende peticiones del usuario.

En ocasiones la ejecución de una línea de ordenes se prolonga durante mucho tiempo y no necesita de la interacción con el usuario. En estos casos resulta útil que el SHELL esté dispuesto para recibir nuevas ordenes antes de que termine la orden anterior.

Esto se consigue haciendo que la ejecución de una orden se realice en modo subordinado:

<background>

Para ello al final de la línea de ordenes añado el símbolo `&'.

No recibe salida de teclado.

Hace que el SHELL no se pare.

7.13 LA E/S EN MODO SUBORDINADO:

Cuando se ejecuta una orden en modo subordinado, el SHELL la procesa iniciando los procesos necesarios, emite un mensaje de identificación formado por 2 números y a continuación solicita otra orden. Estos dos números representan identificador de trabajo y el PID del proceso. Desconecta la entrada estándar de la orden en modo subordinado del teclado. Una orden en modo subordinado, no puede aceptar ordenes de teclado), pero no desconecta la salida ni el error de la pantalla (cuando genere una salida va a ir a la pantalla). Esto a veces va a resultar molesto, ya que la salida de la orden subordinada se entremezcla con el echo del teclado sobre la pantalla. Resulta muy útil el archivo /d/null , puesto que todo lo que de error se graba en él.

7.14 MANTENIMIENTO DE TRABAJOS ACTIVOS.

Lo normal es que una ejecución en modo subordinado sea de operaciones largas.

Una operación muy larga debe continuar, incluso después de que el usuario haya abandonado el sistema.

En condiciones normales cuando un usuario se despide y mantiene trabajos activos, el núcleo elimina este trabajo, con lo cual los trabajos terminan. (Si yo me desconecto el trabajo se elimina)

Para mantener un trabajo activo una vez que la sesión haya terminado se debe ejecutar el trabajo precedido de la orden NOHUP(nohup) esto le indica al núcleo que el trabajo continua incluido después de acabar la sesión, nohup no me obliga a desconectar.

La orden nohup solo se ejecuta en modo subordinado. Todo lo que se ejecute con nohup lo toma como subordinado, la Entrada, la Salida y el Error, se han de direccionar ; si no se hace la Salida y el Error se almacenan en nohup.out.

7.15 ÓRDENES DE CONTROL DE TRABAJOS.

Para controlar procesos empleamos el PID de los mismos. Cualquier referencia a un PID identifica a un único proceso en el sistema.

Para visualizar los procesos existentes en el sistema utilizamos la orden PS(ps). PS ofrece información sobre que procesos hay, cual es su PID y cuales son sus otras características.

PID nos puede ofrecer todo : El espacio asignado en memoria, el terminal asociado, los padres e hijos del proceso...etc.

En el SHELL estándar, el único control que se puede establecer sobre un proceso, es su eliminación. La orden KILL(kill) envía un número de señal a un proceso. Cuando un proceso envía una señal para la que no está preparado, el proceso termina.

Cualquier tipo de señal puede eliminar un proceso.

Las señales para utilizar con Kill son :

- Interrupción : nº 3
- Terminar : nº 10
- Eliminación incondicional : nº 9

La diferencia está en la fuerza.

1 Cuando hay algún error.

2 Para la finalización.

3 Finalización pase lo que pase Ningún proceso puede controlarlo. Si yo quiero que un proceso acabe siempre, le doy un nº 9.

Ej.

SHELL

señal 3 : La ignora

señal 10 : La ignora

señal 9 : El SHELL termina.

Cuando una señal acaba con el SHELL termina la sesión.

Los procesos que nosotros hagamos podrán capturar cualquier señal menos la 9.

7.16 EL SHELL DE TRABAJO

JSH

Es un SHELL destinado para gestionar trabajos de usuario. Nos permite un control más sencillo y más flexible que el SHELL estándar.

Todas las características del Jsh están en el ksh.

Dentro del Jsh podemos referenciar a una orden de dos formas :

- *Por el PID.*
- *Mediante el numero de trabajo asignado a esa orden.*

El numero de trabajo es un número único para cada trabajo de un usuario.

Los números de trabajo no van a estar salteados como los PID(Son de un solo usuario).

Un trabajo es un conjunto de ordenes ejecutadas explícitamente por el usuario.

