Módulo 9: Núcleo de Linux

Finalmente hemos llegado al tema de Linux propiamente dicho, ya que como dijimos hace tiempo, lo que Linus Torvalds desarrolló (y aún lo hace) siendo estudiante, fue un pequeño pero autosuficiente núcleo (*kernel*) para el procesador 80386, el primer procesador de 32 bits verdadero en la gama de CPU compatibles con PC de Intel. El primer núcleo Linux liberado al público fue la versión 0.01, fechada 14 de mayo de 1991. Esta versión no trabajaba con redes, sólo se ejecutaba en procesadores Intel compatibles con el 80386 y hardware de PC, y contaba con soporte de *drivers* de dispositivos extremadamente limitado. El único sistema de archivos que se reconocía era el de Minix. No fue sino hasta el 14 de marzo de 1994 que apareció la siguiente versión de "hito", Linux 1.0. Tal vez la función individual nueva más importante fue el trabajo con redes: 1.0 incluía soporte de los protocolos de redes TCP/IP estándar de UNIX. El núcleo 1.0 también incluía un sistema de archivos nuevo muy mejorado sin las limitaciones del sistema de archivos Minix original. En junio de 1996 apareció Linux 2.0, con la adición de dos nuevas capacidades importantes: soporte de múltiples arquitecturas, incluido un traslado a un sistema Alpha nativo de 64 bits, y soporte de arquitecturas multiprocesador. En enero de 1999 aparece Linux 2.2 con muchas mejoras y soporte para nuevos tipos de hardware[30]. En enero de 2001 aparece Linux 2.4 con mejoras en el soporte a multiprocesadores, dispositivos como el USB, y acceso directo al HW gráfico (2D, 3D).

En muchos sentidos, el núcleo de Linux es el corazón del proyecto Linux, pero otros componentes constituyen el sistema operativo Linux completo. Mientras que el núcleo de Linux se compone exclusivamente de código escrito desde cero específicamente para el proyecto Linux, una buena parte del software de soporte que constituye el sistema Linux no es exclusivo de Linux, sino común a varios sistemas operativos tipo UNIX. En particular, Linux utiliza muchas herramientas desarrolladas como parte del sistema operativo BSD de Berkeley, el X Window System del MIT y el proyecto GNU de la Free Software Foundation.

Características del núcleo

El núcleo de Linux se implementa como un núcleo monolítico tradicional por razones de desempeño, pero su diseño es lo bastante modular como para permitir que la mayor parte de los controladores se cargue y descargue dinámicamente durante la ejecución.

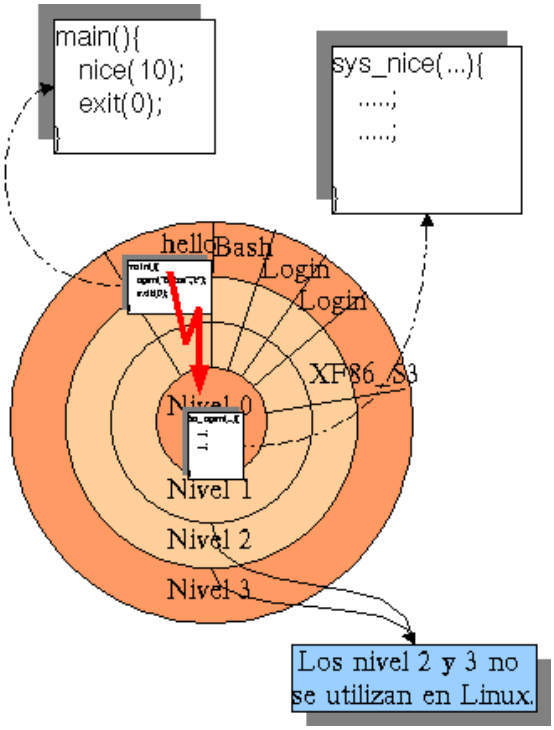
En su diseño global, Linux semeja cualquier otra implementación tradicional de UNIX que no sea en micronúcleo. Linux es un sistema multiusuario, multitarea, con un conjunto completo de herramientas compatibles con UNIX.

Linux es un sistema multiusuario que ofrece protección entre procesos y ejecuta múltiples procesos bajo el control de un planificador de tiempo compartido. Los procesos recién creados pueden compartir partes selectas de su entorno de ejecución con sus procesos padres, lo que hace posible la programación multihilada (*multithreaded*). La comunicación entre procesos se apoya tanto en los mecanismos de System V[31] - colas de mensajes, semáforos y memoria compartida - como en la interfaz de *sockets* de BSD.

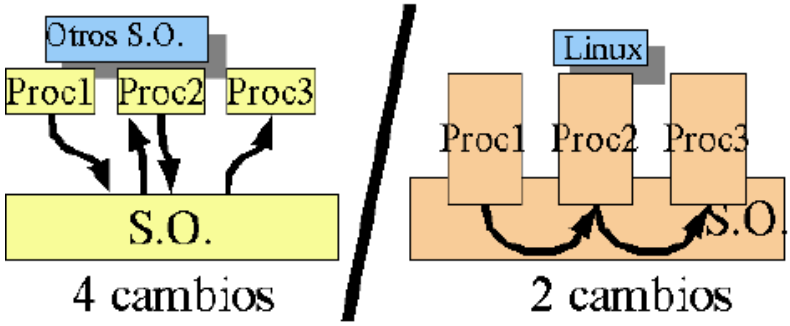
El código del núcleo se ejecuta en el modo privilegiado del procesador con pleno acceso a todos los recursos físicos del computador. Linux llama a este modo privilegiado *modo de núcleo*. Bajo Linux, no se incorpora código en modo de usuario en el núcleo. Cualquier código de soporte del sistema operativo que no necesite ejecutarse en modo de núcleo se coloca en las bibliotecas del sistema.

