UNIX ¿UNA OPCION?

Gerardo Antonio Brenes Trejos*
Mario Daniel Ramírez Meléndez*

RESUMEN

Los creadores de **unix** *diseñaron un sistema operativo que facilitara el desarrollo de programas y la manipulación de textos para usuarios expertos, brindando gran facilidad en la comunicación de programas y su programación. Sin embargo, al considerar al usuario como experto, se brinda poca ayuda en el manejo de condiciones de error. En este artículo se presentan las características del ambiente del sistema operativo* **unix**, *con el propósito de que el lector pueda realizar una comparación con los sistemas operativos que conoce.*

RESEÑA HISTORICA

unix es esencialmente una marca que cubre una serie de sistemas operativos desarrollados en los Laboratorios Bell de la ATT (American Telephone and Telegrap Co.). Dennis Ritchie y Ken Thompson, sus creadores, habían trabajado anteriormente en el desarrollo de *Multics*, proyecto conjunto de la General Electric Company el proyecto *MAC* del Instituto Tecnológico de Massachusetts y el grupo de investigación de los Laboratorios Bell.

El sistema se caracterizó por ser pequeño, programado en ensamblador de una PDP–11 con 128K de memoria principal de los cuales **unix** ocupó 60K. Esta primera versión se realizó entre 1969 y 1971. Luego fue reescrito en **C**, lenguaje diseñado especialmente para ese efecto, al mismo tiempo se le definieron nuevas funciones. Su tamaño aumentó en 20K, pero esto fue prácticamente insignificante, dadas las múltiples ventajas de tener el sistema escrito en un lenguaje de alto nivel. Su nombre fue una sugerencia de Brian Kernighan y es un juego de palabras, como crítica al tamaño y complejidad de Multics.

Configuración típica

Una configuración típica para un ambiente **unix** consiste de una unidad central de proceso (CPU) del tipo PDP–11, Zilog 8000 o similar (en configuraciones más pequeñas se ha usado el Motorola 68000 o el Intel 80286), 256K bytes de memoria real, 10 terminales, discos de 30 MB en adelante y una unidad de cinta.

VARIAS VERSIONES

En el momento de desarrollo del sistema, ATT estaba imposibilitado de mercadear **unix,** debido a un acuerdo firmado con el gobierno federal norteamericano. Pero sí se ofrecieron copias de **unix**, para propósitos educativos, a diferentes universidades, siendo la más significativa de ellas la Universidad de California (Berkeley) que desarrolló una serie de variantes conocidas como **BSD** (**B**erkeley **S**oftware **D**istributions), la más reciente de las cuales es 4.3 BSD. Por su parte, ATT reunió las diferentes variantes en algo que para efectos comerciales fue conocido como **unix** Sistema III. A esta última versión se le agregaron algunas opciones, saliendo al mercado como **unix** Sistema V, que desde 1983 cuenta con soporte oficial de los laboratorios Bell de la ATT. La última versión conocida como SVR3

* Departamento de Computación. Instituto Tecnológico de Costa Rica

(System V Release 3), incluye algunas opciones como manejo de semáforos y memoria compartida que no existían en versiones anteriores. Según Maurice Bach[1], para principios de 1984 había aproximadamente 100000 instalaciones **unix** en el mundo, en una gran variedad de máquinas –desde microcomputadoras hasta supercomputadoras– y de diferentes proveedores. Debido a esta multiplicidad de usuarios y de ambientes, se ha desarrollado a pasos agigantados la producción de *software* para ambientes **unix**. Otro esfuerzo importante es el realizado por un grupo de la IEEE para tratar una estandarización de **unix**, usando hasta el momento System V como base y discutiéndose la posibilidad de incluir algunas de las mejoras introducidas por Berkeley, tales como el protocolo TCP/IP para telecomunicaciones. Muy notable es X/OPEN, un esfuerzo multiempresarial que tiene lugar para desarrollar una especificación estándar de la interfaz de **unix** para los programadores.