- *Un trabajo no tiene por que ser un proceso. (Puede ser también un conjunto de procesos).*
- *Ejecutamos más procesos que trabajos. (Los trabajos son los procesos o conjunto de procesos que yo he ordenado) .*

El JSH puede :

- *Obtener la lista de trabajos mediante la orden JOB(job).*
- *Pasar un trabajo de modo principal a modo subordinado. BG(bg).*
- *Pasar un trabajo modo subordinado a modo interactivo. FG (fg).*
- *Detener un trabajo. STOP(stop).*
- *Eliminar un trabajo. KILL(kill) Kill −9 <PID> kill %<nºtrabajo>*

8. HERRAMIENTAS DE DESARROLLO DE SOFWARE

El sistema operativo UNIX proporciona al usuario una serie de herramientas o programas de utilidad para el desarrollo de software. Estas herramientas son:

- ***Editor de pantalla (VI):** Tiene como misión principal la creación de archivos fuente.*
- ***CC:** Es el compilador del lenguaje C*
- ***LINT:** Es un analizador sintáctico de lenguaje C mas estricto que el compilador*
- ***AR:** Es un programa para mantener librerías o biblioteca de programas objeto o cualquier otro tipo de archivos.*
- ***MAKE:** Es un programa de utilidad de UNIX que acepta especificaciones de las dependencias existentes entre módulos de un programa y estable mecanismos para mantener las versiones del programa, trasladando al resto de módulos los cambios que se realicen en uno de ellos.*
- ***SCCS:** (Sistema de control de Código Fuente): Son un conjunto de utilidades para el desarrollo de*

aplicaciones con muchos módulos que se desarrollan en distintos equipos.

- ***AWK:*** *Es el interprete de un lenguaje de programación cuyo fin es tabular, dar formato y preprocesar archivos de datos.*
- ***LEX:*** *Es un programa de utilidad que sirve para crear un analizador léxico que basa su análisis en ciertas reglas de lenguaje definidas por el usuario.*
- ***YACC:*** *Es un analizador semántico cuya misión es transformar un archivo con instrucciones de un lenguaje especificado, que cumpla las reglas semánticas dadas, en un modulo compilable en C.*

8.1 LA PROGRAMACION DEL SHELL

8.1.1 GUIONES DEL SHELL

La palabra SHELL hace referencia a dos cosas, intérprete de comandos, y lenguaje de programación.

El lenguaje SHELL es un lenguaje de programación de alto nivel que permite generar ordenes de UNIX controlando el flujo a través de esas ordenes.

El SHELL solo controla el flujo a través de estas ordenes UNIX.

Los guiones del SHELL se utilizan para hacer referencia a un conjunto de ordenes que se ejecuten de la misma forma múltiples veces. Pero todo lo que va dentro de un guión se puede escribir directamente en la línea de ordenes.

Un guión es un fichero de texto.

8.1.2 EJECUCIÓN DE UN GUIÓN

Para hacer ejecutable un guión es necesario conceder autorización de ejecución sobre el archivo.(No hay que compilar).

Haciendo ejecutable un fichero puedo usarlo como una orden de la línea de ordenes. Este modo de ejecución provoca que el SHELL cree un subshell destinado a leer y ejecutar el contenido del guión.

Para hacer una ejecución se crea un SUBSHELL igual que el padre. Este SUBSHELL coge las líneas de una en una al fichero ejecutable.

Todo lo que hay en un guión está en la línea de ordenes. Esto mismo se puede conseguir ejecutando un proceso SHELL que precede como parámetro el nombre de un guión.

Ej

PR 1 ! Ha de tener permiso de ejecución.

Ksh PR1 ! No necesita permiso de ejecución.

A veces resulta útil ejecutar un conjunto de ordenes en un SUBSHELL sin necesidad de crear un guión.

Para ello debo poner la lista de ordenes separadas por [;] y entre paréntesis. Cualquier redireccionamiento para las listas entre paréntesis afecta a todas las ordenes de la lista.

Cuando ejecutamos guiones en un SUBSHELL, todas las variaciones del entorno que generen, afectan al SUBSHELL.

Cuando deseemos que un guión del SHELL cambie el entorno del SHELL en el que dicho guión se ejecuta y no del SUBSHELL, podemos ejecutarlo de las dos siguientes formas :

- *Agrupando las ordenes entre llaves (Afecta el SHELL actual)*
- *Mediante la orden . (dot) seguida de un nombre de guión el guión se ejecuta en el SHELL principal.*

Si ejecutamos el .profile : . .profile.

8.1.3 COMENTARIOS EN LOS GUIONES DEL SHELL

Para incluir un comentario en un guión del SHELL debo incluir el carácter # (almohadilla). El SHELL considera comentario todo lo que aparezca detrás de la almohadilla hasta el salto de línea. No puedo insertar comentarios dentro de una línea, sino que tengo que ponerlo al final de la línea.

8.2 PARÁMETROS POSICIONALES

Los guiones del SHELL son capaces de recibir parámetros en la llamada, para que el guión haga algo diferente.

Para usar parámetros, el SHELL utiliza unas pseudovariables llamadas parámetros posicionales cuyo valor se fija automáticamente al ejecutarse el guión.

Los más importantes son :

$# : Indica el numero de argumentos.

$1,$2,$n.. : Especifica 1º, 2º, enésimo argumento.

$0 : Especifica el nombre del programa.

$: Da la lista completa de argumentos sin el $0.*

Para reordenar los parámetros posicionales utilizo la orden SHIFT(shift). Desplazar hacia la izquierda los parámetros eliminando el primero y disminuyendo el número en uno.