[30] El primer número cambia cuando se da una evolución importante, el segundo es la versión y el tercero la revisión. Las versiones pares son estables y la impares inestables.

[31] Tenga en cuenta esta característica para seleccionarla al momento de compilar el núcleo.



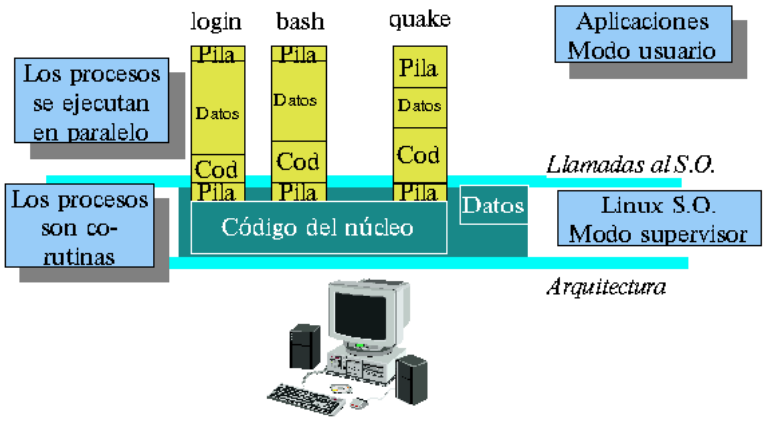
Aunque diversos sistemas operativos modernos han adoptado una arquitectura de transferencia de mensajes para su núcleo, Linux conserva el modelo histórico de UNIX: el núcleo se crea como un solo binario monolítico. La razón principal es para mejorar el desempeño: dado que todo el código y las estructuras de datos del núcleo se mantienen en un solo espacio de direcciones, no se requieren conmutaciones de contexto[32] cuando un proceso invoca una función del sistema operativo o cuando se entrega una interrupción de hardware. No sólo el código de planificación y memoria virtual central ocupan este espacio de direcciones; *todo* el código del núcleo, incluido el de los *drivers* de dispositivos, sistemas de archivos y trabajo con redes, está presente en el mismo espacio de direcciones único.



Conmutación de contexto

En muchos sistemas operativos actuales el núcleo es un proceso; y por lo tanto la forma de acceder al sistema operativo es mediante un cambio de contexto. Sin embargo el núcleo de Linux NO es un proceso. En Linux el cambio de contexto se realiza bajo demanda. El núcleo (el proceso en ejecución, dentro del núcleo) ejecuta una instrucción para cambiar a otra tarea (que también estará en el núcleo, en el punto en el que se quedó al ceder la CPU). Los procesos en Linux son corrutinas que se ceden el procesador de forma explícita.

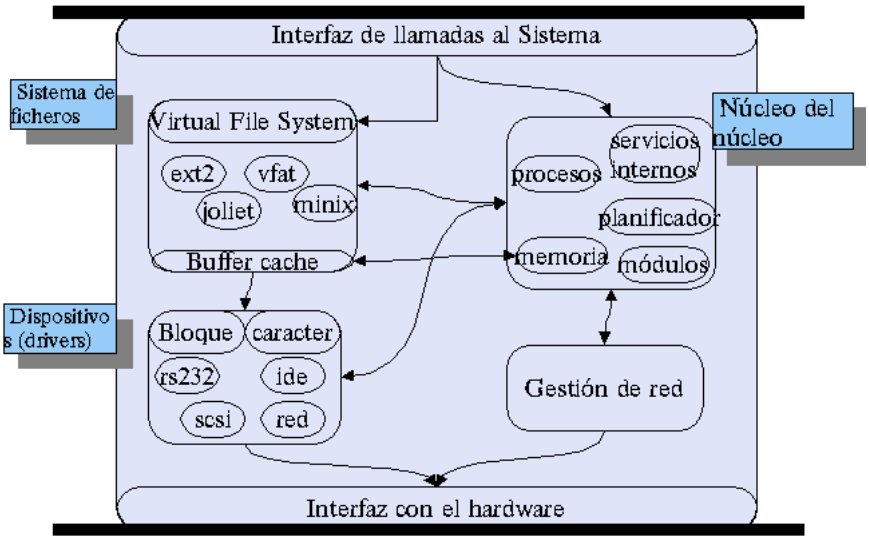
[32] Con el objeto de implementar el tiempo compartido, el sistema operativo restablece registros, variables internas y *buffers* y modifica varios parámetros como preparación para la ejecución del siguiente programa, a esto se lo llama *conmutación de contexto*, luego de la cual el siguiente programa continúa su ejecución a partir del punto en el que se había quedado.



En la arquitectura i386 se pueden invocar hasta cuatro mecanismos distintos para desencadenar el cambio de tarea "automáticamente". Sin embargo, por cuestiones de flexibilidad y seguridad, a partir de la versión 2.2 de Linux, el cambio de contexto está codificado paso a paso.

Módulos

El núcleo de Linux tiene la facultad de cargar y descargar secciones arbitrarias del código del núcleo cuando se le pide hacerlo[33]. Estos **módulos** de núcleo cargables se ejecutan en modo de núcleo privilegiado, y por ello tienen pleno acceso a todas las capacidades de hardware de la máquina en la que se ejecutan. En teoría, no hay restricción respecto a lo que se permite a un módulo de núcleo hacer; típicamente, un módulo podría implementar un *driver* de dispositivo, un sistema de archivos o un protocolo de redes.