CARACTERISTICAS GENERALES

En las diferentes máquinas en que se ha implementado el sistema **unix,** los usuarios tienen un ambiente en común, esperan encontrar (luego de entrar al sistema) un sistema operativo con ciertas opciones características que incorporan una cierta filosofía de desarrollo de *software*. El objetivo de **unix** es proveer un ambiente muy poderoso y flexible para el desarrollo de *software*. Lo anterior se logra por medio de la combinación que se hace del sistema de archivos, el intérprete de comandos y el tratamiento de los dispositivos de entrada y salida.

unix se desarrolló en capas. Las dos capas principales son, el núcleo (la más baja) y los procesos del usuario (la más alta). De afuera hacia adentro, las capas son: interacción con el usuario *(shell)*, procesos del usuario, el núcleo y dispositivos periféricos (implementados como archivos especiales en el directorio **dev**).

Sistema de archivos

En **unix** un archivo es una secuencia de bytes, por lo general en código ASCII, almacenado en bloques de 1K o 1/2K. Incluyen datos, código fuente (programas), código objeto (ejecutable), archivo de comandos y manejador de dispositivos. Los directorios son considerados también archivos.

Los archivos se clasifican en tres tipos: archivos ordinarios, directorios y archivos especiales.

Archivos ordinarios

Son archivos tipo texto, constituidos por una secuencia de bytes, donde el concepto de registro no se usa, quedando a discreción del usuario el tipo de manipulación deseada. **unix** provee una serie de servicios para la manipulación de archivos, y existen funciones para:

- Leer n caracteres
- Escribir n caracteres
- Posicionarse en determinado carácter del archivo
- Leer el siguiente carácter
- Retornar al anterior carácter
- etc.

Como se puede apreciar solo existe manipulación de archivos tipo secuencial o *random*; para poder llevar a cabo un acceso tipo indexado, por ejemplo, éste debe ser implementado por el usuario. Lo mismo sucede con los otros métodos de acceso.

Son considerados archivos ordinarios los programas fuente, el código ejecutable, los archivos que contengan cualquier tipo de dato, exceptuando los directorios y los archivos especiales.

Directorios

Los directorios son manejados en una estructura jerárquica, con un nivel de profundidad indefinido. El contenido puede ser cualquier cantidad y tipo de archivo.

En todo directorio existen dos sub-directorios definidos para efectos de navegación, ".." para direccionar al directorio padre, y "." que apunta al directorio actual. Por ejemplo si estoy en la estructura a varios niveles de profundidad y deseo subir tres niveles, uso el comando **cd**../../..

El directorio principal se denomina raíz, y es direccionado por medio de barra inclinada (/), a partir

de él se construye toda la estructura, lo que permite realizar referencia a los archivos en forma directa (especificando toda la ruta). Por ejemplo **cat/user/est/rojas/prog/tarea 1**, o indirecta (a partir del directorio actual) como en **cat ../ ../prog/tarea1.**

Archivos especiales

Los diferentes dispositivos permitidos en el sistema (disco duro, terminales, unidades de cintas, impresoras, etc.), son definidos por medio del comando **mknod**, que especifica si el dispositivo es tipo carácter o block, la dirección y el nombre del mismo. Dicho comando crea un archivo que contiene ciertas características del dispositivo. Para hacer referencia al dispositivo, se menciona el archivo. Por ejemplo, para realizar un respaldo del directorio/**bin**, en un diskete, se usaría el comando **tar–cvf /dev/sfda/bin**. Por lo general todas las definiciones de dispositivos son agrupadas en el directorio **/dev**. En el ejemplo anterior, **/sfda** estaría asociado con "**s**mall **f**loppy **d**isk **a**".

Protección

Cada archivo puede tener tres tipos de usuarios: el dueño, el grupo al cual pertenece el dueño y el resto de usuarios de sistema. Para cada uno de estos tipos de usuario se especifica el tipo de acceso que puede realizar. Los tipos de acceso permitidos son lectura, escritura y ejecución. Estos son controlados por el dueño quien decide qué tipo de protección desea asignar. También existe un usuario denominado "super usuario", quien tiene potestad para cambiar los atributos de un archivo, cualquiera que sea su protección.