Para asignar un valor a los parámetros posicionales de un guión utilizo la orden SET(set) seguida de una ejecución. Esto hace que la salida de la ejecución se asigne sobre los parámetros posicionales del guión.

En la secuencia de cadenas de ejecución de un guión interviene $? Que contiene el valor devuelto por la última sentencia de guión ejecutable.

8.3 OTRAS VARIABLES DEL GUIÓN

Un guión posee por defecto todas las variables externas del SHELL que lo ejecutó. Además podemos definir cualquier otra variable de la misma forma que el SHELL.

Esa variable desaparece cuando desaparece el guión.

Dentro de un guión podemos aplicar dos operadores adicionales para asignar valor a la variable en case de que esta no lo tenga.

Si lo que hacemos es una consulta la variable no declarada toma este valor por defecto.

Existen dos formas :

- *$ {nom_variable :− valor por defecto}*

Si la variable no tiene valor, devuelve el valor por defecto pero no se lo asigna}

- *${nom_variable := valor por defecto}*

Igual que :− pero si que se lo asigna.

Ej.

$0 = pr1

P=3

echo ${P :=7} devuelve 7

echo ${D :=5} devuelve 5

echo $D devuelve 5

echo ${E :−3} devuelve 3

echo $E devuelve no definido

8.4 SENTENCIAS DE CONTROL EN LA PROGRAMACIÓN SHELL

Dentro de la programación del SHELL voy a utilizar ordenes de la línea de comandos.

Operaciones lógicas :

Evaluación de expresiones lógicas : *Se realizan mediante la orden test (TEST). Test permite realizar comprobaciones sobre enteros, cadenas de caracteres y estado de ficheros. Si la comparación es cierta, test genera un código de retorno 0 y si es falsa, distinto de 0.*

*Los **test** admitidos son :*

- *Sobre enteros :*

n1 −eq n2 =

n1 −en n2 "

n1 −gt n2 >

n1 −ge n2 >=

n1 −lt n2 <

n1 −le n2 <=

- *Sobre cadenas :*

−z cad Si la longitud de la cadena es = 0.

−n cad Si existe la cadena.

cad1 = cad2 cad1 igual a cad2

cad1 != cad2 cad1 distinto de cad2.

Cadena Si la cadena es distinto de la cadena vacía.

- *Sobre ficheros* :

−a fichero Si existe el fichero.

−r fichero

−w fichero Permisos del fichero.

−x fichero

−f fichero Fichero ordinario.

−d fichero Directorio.

−h fichero Vínculo.

−c fichero Especial de carácter.

−b fichero Especial de bloque.

−p fichero Tubería.

−s fichero Tamaño > 0.

La orden test permite también concatenar expresiones lógicas mediante los operadores :

−a : AND

−o : OR

! : NOT

Ej.

test $# −gt 5 −a −w salida

Si el número de parámetros del guión es mayor que cincoy se puede escribir en el fichero salida

Una sintaxis alternativa para test consiste en situar la expresión lógica entre corchetes.

El SHELL de Korn, proporciona la orden [[ ]] que amplia el funcionamiento de test. Una expresión lógica entre doble corchete permite utilizar operadores clásicos y trata correctamente las variables nulas, es decir si una variable no es nula lo compara con la variable vacía.

8.5 ÓRDENES DE CONTROL

IF ...THEN

Sintaxis :

if orden **if** orden

then órdenes **then** órdenes

fi else órdenes

fi

El if ejecuta la orden de antes del then y si el resultado devualto es 0, ejecuta el then. Si el resultados es distinto de 0 o bien ejecuta el else, o nada.

CASE

Sintaxis :

case cadena

in

(patron/lista) orden

orden

; ;

(patron/lista) orden

orden

; ;

esac

Compara la cadena con cada caso, y si alguno coincide, ejecuta las ordenes asociadas. Cuando ejecuta las ordenes, sale del case.

(patron/lista) es un valor, o una lista de valores.

*Cualquier cadena con un asterisco(*) sería el else. Si se pone el *, se debe poner siempre al final.*

SELECT

La sentencia select permite comparar una cadena con un conjunto de casos, pero obliga a que la cadena se corresponda con algún caso.

Por tanto lee un valor de la entrada estándar y si no hay concordancia continua solicitando hasta que lo haya.

FOR

Sintaxis :

for variable

in lista

do

órdenes

done

Realiza una iteración para cada elemento de la lista, asignándoselo como valor a la variable.

Si no hay una lista, itera sobre los parámetros posicionales.

WHILE

Sintaxis :

while orden

do

órdenes

done

UNTIL

Sintaxis :

until orden

do

órdenes

done

Las ordenes de interrupción de bucle : Si se desea interrumpir un bucle independientemente de la condición el SHELL ofrece dos órdenes :

- ***Break :*** *Finaliza la interación y continúa con la siguiente instrucción.*
- ***Continue :*** *Finaliza la iteración y vuelve a la condición o comparación.*

Las dos permiten un parámetro

Órdenes true y false :

Son 2 órdenes constantes que devuelven un valor 0 y distinto de 0 respectivamente.