Hay varias razones por las que es conveniente tener módulos del núcleo. El código fuente de Linux es libre, así que cualquier persona que desee escribir código del núcleo puede compilar un núcleo modificado y rearrancar el sistema para cargar esa nueva funcionalidad; sin embargo, volver a compilar, enlazar y cargar todo el núcleo es un ciclo demasiado laborioso como para realizarlo cuando se está desarrollando un nuevo *driver*. Si se usan módulos del núcleo, no es necesario crear un nuevo núcleo para probar un nuevo *driver*; éste podría compilarse de forma independiente y cargarse en el núcleo que ya se está ejecutando. Desde luego, una vez que se escribe un nuevo *driver*, puede distribuirse como módulo para que otros usuarios puedan beneficiarse de él sin tener que reconstruir sus núcleos.

[33] Actualmente es así, en las versiones anteriores era necesario un proceso “demonio” llamado *kerneld* que se encargaba de la carga y descarga de módulos.

Procesos e hilos

La mayor parte de los sistemas operativos modernos manejan tanto procesos como hilos *(threads*). Aunque la diferencia precisa entre los dos términos suele variar de una implementación a otra, podemos decir que un *hilo*, también llamado *proceso ligero (LWP, lightweight process)*, es una unidad básica de utilización de la CPU, y consiste en un contador de programa, un juego de registros y un espacio de pila. El hilo comparte con sus hilos pares la sección de código, sección de datos y recursos del sistema operativo como archivos abiertos y señales, lo que se denomina colectivamente una *tarea*[34]. Un proceso tradicional o *pesado* es igual a una tarea con un solo hilo.

Podríamos intentar definir la distinción principal entre ambos: los *procesos* representan la ejecución de programas individuales, mientras que los *hilos* representan contextos de ejecución individuales pero concurrentes dentro de un solo proceso que ejecuta un solo programa. Dos procesos individuales cualesquiera tienen su propio espacio de direcciones independiente, aun si están usando memoria compartida para compartir parte del contenido (pero no todo) de su memoria virtual. En contraste, dos hilos dentro del mismo proceso comparten el mismo espacio de direcciones (no sólo espacios de direcciones similares: cualquier cambio que un hilo haga a la organización de la memoria virtual será visible de inmediato para los demás hilos del proceso, porque en realidad sólo hay un espacio de direcciones en el que todos se están ejecutando). Hay varias formas distintas de implementar los hilos. Se puede implementar un hilo en el núcleo del sistema operativo como objeto propiedad de un proceso, o puede ser una entidad totalmente independiente. Los hilos no tienen que implementarse en el núcleo; es posible hacerlo enteramente dentro del código de una aplicación o biblioteca con la ayuda de interrupciones de temporizador suministradas por el núcleo. El núcleo de Linux maneja de forma sencilla la diferencia entre procesos e hilos: utiliza exactamente la misma representación interna para todos. Un hilo no es más que un proceso nuevo que por casualidad comparte el mismo espacio de direcciones que su padre.

Planificación de procesos

El planificador es el algoritmo responsable de repartir el uso del procesador o procesadores entre todos los procesos activos del sistema. A pesar de la importancia que se le suele dar a los algoritmos de planificación en los libros sobre sistemas operativos, en la práctica el código del planificador es pequeño y sencillo. Es importante diferenciar entre el algoritmo de planificación propiamente dicho y el cambio de contexto. El **algoritmo de planificación** únicamente tiene que decidir cuál será el siguiente proceso que utilizará el procesador (se elije un proceso ganador entre todos los que no están bloqueados). El **cambio de contexto** es el mecanismo que efectivamente pone en ejecución a un proceso. Esta operación es muy dependiente de la arquitectura del procesador sobre el que se esté ejecutando, por lo que se tiene que realizar a bajo nivel (en ensamblador). Para poder planificar procesos, tenemos que saber qué es un "proceso". En todo sistema operativo un proceso está representado por una estructura de datos donde se guarda toda la información relevante de éste, el PCB (Process Control Block). En Linux, el PCB es una estructura llamada `task_struct` en el archivo `include/linux/sched.h`. En ella aparece todo tipo de información sobre cada uno de los procesos. Inicialmente, la **tabla de procesos** era un vector de tamaño fijo de `task_struct`, con lo que el número máximo de procesos estaba limitado. Actualmente, la tabla de procesos es realmente una lista doblemente enlazada mediante `next_task` y `prev_task`.

Una vez que el núcleo llega a un punto de replanificación, debe decidir cuál proceso ejecutará a continuación. Linux tiene dos algoritmos de planificación de procesos independientes. Uno es un algoritmo de tiempo compartido para la planificación expropiativa (*preemptive*) justa entre múltiples procesos, y el otro está diseñado para tareas en tiempo real en las que las prioridades absolutas son más importantes que la equitatividad. Por ejemplo los procesos de un grabador de CD-RW o un reproductor de música en MP3 debería utilizar las políticas orientadas a procesos de tiempo real.

En el caso de procesos de tiempo compartido, Linux usa un algoritmo con prioridades *basado en créditos*. Cada proceso posee cierto número de créditos de planificación; cuando es preciso escoger una nueva tarea para ejecutarla, se selecciona el proceso que tiene más créditos. Cada vez que ocurre una interrupción de

[34] No confundir con el concepto de tarea del intérprete de comandos bash.

temporizador, el proceso que se está ejecutando en ese momento pierde un crédito; cuando sus créditos llegan a cero, se suspende y se escoge otro proceso.