En el momento de su creación el archivo posee una protección pre–establecida. Como la filosofía de **unix** es compartir, dicha protección por lo general permite lectura y escritura a los diferentes tipos de usuarios, y ejecución al dueño.

En el caso de un directorio, el poder ejecutarlo se interpreta como el hecho de poderlo establecer en determinado momento como el directorio actual, por medio del comando **cd.**

Procesos

Todos los eventos que ocurren en el ambiente **unix** son efectuados por medio de la creación de un proceso. Tanto a los diferentes comandos que invoca el usuario, como a los programas que ejecutan, se les asigna un número de identificación de proceso que se utiliza para invocaciones futuras.

El sistema usa las rutinas **fork** y **exec** para crear dichas tareas, pudiéndose establecer comunicación entre los procesos creados. El mecanismo de *fork* está también disponible para el usuario, quien desde sus aplicaciones puede crear sus propios procesos y establecer comunicación entre ellos.

Cuando un proceso es ejecutado, éste toma control de la terminal en que se invoca hasta su terminación (ejecución en *foreground*). Si el usuario desea tener control sobre la terminal mientras el programa se ejecuta, para poder invocar otros comandos, es necesario incluir al final del comando el carácter **&** (ejecución en *background*). En este último caso, al teclear el comando, el sistema despliega el número de identificación del proceso y aparece el símbolo que señala la terminal como disponible *(prompt).*

Interacción con el usuario

A cada usuario se le asigna un *shell* en la definición del número de usuario (identificación o *login*), este programa es el primero en atenderlo cuando ingresa al sistema (una vez cumplido el requisito de dar su correspondiente número de usuario y la palabra clave, asignados para su protección). Por lo general el *shell* es un programa intérprete de comandos, que intercepta todas las diferentes instrucciones que se dan y decide qué acción se realiza. Existen dos tipos de *shell*, uno denominado ***bourne shell*** y el otro ***c shell.***

Si se deseara que un usuario en particular solo pueda ejecutar un programa en el momento de ingresar al sistema (para captura de datos, por ejemplo), ese programa se puede definir como el *shell* de ese número de usuario y una vez que el usuario finaliza la ejecución del mismo, el sistema vuelve a pedir que se digite un número de usuario. Para el usuario el ***shell*** es **unix**, para el núcleo el ***shell*** es un proceso del usuario.

Redirección de la entrada y salida

Cada uno de los diferentes comandos de **unix**, por lo general, recibe su entrada de la terminal y despliega su resultado también en la terminal.

Por ejemplo: el comando **ls** lista el contenido de determinado directorio. Si se teclea **ls /bin,** el contenido del directorio **bin** que se encuentra en el directorio raíz se presentará en la pantalla. Pero si se cambia por **ls /bin > lista**, en lugar de presentarse en la terminal, se crea un archivo de nombre **lista** con dicha información. Si en lugar del carácter ">" se usa ">>", el sistema agrega el resultado al final del archivo **lista**, si es que ya existía.

Para redireccionar la entrada se utiliza el carácter "<", por ejemplo si se desea clasificar un archivo y crear uno nuevo con la información ya clasificada, se usa: **sort <arch_act>arch_clas**. En este ejemplo se toma el contenido del archivo **arch_act**, se clasifica y los datos ordenados se graban en el archivo **arch_clas.**

Para los programas que desarrolla el usuario también es posible utilizar la entrada y salida asignada a la terminal, para facilitar su depuración y, una vez probados, se puede utilizar la redirección anterior, para usarlos con diferentes archivos y evitar con ello complicaciones adicionales.

Comodines

El *shell* permite la utilización de comodin "*", en la lista de argumentos de los comandos.

El asterisco ("*") permite la omisión de n caracteres y puede repetirse tantas veces como se desee, por ejemplo si el argumento es ***a*x*1**, los nombres que posean una "**a**", luego una "**x**", y termine en "**1**", serán los que se consideren como argumentos.