8.6 OPERACIONES ARITMÉTICAS

*El SHELL estándar permite realizar operaciones aritméticas mediante la orden **expr**.*

*La orden **expr** recibe tres parámetros (dos operadores y un operador) y muestra en la salida estándar el resultado.*

*Para realizar operaciones más complejas se hace uso del operador **grave** también llamado restitución de ordenes y que se representa con dos comillas invertidas. El operador grave ejecuta el comando entre comillas y sitúa la salida del mismo en lugar del comando. Permite que la salida de una orden forme parte de la línea de órdenes.*

Ej.

expr `expr 1+2`+1

El Kshell facilita las operaciones aritméticas mediante la orden let (LET). Let permite operar con variables directamente, sin poner $, utilizan más de un operador por línea, realizan operaciones más complejas y realizan comparaciones entre enteros. Una alternativa del let es (( ))

Ej.

let x = 2*y%z " x = `expr ` expr 2*y `opr $z`

8.7 OTRAS ÓRDENES DEL SHELL

Órdenes de E/S :

read *Lee una línea de la entrada estándar y se la asigna a una o más variables de tal forma que la primera palabra se asigna a la primera variable, la segunda a la segunda,. Y el resto de línea a la ultima variable.*

El SHELL de korn permite combinar una petición de entrada con la lectura de variable.

Ej.

read A B ? Entrada :

print *Sustituye en el SHELL de korn el echo y permite ampliar los formatos de este.*

8.8 Otras órdenes :

trap *La orden trap permite especificar una secuencia de acciones a realizar cuando se recibe una señal :*

Sintaxis : trap <conjunto de ordenes> <números de señal>

Ej. trap rm tmp$$ 2 3 15

Cuando llegue la señal 2, 3, 15 se ejecutarán las órdenes.

La señal 9 no se puede capturar.

Exit *Provoca la finalización de un proceso puede recibir un parámetro entero que se corresponde con el código de retorno que generará el proceso al terminar. Si no hay parámetro, se devuelve un cero. Los códigos de retorno que se devuelven por convenio son :*

0 terminación correcta.

1 terminación anormal.

2 error en los parámetros.

Xargs *Muchas ordenes de UNIX no pueden recibir sus parámetros desde la entrada estándar lo que impide que otra orden pueda pasárselos mediante una tubería. La orden xargs permite redireccionar la salida de una orden como parámetros de otra.*

Su sintaxis es :

xargs [*indicadores] [orden[(argumentos iniciales)]]*

Admite dos indicadores :

−i toma cada línea de la entrada estándar y realiza una ejecución de la orden para esa entrada.

−p pide confirmación.

8.9 MATRICES Y FORMA DE ACCESO

El ksh permite agrupar variables para formar un vector.

Solo admite matrices bidimensionales.

Los valores que forman una matriz tienen en común la referencia a un único nombre base de matriz entre los valores de la misma.

Los elementos de una matriz no tiene por que ser del mismo tipo.

En UNIX el concepto de variable no tiene declarado un tipo.

Las matrices no son más que un elemento lógico del programador.

Para asignar valores a un elemento de una matriz se utiliza la siguiente sintaxis :

PR[PAT] = 30

PR[ZAN] = 15

PR[LECH] = `no se'

Para acceder al valor de una matriz se hace mediante la siguiente sintaxis :

echo ${PR[PAT]}

Las matrices solo tiene interés cuando relaciones elementos.

Ej :

for I

in `cat hortalizas'

do

echo ${PR[$I]}

done

8.10 LA HERRAMIENTA AWK :

Es una de las cosas más fáciles de explicar y con lo que más problemas vamos a tener.

Es una herramienta de programación.

Se aproxima a la programación funcional.

Se denomina programación por patrones.

Se divide en patrones y acciones. (Acciones asociadas a patrones)

Acciones Lenguaje C

Awk se encarga de leer la entrada (normalmente la estándar). La divide en registros. (Cada registro es una linea Separador de registros(!))

Para cada registro buscamos concordancia con algún patrón, si se produce concordancia ejecuta la acción asociada al patrón sobre el registro.

Awk es un traductor. (La entrada puede ser una cosa, y la salida otra)

La sintaxis de awk es de dos tipos :

awk patrón {acción} ; patrón {acción} ;...´ [fichero de entrada]

awk −f fich.prog [fichero de entrada]

−f Indica separador de campo.

−v Asignación de valor.

8.11 PATRONES

Los patrones son los encargados de resolver la entrada.

Funciona como un gigantesco CASE.

Si no hay patrón para una acción se ejecuta sobre todas las líneas de ficheros.

Patrón vacío es cualquier o ningún registro.