Si ningún proceso ejecutable tiene créditos, Linux realiza una operación de renovación de créditos en la que se añaden créditos a todos los procesos del sistema (no sólo a los que se pueden ejecutar) según esta regla: *créditos=(créditos/2)+prioridad*.

Este algoritmo tiende a combinar dos factores: el historial del proceso y la prioridad del proceso. Después de aplicarse el algoritmo, un proceso retendrá la mitad de los créditos que le habían quedado después de la última operación de renovación de créditos, con lo que se conserva un antecedente del comportamiento reciente del proceso. Los procesos que se ejecutan todo el tiempo tienden a agotar sus créditos rápidamente, pero los que pasan una buena parte de su tiempo supendidos pueden acumular créditos a lo largo de varias renovaciones y por consiguiente terminarán con una mayor cantidad de créditos después de una renovación. Este sistema de crédito da prioridad automáticamente a los procesos limitados por E/S, para los cuales es importante una respuesta rápida.

Multiprocesamiento simétrico

El núcleo de Linux 2.0 fue el primer núcleo Linux estable que manejó hardware de *multiprocesador simétrico (SMP, symmetric multiprocessor)*. Procesos o hilos individuales se pueden ejecutar en paralelo en procesadores distintos. Sin embargo, a fin de conservar los requisitos de sincronización no expropiable del núcleo, la implementación de SMP en este núcleo impone la restricción de que sólo un procesador a la vez puede estar ejecutando código en modo de núcleo. Esta situación puede llegar a cambiar en el futuro próximo con el actual desarrollo del núcleo expropiable versión 2.5.

Gestión de memoria

La memoria es uno de los recursos más valiosos que administra el sistema operativo. Uno de los elementos principales que caracterizan un proceso es la memoria que utiliza. Ésta está lógicamente separada de la de cualquier otro proceso del sistema, excepto los *threads* de un mismo proceso que comparten normalmente la mayor parte de la memoria que tienen asignada. Un proceso no puede acceder, ni accidentalmente, al espacio de memoria asignado a otro proceso, lo cual es imprescindible para la seguridad y estabilidad del sistema. En Linux, además, un proceso tiene dos espacios de memoria: el espacio de memoria del usuario, único para ese proceso, y el espacio de memoria del núcleo, idéntico en todos los procesos.

La gestión de memoria en Linux tiene dos componentes. Primero, el sistema de gestión de memoria física se encarga de asignar y liberar páginas, grupos de páginas y bloques pequeños de memoria. El segundo componente maneja la memoria virtual, que es memoria que tiene una correspondencia con el espacio de direcciones de procesos en ejecución.

El termino de "memoria virtual" se asocia a dos conceptos que normalmente aparecen unidos:

1. El uso del almacenamiento secundario para ofrecer al conjunto de las aplicaciones la ilusión de tener más memoria RAM de la que realmente hay en el sistema. Esta ilusión existe tanto a nivel del sistema, es decir, teniendo en ejecución más aplicaciones de las que realmente caben en la memoria principal, sin que por ello cada aplicación individual pueda usar más memoria de la que realmente hay, o incluso, de forma más general, ofreciendo a cada aplicación más memoria de la que existe físicamente en la máquina.
2. Ofrecer a las aplicaciones la ilusión de que están solas en el sistema, y que, por lo tanto, pueden usar el espacio de direcciones completo. Esta técnica facilita enormemente la generación de código, puesto que el compilador no tiene porque preocuparse sobre dónde residirá la aplicación cuando se ejecute.

Denominaremos paginación con intercambio a la primera técnica y memoria virtual a la segunda.

En la paginación con intercambio se da cabida a la ejecuciőn de más aplicaciones de las que pueden residir simultáneamente en la memoria del sistema, y de mayor tamaño que la memoria disponible. Se dividen los

procesos en fragmentos de tamaño fijo, intercambiando con el almacenamiento secundario sólo aquellos fragmentos que sean necesarios. Todos los fragmentos (páginas) pertenecientes a una aplicación no tienen por qué estar simultáneamente en memoria principal. Una aplicación puede tener más fragmentos de los que caben en memoria principal.

Gestión de memoria física

El administrador primario de memoria física del núcleo de Linux es el **asignador de páginas**, el cual se encarga de asignar y liberar todas las páginas físicas, además de que puede asignar intervalos de páginas contiguas físicamente si se le solicitan. El asignador usa un algoritmo de **montículo de compañeras** para seguir el rastro a las páginas disponibles. Un asignador de montículo de compañeras aparea unidades adyacentes de memoria asignable; de ahí su nombre. Cada región de memoria asignable tiene una compañera adyacente, y siempre que dos regiones compañeras asignadas quedan libres, se combinan para formar una región más grande. Esa región mayor también tiene una compañera con la que puede combinarse para formar una región todavía mayor.

División de memoria en un montículo de compañeras

Memoria virtual

El sistema de memoria virtual de Linux se encarga de mantener el espacio de direcciones visible para cada proceso. Este sistema crea páginas de memoria virtual por solicitud y gestiona la carga de dichas páginas de disco o su intercambio a disco si es necesario. Bajo Linux, el administrador de memoria virtual mantiene dos vistas distintas del espacio de direcciones de un proceso: como un conjunto de regiones individuales y como un conjunto de páginas.