Pipes y filtros

La salida estándar de un programa o comando se puede convertir en la entrada estándar de otro programa o comando, por medio del carácter "**|**" (carácter para establecer un ***pipe***) permitiéndose con ello ahorrar tiempo en la creación de archivos intermedios, que una vez utilizados, deben ser eliminados. Por ejemplo, si se desea obtener el contenido de un directorio a doble espacio, es necesario crear un archivo y luego pasar ese archivo a doble espacio. Con un *pipe*, la creación del archivo es transparente para el usuario de la siguiente forma: **ls /bin | pr –d.**

Cuando intervienen más de dos programas o comandos, uno o algunos tienen su entrada y su salida redireccionada y actúan como filtros. Si se desea imprimir el resultado del ejemplo anterior, se puede realizar con el siguiente comando: **ls /bin | pr – d | 1p**. En este caso el comando **pr** está actuando como filtro.

Los programas, mediante llamadas al sistema, pueden establecer *pipes* con otros programas, pero el intercambio de información es controlado por medio de instrucciones del lenguaje (en **C** existen instrucciones para abrir, cerrar, leer y escribir en un *pipe*).

AMBIENTE Y FILOSOFIA

Ritchie y Thompson[5] resumen las facilidades del sistema en:

1. Un sistema de archivos jerárquicos con volúmenes que pueden ser incorporados por el usuario al sistema y eliminados en el momento que se desee por medio de los comandos **mount** y **umount**
2. Compatibilidad en archivos, dispositivos de entrada y salida entre procesos
3. Habilidad para iniciar la ejecución de procesos en forma asincrónica
4. Un lenguaje de comandos del sistema seleccionable por el usuario.

La combinación de esas facilidades provee el ambiente de programación del sistema **unix**. Este promueve la escritura de herramientas de *software* pequeñas, programas elegantes que se concentran en una función simple y por eso fácil de construir, entender, describir y mantener.

La programación en el *shell* puede proveer una rápida y fácil elaboración de las primeras versiones experimentales de los programas; aquellos programas diseñados para interactuar con archivos pueden ser analizados en la terminal y la flexibilidad del *shell* facilita su rediseño.

El lenguaje de programación C

unix fue re-escrito en **C**, el cual fue diseñado por D. M. Ritchie para ese propósito. Su nombre se

refiere a **C** como una secuencia del lenguaje **B** diseñado previamente por Ritchie. Algunos compiladores han sido escritos para proveer lenguajes tales como Fortran, Pascal, Lisp, APL, Modula 2 y Ada en ambientes **unix**. Sin embargo, **C** es el único lenguaje común en todos los sistemas **unix** y es la interfaz más natural con el sistema.

C es un lenguaje pequeño y versátil, que se limita a operaciones que son fácilmente traducidas al lenguaje de máquina. Las funciones más complejas (incluyendo entrada y salida) son cubiertas por rutinas definidas en bibliotecas. Tiene un amplio conjunto de operadores, incluyendo manipulación de datos a nivel de bit.

HERRAMIENTAS PARA EL DESARROLLO DE *SOFTWARE*

unix ofrece en sí un ambiente propicio para el desarrollo de *software*, pero también existen herramientas diseñadas específicamente para facilitar y automatizar algunas labores de desarrollo, éstas incluyen:

1. Recompilación parcial. **make** es un utilitario que recompila solo las partes de un sistema que han sido modificadas después de la última recompilación.
2. Control de modificaciones de sistemas. El *Source Code Control System* es un conjunto de comandos que registra los cambios realizados sobre archivos y permite reconstruirlos en determinada fecha.
3. **sdb:** depurador simbólico. **sdb** fue diseñado para trabajar en conjunto con el compilador de **C**, el cual genera suficiente información como para permitir al **sdb** interactuar con un programa que se está ejecutando. Facilita la ubicación de puntos de control, el seguimiento instrucción por instrucción del programa, así como examinar variables por nombre. El vaciado de memoria que se genera cuando un programa termine en forma anormal también puede ser analizado por este utilitario.
4. Herramientas de desarrollo de lenguajes. Uno de ellos conocido como y**acc** (*yet another compiler compiler*), convierte una gramática libre de contexto en un conjunto de tablas para un autómata que ejecuta un algoritmo de análisis gramatical. La gramática puede ser ambigua, pero se pueden especificar reglas de precedencia para eliminar dichas ambigüedades. Otra herramienta, el **lex**, es un generador de reconocedores de expresiones regulares, que puede ser usado para generar analizadores de léxico.

   yacc y lex han sido usados en conjunto para producir un compilador y un marco preprocesador más portables para el lenguaje **C**.