Tipos de patrones :

- Patrón constante : *Son cadenas de caracteres fijas situadas entre barras ` / ' y para las que se busca la aparición en cualquier punto del registro.*

Ej. /patata/ {print} Busca la cadena patata en cualquier posición

- Expresiones regulares : *Son combinaciones de caracteres y operadores de carácter. Los operadores permitidos son*

^ Principio de línea.

$ Fin de línea.

[ ] Clase de caracteres.

| OR

* Cero o más apariciones.

+ Una o más apariciones.

? Cero o una aparición.

. Comodín.(Un carácter)

( ) Agrupación.

− Rango.(Utilizando caracteres ASCII)

Ej. Localice en el fichero passwd aquellos usuarios cuyo número de expediente es impar y pertenecen al grupo 109.

Awk `^f.....[13579] :.* :109 {print} [etc/passwd]

- Comparación de cadenas : *Permiten ejecutar acciones en función de determinados valores del registro de entrada. Para acceder a partes del registro se definen las variables$1, $2...$199 que contienen automáticamente el primero, segundo...etc campo del registro. Los comparadores admitidos son :*

~ Identificación : Comprueba si una cadena se ajusta a un patrón.

!~ No identificación.

== Igualdad.

!= Desigualdad.

<,>... Comparación.

- Patrones compuestos : *Se obtienen combinando patrones simples mediante los operadores :*

&& AND.

|| OR

! NOT.

- Patrones de rango : *Se forman con dos patrones separados por una coma. Awk ejecuta la acción sobre todos los registros de la entrada situada entre el registro que coincide con el primer patrón y el que coincida con el segundo.*
- Patrones BEGIN y END : *Son dos patrones especiales de awk. La acción asociada al patrón BEGIN se ejecuta antes de leer ninguna línea de la entrada. Se utiliza para inicializaciones. La acción asociada a END se ejecuta después de leer el fin de fichero ; se utiliza para presentaciones de resultados. Ninguno de los dos utiliza un registro en una línea para la acción. (No se puede hacer referencia a $1, $2 ..en BEGIN o END porque no hay registros contenidos)*

8.12 ACCIONES

Las acciones en awk son operaciones en lenguaje C que utilizan los campos del registro que concuerdan con el patrón.

La acción vacía es equivalente a un print.

Variables :

Awk utiliza la misma nomenclatura de variables que C, pero no exige que una variable esté declarada para poder usarla.

Para poder operar con una variable debe tener un valor.

Además de las variable definidas por el usuario, awk puede acceder a todas las variables del SHELL situándolas entre comillas.

Awk también ofrece un conjunto de variables predefinido. Las más importantes son :

***FS** Separador de campo.*

***NF** Número de campos del registro.*

***NR** Número de registros leídos.*

***FILENAME** Contiene el fichero de entrada.*

ARGV *Array de argumentos de la llamada a awk.*

Asignación :

−v nombe = valor

8.13 OPERADORES

Permite operadores sobre caracteres y sobre enteros.

Los operadores permitidos son entre otros :

- ***Aritméticos :*** *+, −, *, /, %,^...etc.*
- ***De asignación :*** *=, +=, −=, ^=, %=...etc.*
- ***Operadores de comparación :*** *==, >, <, >=, <=, !=*
- ***Funciones :*** *tan, sen, cos, log, exp, sqrt, rand...etc.*
- ***Funciones sobre cadena :*** *substr, match, length, split...etc.*
- ***Operadores lógicos :*** *&&, ||, !*
- ***Unión de dos cadenas :*** *Se pone una detrás de la otra.*

8.14 ARRAYS

La definición de matrices en awk es idéntica a su definición en UNIX ; Un conjunto de valores que no tienen relación de tipo se encuentran unidos lógicamente por un elemento base y un conjunto de índices.

Los arrays son unidimensionales. Los índices pueden ser cualquiera.

Para acceder a un elemento de array tanto en asignación como en obtención de valor se hace uso de la sintaxis de C, que es la misma que la de pascal.

Para simplificar la gestión de arrays awk ofrece las siguientes estructuras :

- *delete <BASE> [<INDICE>] Elimina un elemento del array.*
- *<subíndice> in <BASE> Es cierto si el subíndice existe.*
- *for <VARIABLE> in <BASE> : Realiza una iteración por índice.*

sentencia

8.15 FUNCIONES DEFINIDAS POR EL USUARIO

La definición de una función utiliza la sintaxis de C, pero sin tipo.

Sintaxis :

function <nombre> (lista de parámetros)

{lista de sentencias}

Aquí no hay tipos, pero puede devolver un valor con la sentencia return. (Si incluye return es función , sino es procedimiento).

8.16 SENTENCIAS DE CONTROL

Las sentencias de control de flujo son :

- **if** (condicion) sentencia [**else**][sentencia]
- **while** (condición) sentencia
- **do** (sentencia) **while** (condición)
- **for** (inicialización ; test ; incremento) sentencia
- **break** Fuerza la salida del bucle.
- **exit** finalización de la entrada.