Intercambio y paginación

Una tarea importante de un sistema de memoria virtual es reubicar páginas de la memoria física al disco cuando se necesita la memoria. Los primeros sistemas UNIX efectuaban esta reubicación intercambiando a disco el contenido de procesos enteros a la vez, pero los UNIX modernos se apoyan más en la *paginación*: la transferencia de páginas individuales de memoria virtual entre la memoria física y el disco. Linux no implementa el intercambio de procesos enteros; emplea exclusivamente el mecanismo de paginación.

Memoria virtual del núcleo

Linux reserva para su propio uso una región constante, dependiente de la arquitectura, del espacio de direcciones virtual de cada proceso. Las entradas de la tabla de páginas que corresponden a estas páginas del núcleo se marcan como protegidas, de modo que no pueden verse ni modificarse cuando el procesador esté operando en modo de usuario.

Ejecución de programas de usuario

La ejecución de programas de usuario por el núcleo de Linux se dispara con la llamada al sistema **exec**. Esta llamada ordena al núcleo ejecutar un programa nuevo dentro del proceso actual, sobreescribiendo por completo

el contexto de ejecución actual con el contexto inicial del nuevo programa. La primera tarea de este servicio del sistema es verificar que el proceso invocador tenga permiso de ejecutar el archivo. Una vez comprobado eso, el núcleo invoca una rutina de carga para iniciar la ejecución del programa. El cargador no necesariamente carga el contenido del archivo de programa en la memoria física, pero al menos establece la correspondencia entre el programa y la memoria virtual.

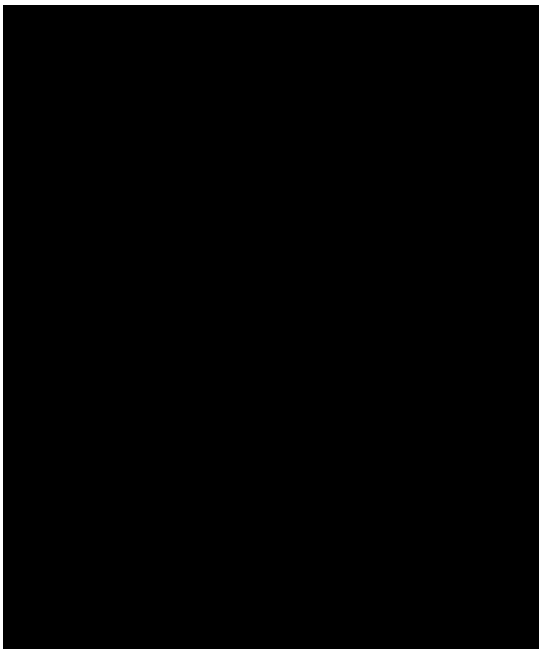
Correspondencia entre los programas y la memoria

Bajo Linux, el cargador de binarios no carga un archivo binario en la memoria física. Más bien, se establece una correspondencia entre las páginas del archivo binario y regiones de la memoria virtual. Sólo cuando el programa trata de acceder a una página dada se genera un fallo de página (*page fault*) que hace que se cargue esa página en la memoria física.

Sistemas de archivos

Linux retiene el modelo de sistema de archivos estándar de UNIX. En UNIX, un archivo no tiene que ser un objeto almacenado en disco o que se trae por una red desde un servidor de archivos remoto. Más bien, los archivos UNIX pueden ser cualquier cosa capaz de manejar la entrada o la salida de un flujo de datos. Los *drivers* de dispositivo pueden aparecer como archivos, y el usuario también ve como archivos los canales de comunicación entre procesos o las conexiones de red.

El núcleo de Linux maneja todos estos diferentes tipos de archivos ocultando los detalles de implementación de cualquier tipo de archivo individual detrás de una capa de software, el *sistema de archivos virtual (VFS, virtual file system)*.



El sistema de archivos virtual

El VFS de Linux se diseñó según los principios de la orientación a objetos, y tiene dos componentes: un conjunto de definiciones que definen el aspecto que puede tener un objeto archivo y una capa de software para manipular dichos objetos. Los tres tipos principales de objetos definidos por el VFS son las estructuras de *objeto i-nodo* y *objeto archivo*, que representan archivos individuales, y el *objeto sistema de archivos*, que representa todo un sistema de archivos.

El sistema de archivos Linux ext2fs

El sistema de archivos de disco estándar que Linux emplea se denomina *ext2fs* por razones históricas. Linux se programó originalmente con un sistema de archivos compatible con Minix, a fin de facilitar el intercambio de datos con el sistema de desarrollo Minix, pero ese sistema de archivos estaba muy restringido por lo límites de 14 caracteres para los nombres de archivo y de 64 megabytes para el sistema de archivos. El sistema de archivos Minix fue reemplazado por uno nuevo, que se bautizó como *sistema de archivos extendido (extfs, extended file system)*. Un rediseño posterior de este sistema de archivos para mejorar el desempeño y la escalabilidad y añadir unas cuantas funciones que faltaban dio pie al *segundo sistema de archivos extendido, ext2fs*.