CATEGORIZACION DE LOS COMANDOS

Los comandos más usados en **unix** se pueden clasificar en nueve principales categorías, ellas son:

1. Obtención de acceso y comunicaciones
2. Manipulación de archivos y directorios
3. Edición de texto y preparación de documentos
4. Compilación y (o) interpretación de programas
5. Interrogación del estado
6. Ayudas de programación
7. Control avanzado de ejecución
8. Manipulación avanzada de archivos y datos
9. Fijación de terminales y gráficos

POSIBLES INCONVENIENTES DE UNIX

El diseño de **unix** incluye una serie de características que en determinadas circunstancias pueden ocasionar pérdidas de información no controladas por el usuario, las que si se ignoran pueden repercutir en forma muy desfavorable en el rendimiento que, bajo dicho sistema operativo, pueda brindar un equipo a la organización. Cabe anotar que la mayoría de los sistemas administradores de bases de datos (SABD) que corren en ambiente **unix** tienen algún tipo de facilidad para recuperación de información y mantenimiento de pistas de auditoría.

AREA DE ALMACENAMIENTO RAPIDA

En todas las versiones de sistemas **unix**, la transferencia de datos ocurre entre el sistema de archivos y un área de almacenamiento rápido. Cuando un bloque va a ser leído, el sistema primero revisa si está en dicha área. Si es así, el sistema no tiene que leerla del disco. Cuando un bloque se

escribe, es ubicado en un área similar, que es grabada en el disco hasta que se llene o después de cierto intervalo de tiempo. Lo anterior genera, definitivamente un incremento en el rendimiento del sistema, porque no es necesario esperar a que termine la operación de escritura para retornarle el control al programa; sin embargo puede suceder que si existe una caída del sistema por una falla eléctrica, la información residente en el área se pierda mientras el usuario supone que ya reside en el disco.

Se han realizado diferentes correcciones a dicho problema. La más viable se ha orientado a la creación de versiones que no tomen ventaja de lo anterior, sacrificando un poco el rendimiento.

En ambientes donde existe un comportamiento muy inestable de la energía eléctrica , se usan fuentes ininterrumpidas de poder (UPS) suficientemente poderosas para eliminar dicha inestabilidad.

Otra solución a medias es la ejecución de un utilitario en *background*, para forzar que cada cierto intervalo de tiempo (5 segundos es un estándar más o menos establecido) se escriba el contenido del área a disco, sin embargo durante ese intervalo, el problema de confiabilidad sigue existiendo.

SISTEMA CORRUPTO

El sistema de archivo es dividido en bloques (por lo general de 1K o 1/2K) que son asignados a los diferentes archivos que se van creando en el sistema.

Dentro del sistema existe:

1. Un *boot block* que contiene la secuencia de arranque del equipo
2. Un *super block* que contiene información de las áreas *i–node, file y swap*. También contiene un apuntador a la lista de bloques libres
3. El área *i–node* que contiene el descriptor de cada archivo o directorio en el disco. Por convención, el segundo *i–node* representa el directorio raíz del disco
4. El área *file* que es usada para almacenar el contenido de los archivos
5. El área *swap* que guarda los procesos de los usuarios cuando son sacados de la memoria principal.

Si el sistema es apagado en forma incorrecta, se da alguna falla del *hardware*, o una falla en el fluido eléctrico ocasiona una caída del sistema, se produce una pérdida de información, debido a que toda la información que existe en el área de almacenamiento rápido se pierde, surgiendo bloques duplicados, tamaños de archivos incorrectos, pérdidas de referencias a archivos, cuenta de bloques libres errónea, bloques duplicados en la lista de libres,etc.