8.17 FUNCIONES DE UNIX

La sintaxis de las funciones UNIX es :

function nombre

{

orden ;

orden ; Ordenes UNIX (Variables con $)

...

orden ;

}

Donde las ordenes se correspondes con comandos del SHELL. Una vez que se ha definido una función en un SHELL esta es accesible solo desde ese SHELL. Los parámetros de esa función se referencian desde dentro de la misma mediante las pseudo variables $1, $2...etc.

Los parámetros son iguales que en los guiones.

No existe la exportación de funciones.

Las funciones desaparecen cuando desaparece el SHELL.

8.18 E/S EN AWK

Entrada :

Awk recorre automáticamente la entrada estándar analizando cada registro y comparándolo con los patrones.

En ocasiones es necesario leer de forma automáticamente alguna línea de la entrada estándar o de otro fichero. La función ***getline*** *toma una línea de la entrada estándar y almacena su valor en una variable.*

Esta lectura es aparte de la que realiza el awk .

Lee de la entrada estándar una línea y la almacena en una variable.

Salida :

*Para generar salida awk dispone de las funciones **print** y **printf** las dos escriben en la salida estándar pero **printf** permite dar formato a la salida.*

Tanto la entrada como la salida puede redireccionarse a un fichero distinto del estándar mediante >,>> y < con el fichero entre comillas.

El fichero de terminal se especifica como /dev/tty

Ejemplos :

- Dada una tabla como la siguiente :

	ENERO	FEBRERO	...	DICIEMBRE
PERAS	10	20	...	7
UVAS	15	7	...	40
...	...	...	...	...
MANZANAS	15	20	...	30

Generar mediante awk un fichero de salida que muestre la misma tabla y calcule el total anual de cada fruta, el total de ventas por cada mes y el total general.

Solución :

Necesitamos tres patrones diferentes:

1º. Patrón para la cabecera: [^0−9] ó ^[\ t] ó NR ==1

2º. Patrón para el resto: [^0−9] ó NR !=0

3º. Patrón para la última línea: end

Los cuales tiene sus respectivas acciones:

1º { **print** $0 \t TOTAL; // sino se pone acción asume un print.

for (i=2 ; i < NF ; i++) // Para inicializar la matriz.

total [i] = 0 ;

}

2º { suma = 0 ;

for (i=2 ; i < NF ; i++)

{

suma = suma + $i

total[i] = total[i] + $i

}

print $0 \ t suma ;

}

3º { suma = 0 ;

for (i in total)

{

linea = linea \ t total [i] ;

suma = suma + total [i] ;

}

print linea \ t suma ;

}

NOTA : Es bueno poner ; después de todas las acciones.

- Realizar mediante awk un corrector ortográfico que elimine palabras duplicadas y consecutivas. Para cada palabra duplicada solicitará confirmación interactivamente.

<patrón_vacío>

// acción

{

if (ant == $1)

{

print Palabra $1 duplicada, ¿Eliminar ? > /dev/tty

getline resp < /dev/tty ;

if resp = N **printf** (%s \ n, ant)

else printf(\ n)

}

for (i=1 ; i <NF ; i++)

if ($i == $(i+1))

{

print Palabra $1 duplicada, ¿Eliminar ? > /dev/tty

getline resp < /dev/tty ;

if resp = N **printf** (%s, $i) ;

else printf(%s, $i) ;

}

ant = $NF ;

}

BEGIN {

ant =

}

END {

printf (%s, $i)

}

9. ADMINISTRACION DEL SISTEMA

En computadoras que funcionan bajo el sistema operativo UNIX, existe un usuario que se distingue de los demás por ser el encargado de realizar la administración del sistema. Las funciones propias del administrador del sistema son:

- *Actualización y mantenimiento del sistema:*

- *Mantenimiento del sistema de archivos.*
- *Determinación de altas y bajas de archivos.*
- *Control de periféricos.*

- *Realización periódica de copias de seguridad (Backups)*
- *Suministros de soporte técnico al resto de los usuarios.*
- *Gestión de los recursos de la computadora*
- *Etcétera.*

En el sistema operativo UNIX existe un directorio de uso exclusivo del administrador del sistema donde se encuentran una serie de comandos para la realización de dichas funciones, que no pueden ser utilizadas por el resto de los usuarios.

El administrador del sistema, denominado mas comúnmente SUPERUSUARIO, tiene asociada una cuenta que se identifica en el terminal por un símbolo diferente al del resto de los usuarios, normalmente # en lugar de $.

En general, el sistema operativo UNIX puede tener además una serie de usuarios que gocen de cierta libertad

para la gestión del sistema; estos son los denominados usuarios especiales.