Ext2fs tiene mucho en común con el Sistema de Archivos Rápido (ffs) de BSD. Las diferencias principales entre ext2fs y ffs atañen a políticas de asignación de disco. En ffs, el disco se asigna a los archivos en bloques de 8 kilobytes, y los bloques se subdividen en fragmentos de 1 kilobyte para almacenar archivos pequeños o bloques parcialmente llenos al final de un archivo. En contraste, ext2fs no usa fragmentos, sino que realiza todas sus asignaciones en unidades más pequeñas. El tamaño de bloque por omisión en ext2fs es de 1 kilobyte, aunque también se manejan bloques de 2 y 4 kilobytes. Para mantener un desempeño alto, el sistema operativo debe tratar de realizar la E/S en trozos grandes siempre que sea posible, *agrupando* solicitudes de E/S adyacentes físicamente. El agrupamiento reduce el gasto extra por solicitud que los drivers de dispositivo, discos y controladores de disco en hardware incurren. Un tamaño de solicitud de E/S de 1 kilobyte es demasiado pequeño para mantener un buen desempeño, por lo que ext2fs usa políticas de asignación diseñadas para colocar bloques lógicamente adyacentes de un archivo en bloques de disco físicamente adyacentes, a fin de poder emitir una solicitud de E/S por varios bloques de disco en una sola operación.

Dentro de un grupo de bloques, ext2fs trata de mantener las asignaciones físicamente contiguas si es posible, reduciendo la fragmentación si puede. Ext2fs mantiene un mapa de bits de todos los bloques libres de un grupo de bloques. Al asignar los primeros bloques de un archivo nuevo, ext2fs comienza a buscar un bloque libre desde el principio del grupo de bloques; al extender un archivo, continúa la búsqueda a partir del bloque que se asignó más recientemente al archivo.

El sistema de archivos Linux ext3fs

Esta es la versión con *journalling* del sistema de archivos extendido segundo (ext2fs), también llamado “ext3”. Podríamos traducir *journalling* como “libro diario”; este libro diario mantiene la pista de cualquier cambio que haya sido hecho al sistema de archivos, de manera que ante una eventual caída del sistema se podrán recuperar las modificaciones a partir de este “libro diario”. Prácticamente no será necesario ejecutar el reparador del sistema de archivos **e2fsck** al reiniciar el sistema. Es importante recordar que este *journal* trabaja a nivel del sistema de archivos, es decir que no hay que confundir su función con la que podría realizar una aplicación (supongamos un editor de texto) que recupera los cambios efectuados a un archivo a partir de un “archivo *journal*”. Este “libro diario” es un bloque específico adicional en el sistema de archivo “ext2”, por este motivo es posible transformar un sistema “ext2” a “ext3” sin mayores inconvenientes, el formato del “ext3” es idéntico al del “ext2” y se puede cambiar de uno a otro fácilmente. Este sistema de archivos está soportado en el núcleo de Linux y puede compilarse como módulo.

El sistema de archivos proc de Linux

El VFS de Linux tiene la suficiente flexibilidad como para poder implementar un sistema de archivos que no almacene datos persistentes en absoluto, sino que más bien proporcione una interfaz a otra funcionalidad. El sistema de archivos de procesos de Linux, conocido como sistema de archivos **proc**, es un ejemplo de sistema de archivos cuyo contenido no está almacenado realmente en ninguna parte, sino que se calcula bajo demanda según las solicitudes de E/S de archivo de los usuarios[35].

[35] Tenga en cuenta esta característica para seleccionarla al momento de compilar el núcleo.

El sistema de archivos **proc** no es exclusivo de Linux, UNIX SVR4 introdujo un sistema de archivos **proc** como una interfaz eficiente con el soporte de depuración de procesos del núcleo. Cada subdirectorio del sistema de archivos correspondía no a un directorio de disco, sino a un proceso activo del sistema actual. Un listado del sistema de archivos revela un directorio por proceso, donde el nombre del directorio es la representación ASCII decimal del identificador de proceso (PID) único de ese proceso.

Linux implementa un sistema de archivos **proc** de este tipo, pero lo extiende considerablemente añadiendo varios directorios y archivos de texto adicionales bajo el directorio raíz del sistema de archivos. Estas nuevas entradas corresponden a diversas estadísticas relativas al núcleo y a los *drivers* cargados asociados. El sistema de archivos **proc** ofrece a los programas una forma de acceder a esta información en forma de archivos de texto simple, que se pueden procesar con la potentes herramientas que el entorno de usuario UNIX estándar proporciona. Por ejemplo, en el pasado el comando **ps** tradicional de UNIX para listar los estados de todos los procesos en ejecución se ha implementado como proceso privilegiado que lee el estado de los procesos directamente de la memoria virtual del núcleo. Bajo Linux, esta orden se implementa como un programa sin privilegios que sencillamente analiza sintácticamente y formatea la información de **proc**.

Entrada y salida

Para el usuario, el sistema de E/S de Linux es muy parecido al de cualquier UNIX. Esto es, hasta donde es posible, los *drivers* de dispositivos aparecen como archivos normales. Un usuario puede abrir un canal de acceso a un dispositivo de la misma forma como puede abrir cualquier otro archivo; los dispositivos pueden aparecer como objetos dentro del sistema de archivos. El administrador del sistema puede crear archivos especiales dentro de un sistema de archivos que contengan refencias a un *driver* de dispositivo específico, y un usuario que abra tal archivo podrá leer del dispositivo al que hace referencia, y escribir en él. Usando el sistema de protección de archivos normal, que determina quién puede acceder a cuál archivo, el administrador puede establecer permisos de acceso para cada dispositivo.