Existe el utilitario **fsck** *(File System Checking)*, que revisa y repara el sistema de archivos en forma interactiva. Si se detecta alguna inconsistencia, ésta es reportada al operador, quien elige si lo desestima o repara.

Cuando se ha producido una falla, es necesario, una vez reiniciado el sistema, realizar la revisión del sistema de archivos, para eliminar los posibles errores que se han provocado. Si la revisión se omite, el sistema funcionará con un nivel de corrupción, que irá aumentando cada vez más si se continúa usando el sistema, hasta llegar al punto que el sistema no funciona más y la pérdida de información puede alcanzar niveles desastrosos.

En complemento al **fsck**, existe el comando **mklost+found**, que crea el directorio **lost+found**, utilizado por el sistema para guardar información del sistema que es utilizada en el momento de corregir las corrupciones. En algunas ocasiones es posible recuperar archivos por medio de información adicional que se graba en este directorio. El directorio **lost+found** es creado en cada uno de los volúmenes que el sistema posee.

USUARIO EXPERTO

unix fue desarrollado para ser un sistema operativo de uso personal, creado por dos expertos, quienes lo diseñaron para cubrir sus necesidades como usuarios. No se incluyó un sistema apropiado de mensajes de error, ni protección de los posibles errores que cometa el usuario en la invocación de comandos, al ser sus nombres muy reducidos, por lo general de dos caracteres.

El sistema de seguridad permite al administrador del sistema limitar la invocación de ciertos comandos, que podrían ocasionar problemas si son utilizados en forma incorrecta por usuarios principiantes. Una vez que el usuario ha acumulado suficiente experiencia, se le puede permitir acceso a todos los recursos del sistema, para que pueda obtener de él todas las facilidades existentes.

En cuanto a la ausencia de mensajes de error, una vez que el usuario está bien familiarizado con el sistema, y ha utilizado los diferentes comandos, dichos mensajes no son necesarios. En este caso, mensajes resumidos, facilitan la labor del usuario.

CONCLUSION

Aunque **unix** no fue desarrollado con fines comerciales, en la actualidad encontramos muchos equipos funcionando bajo dicho sistema operativo en diversas empresas de nuestro país. En la mayoría de los casos, se utiliza para llevar a cabo labores de soporte a la administración, y en otros con fines educacionales, donde está obteniendo mucha popularidad.

Quizá el problema mayor que se presenta al utilizar un equipo que corre bajo **unix**, es el hecho que el usuario no está acostumbrado a trabajar con un sistema con un tipo de filosofía tan diferente al resto de los existentes en Costa Rica, por lo que enfrenta una serie de inconvenientes para poder cumplir bien con sus labores. Sin embargo, una vez que el usuario conoce bien el sistema, puede obtener una serie de facilidades, que le permitirán realizar sus funciones en una forma más rápida y eficiente.

Además, las facilidades para comunicaciones que presenta el sistema, la forma en que se manejan los diferentes usuarios y dispositivos, la diversidad de herramientas desarrolladas para correr en ambientes **unix** y la estandarización existente en ámbito mundial en el uso del mismo lo presentan como una excelente opción cuando se desee un ambiente multiusuario.

REFERENCIAS BIBLIOGRAFICAS

1. Bach, M. **The design of the unix operating system.** New Jersey: Prentice–Hall, 1987.
2. Holt, R. **Concurrent Euclid, the unix system and tunis.** Massachusetts: Addison–Wesley, 1983.
3. Kernighan, B. y Ritchie, D. **The unix programming environment.** New Jersey: Prentice–Hall, 1984.
4. Kernighan, B. y Ritchie, D. **The C programming language.** New Jersey: Prentice–Hall, 1978.
5. Ritchie, D. M. y Thompson, K. *The unix time–sharing system.* **The Bell System Technical Journal.** 1978.
6. Rochkind, M. J. **Advanced unix programming.** New Jersey: Prentice–Hall, 1985.