El supersusuario es quien organiza y designa el arranque del sistema editando un archivo de comandos de uso exclusivo, en el que se ordenan las siguientes acciones:

- *Comprobar si el sistema de archivos es correcto, para que si encuentra archivos erróneos proceda a su restauración.*
- *Limpiar el registro de usuarios activos.*
- *Montar el sistema de archivos.*
- *Limpiar los directorios temporales.*
- *Arrancar los procesos iniciales (update y cron).*
- *Presentar la fecha en la consola maestra*

De igual forma, el supersusuario establece el proceso de apagado del sistema cuando termina una jornada. Este proceso suele comprender de las siguientes acciones:

- *Enviar mensajes de aviso a los terminales activos.*
- *Terminar o interrumpir los procesos activos excepto el de la consola maestra*
- *Asegurar que toda la actividad sobre el sistema de archivos ha terminado.*
- *Desmontar el sistema de archivos.*

Otras actividades de la administración del sistema son:

- *Cambiar las características de los terminales. Símbolos de teclado, velocidad de transmisión, etc.*
- *Ejecutar periódicamente determinados procesos. Existe un comando (cron) que analiza e un archivo especial (crontab) determinados procesos que tienen que ejecutarse automáticamente cada minuto, hora, día, etc.*

CONCLUSIONES

Se trata de un sistema operativo de los mas utilizados y con mas futuro debido a que son muchos organismos oficiales y particulares los que defienden su utilización, así como muchas firmas de fabricación y comercialización de computadoras que lo incorporan en sus productos. Podemos citar el ejemplo de la Comunidad Económica Europea, que impone el sistema operativo UNIX en todas las aplicaciones que se desarrollan bajo sus auspicios.

Para UNIX el universo es un sistema de ficheros. No existen periféricos, sólo ficheros. De este modo se unifican todos los procesos E/S. Los directorios son ficheros que contienen enlaces con otros ficheros. Terminales, discos compactos e impresoras son ficheros, en teoría se puede escribir y leer de todos ellos. Para encontrar los ficheros en el disco, el sistema utiliza punteros llamados i−nodos. Al borrar un fichero, simplemente se borra su i−nodo; pero, al contrario que en DOS, una vez se ha borrado algo en UNIX es ***IRRECUPERABLE*** *ya que no hay forma de encontrar de nuevo el camino a la información en disco.*

Cada proceso o programa en memoria tiene un owner o propietario que es el sujeto que lo ha lanzado. Desde su nacimiento adquiere los permisos del owner. Cuando un proceso queda huérfano se le denomina demonio o daemon .

El sistema ***no es S.O. de tiempo real****. ¡CUIDADO AL DESCONECTAR UN SISTEMA UNIX!, podrían perderse datos que figuran como archivados. El S.O ejecuta las órdenes cuando quiere y aunque dé por recibida una orden de escritura en disco puede estar dando prioridad a otro programa (realmente realiza una labor de caché interna). Para asegurarse de que todo se refleja en la memoria de masa, es decir, de que se ha escrito de verdad lo que se tenía que escribir, existe la orden sync. Si se desea apagar un ordenador*

bajo UNIX, se deber entrar al sistema como root o supersusuario y ejecutar la orden halt. Una vez el sistema haya realizado las operaciones oportunas, lo notificará y se podrá apagar la máquina sin peligro.

- *Un sistema de ficheros jerárquico en el que todo se encuentra anclado en la raíz. La mayoría de la literatura sobre el tema dice que el sistema de ficheros UNIX es un grafo acíclico, sin embargo, la realidad es que se trata de un grafo cíclico. El DOS, por ejemplo, es un árbol, con un directorio raíz del que cuelgan subdirectorios que a su vez son raíces de otros subárboles. Un grafo cíclico es como un árbol en el que se pueden enlazar nodos de niveles inferiores con un nivel superior. Es decir, se puede entrar en un subdirectorio y aparecer más cerca de la raíz de lo que se estaba.*
- *El sistema de ficheros está basado en la idea de volúmenes, que se pueden montar y desmontar para lo que se les asigna un nodo del árbol como punto de anclaje. Un sistema físico puede dividirse en uno o más volúmenes.*
- *UNIX realiza un riguroso control de acceso a ficheros. Cada uno se encuentra protegido por una secuencia de bits. Sólo se permite el acceso global al root o superusuario. Por tanto, el universo de usuarios de UNIX se encuentra dividido en dos grupos principales, no sólo para el acceso a ficheros sino para todas las actividades: el root, todopoderoso, para el que no hay barreras; y el resto de los usuarios, controlados por el S.O. según las directivas del root.*
- *Una de las grandes ideas de UNIX es la unificación y compatibilidad de todos los procesos de entrada y salida. Para UNIX, el universo es un sistema de ficheros. De esta forma existe compatibilidad entre ficheros, dispositivos, procesos, pipes y sockets.*
- *El núcleo de UNIX es relativamente compacto en comparación con otros sistemas de tiempo compartido. Introduce la idea de reducir el tamaño del kernel y ceder ciertas funciones a programas externos al núcleo llamados demonios. Esto ha sido muy desarrollado y en la actualidad, la tendencia es el desarrollo de micro−kernels, sin embargo UNIX, aunque pionero, es anterior a estos desarrollos.*
- *El sistema presenta comandos de usuario (es decir, a nivel de shell) para iniciar y manipular procesos concurrentes asíncronos. Un usuario puede ejecutar varios procesos, intercambiarlos e interconectarlos a través de pipes o tuberías, simbolizados por el carácter ¦ (ASCII 124). En DOS, también existe la idea del pipe, sin embargo, al no existir concurrencia de procesos, no se trata de una comunicación en "tiempo real", sino de un paso de información a través de ficheros temporales.*
- *UNIX es un S.O. de red, algo que muchos confunden con un S.O. distribuido, lo que se discutirá en los capítulos de internetworking. Por ello, se ha incluido en su núcleo la arquitectura de protocolos de Internet, TCP/IP.*