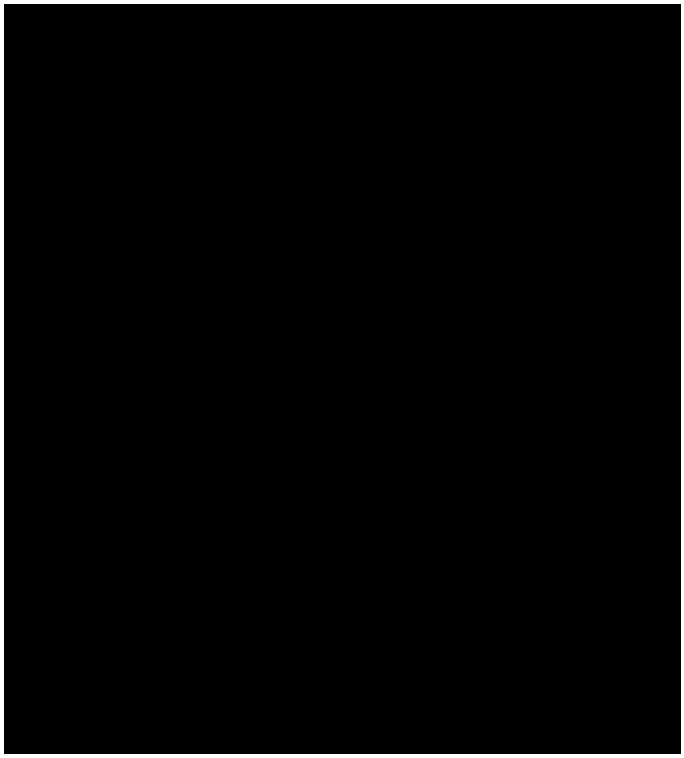
Linux divide todos los dispositivos en tres clases: *dispositivos por bloques*, *dispositivos por caracteres* y *dispositivos de red*. Los dispositivos por bloques incluyen todos los dispositivos que permiten acceso aleatorio a bloques de datos de tamaño fijo totalmente independientes, incluidos los discos rígidos, discos flexibles y CD-ROM. Los dispositivos por bloques suelen usarse para almacenar sistemas de archivos, pero también se permite el acceso directo a un dispositivo por bloques para que los programas puedan crear y reparar el sistema de archivos que el dispositivo contiene. Los dispositivos por caracteres son casi todos los demás, con la importante excepción de los dispositivos de red. Los dispositivos por caracteres no necesitan apoyar toda la funcionalidad de los archivos normales. Por ejemplo, un parlante permitirá escribir datos en él, pero no manejaría la lectura de datos de él. De forma similar, se podría manejar la búsqueda a una posición determinada de un archivo en el caso de un dispositivo de cinta magnética, pero no tendría sentido en un dispositivo de apuntar como un ratón. Los dispositivos de red se tratan de manera diferente que los dispositivos por bloques y por caracteres. Los usuarios no pueden transferir datos directamente a los dispositivos de red, sino que deben comunicarse indirectamente abriendo una conexión al subsistema de red del núcleo.

El trabajo con redes es un área de funcionalidad clave para Linux. No sólo apoya los protocolos estándares de Internet que se emplean para casi todas las comunicaciones UNIX-UNIX, sino que también implementa varios protocolos nativos de otros sistemas operativos distintos de UNIX. En particular, dado que Linux se implementó originalmente en PC, no en estaciones de trabajo grandes ni en sistemas de clase servidor, reconoce muchos de los protocolos que suelen usarse en las redes de PC, como AppleTalk de Apple e IPX de Novell.

Internamente, el trabajo con redes en el núcleo de Linux se implementa con tres capas de software:

- La interfaz de *sockets*.
- Controladores de protocolos.
- *Drivers* de dispositivos de red.

Las aplicaciones de usuario efectúan todas las solicitudes de trabajo con redes a través de la interfaz de *sockets*.



¿Por qué compilar el núcleo?

La velocidad y eficiencia siguen siendo metas importantes del diseño de Linux, pero gran parte de los trabajos recientes y actuales con Linux se han concentrado en un segundo objetivo importante del diseño: la estandarización. Uno de los precios que se pagaron por la diversidad de implementaciones de Unix que hay actualmente es que el código fuente escrito para una de ellas no necesariamente se compilará o ejecutará correctamente en otra. Aun si las mismas llamadas al sistema están presentes en dos sistemas Unix distintos, no necesariamente se comportarán exactamente de la misma manera.

Los estándares POSIX son un conjunto de especificaciones de diferentes aspectos del comportamiento de un sistema operativo. Hay documentos POSIX para la funcionalidad común del sistema operativo y para extensiones como hilos de procesos y operaciones en tiempo real. Linux se diseñó de modo que cumpliera con los documentos POSIX pertinentes; al menos dos distribuciones de Linux han logrado la certificación POSIX oficial.

Básicamente tendríamos dos situaciones posibles por las que compilaríamos el código fuente del núcleo: para actualizarlo o para personalizarlo o ceñirlo a nuestras necesidades.

Si tenemos un sistema GNU/Linux funcionando pero su núcleo está desactualizado, y además le agregamos al computador algún periférico de última generación, seguramente nuestro núcleo no lo reconocerá o no estará suficientemente soportado. En esta situación será necesario actualizarlo a la última versión. Recordemos que básicamente no es necesario actualizar el sistema completo, sino solamente el núcleo. Aunque esto último a veces es recomendable.

Los núcleos nuevos normalmente ofrecen la posibilidad de entenderse con más accesorios hardware (o sea, incluyen más controladores), se ejecutan más rápidamente, son más estables o corrigen errores de otras versiones. Mucha gente se actualiza el núcleo para poder usar nuevos controladores que necesitan o librarse de "bugs" de la versión que usaban.