UNIX, a pesar de su gran éxito y de su profunda evolución, se remonta a comienzos de los años '70. Desde sus orígenes han transcurrido 25 años, inconcebible en informática, donde algo con 5 años es "histórico" o, incluso, "prehistórico". Quizá esta sea la mejor prueba de su eficacia y potencia. Sin embargo, no fue desarrollado con las técnicas y metodología actuales, por lo que no goza de una estructuración en niveles tan clara. Intentando encajar las piezas y para ofrecer una visión genérica al lector se podría desglosar la estructura de UNIX

BIBLIOGRAFIA

- *WEB DE MANUALES DE CHILE*
- *TRABAJO PRACTICO DE LA UNIVERSDAD DE GIJON*
- *Waite, Prata, Martín. "Introducción al UNIX". Anaya, 1986*
- *McGilton, Morgan. "Introducción al UNIX". McGraw, 1988*

- *Lucas, Martín. "Sistema Operativo UNIX". Paraninfo, 1987*
- *Miller, Boyle. "UNIX for Users". Blackwell, 1984*
- *Silvester. "The UNIX System Guidebook". Springer, 1984*
- *http://members.xoom.com/arturovaldes/linuxcur.htm*
- *Universidad Politécnica de Madrid. Departamento de Informática Aplicada. Sistemas Abiertos*
- *Universidad Rey Juan Carlos. Departamento de Ciencias Experimentales e Ingeniería. Sistemas Operativos*
- *http://gondor.gdl.iteso.mx/~mike/unix/indunix.html*
- *http://highland.dit.upm.es:8000/UNIX/*

En UNIX por defecto un bloque son dos sectores y un sector son 512 bytes. La memoria del disco se asigna no por bloques sino por zonas, donde una zona el 2n bloques, tal que n es un valor que decide el administrador cuando crea el sistema de ficheros.

Apéndice CSHELL al final (Independiente del KSHELL)

Sistema Operativo UNIX Página 3

Sistema Operativo UNIX Página 80

VINCULO

VINCULO

Archivo Físico

PROPIETARIO

GRUPO

RESTO

(120)

(6)

(132)

1 .

6 .

1 ..

1 ..

4 bin

...

7 dev

26 prácticas.txt

14 etc

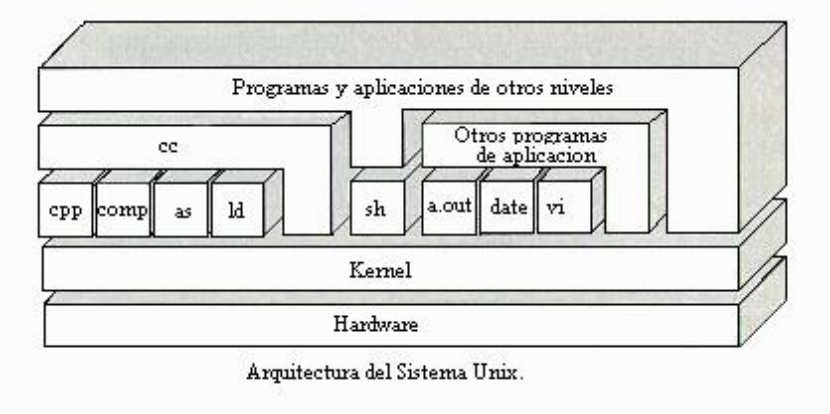
132

6 user

8 tmp

9 spool

...



Arquitectura del Sistema Unix.

UNIX System PDP 7 / 9

Versión 3

PDP 11

INTERDATA

Versión 6

PDP

INTERDATA

Versión 7

PDP 11

INTERDATA

4.1 BSD

VAX

4.2 BSD

VAX

SUN

XENIX

32 V

VAX

System III

PDP 11 / VAX

VARIOS

System V

SPS − 7

System V

PDP 11 / VAX

3 B 2

DESARROLLO INTERNO

BELL LABS

SISTEMAS COMERCIALES

OTROS

BELL LABS

UNIVERSIDAD DE BERKELEY

$ nombre [calificador] [argumentos] [terminador]

$ cc −0 ejemplo.c ; who

-