Pero si tenemos el último sistema GNU/Linux funcionando con el núcleo actualizado, es probable que queramos personalizarlo o ceñirlo a nuestras necesidades. Las distribuciones de Linux generan un núcleo "genérico" con una serie de características que lo hagan factible de instalar en la mayor cantidad de máquinas posibles, con soporte para los dispositivos más difundidos en el mercado. De manera que no es raro encontrar que estos núcleos tienen soporte tanto para dispositivos PCMCIA (para poder ser instalados en notebooks

portátiles) como para dispositivos SCSI (para poder ser instalados en equipos grandes y potentes que cumplirán la función de servidores) y para dispositivos IDE (para poder ser instalados en PC hogareñas). Generalmente los núcleos han sido compilados para procesadores tipo 80386, y estos procesadores difícilmente se estén usando en estos días, sino más bien el 80586 o Pentium o superior. Obviamente el núcleo compilado para 80386 no hace uso de las características más avanzadas del Pentium (ni qué decir del Pentium II, III, IV). De manera que acá tenemos una muy buena oportunidad de compilar el núcleo adaptándolo a nuestro hardware.

Recordemos además que si logramos un núcleo compilado para nuestro procesador específico, tendrá mejor desempeño y velocidad, y podemos dejar compilados como módulos para el futuro a todos aquellos controladores del hardware que tal vez le incorporemos más adelante.

¿Cuánto espacio en disco necesito?

Según el "Kernel-HOWTO" la versión 2.0.10 del núcleo ocupaba, comprimida, 6 megabytes, pero al descomprimir ocupaba unos 24 MB. Pero aquí no acaba la cosa: para compilar se necesita espacio para archivos temporales, dependiendo de la configuración que se elija. Por ejemplo, en un 386, con controlador de red de 3Com y cinco sistemas de archivos supone 30 MB. Si a esto añadimos las fuentes comprimidas, serán 36 MB. Recordemos que en cada nueva versión se agregan gran cantidad de líneas de código, por ejemplo la versión 2.0 tiene unas 400.000 líneas de código, pero ya la 2.1 tiene unas 800.000.

Pero la versión 2.4.7 ocupa unos 100 MB, y la 2.4.18 unos 150 MB !!

¿Cuánto tarda en compilar?

Obviamente tarda mucho, según algunos muchísimo. Por supuesto que está en función directa con el tipo y velocidad del procesador, la cantidad de memoria y la velocidad del disco rígido; en cuanto al hardware, y a la cantidad de opciones y módulos que hayamos seleccionado para compilar.

¿De dónde obtengo los fuentes?

Es muy importante obtener los fuentes de un lugar confiable. Es decir, del CD de la distribución que utilizamos, o de los sitios oficiales o sus "mirrors" o espejos autorizados. El sitio web oficial de los fuentes de Linux es http://www.kernel.org/ [36]. Ahí se pueden obtener todas las versiones y están indicados los mirrors autorizados, con copias idénticas al original. Decimos que es muy importante, porque sería desastroso si obtuviéramos una versión que ha sido alterada con mala intensión (lo cual si bien es difícil no es imposible). O también podríamos encontrarnos con una versión alterada para algún otro propósito o en desarrollo; de nuevo, si bien esto es difícil no es imposible.

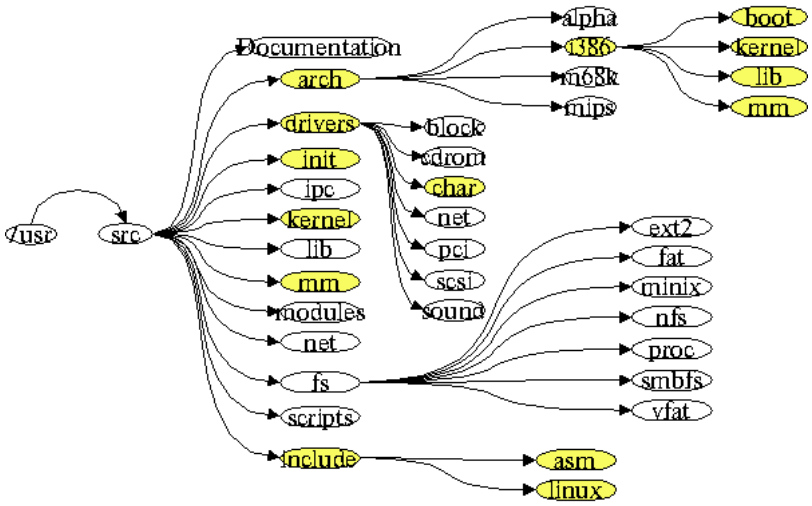
Típicamente tienen el nombre linux-x.y.z.tar.gz, donde x.y.z es el número de versión y revisión. Las versiones se encuentran en directorios v1.1, v1.2 y v1.3 (-- Y 2.0 y 2.1--) .

También podemos optar por instalar los fuentes desde el CD de la distribución que utilizamos, en cuyo caso se tratará de un paquete con formato "rpm" o "deb", según si siguen el formato de Red Hat o Debian.

En cualquiera de los dos casos los fuentes deben quedar por debajo del directorio /usr/src. En él se crea un directorio "linux" que contiene los fuentes[37].

[36] Se puede obtener de vía ftp desde ftp://ftp.kernel.org/pub/linux/kernel/ .

[37] A partir de la versión 2.4 se puede colocar en otro directorio, en el que uno tenga permiso de escritura.



Debemos notar que a veces el directorio **linux** es un enlace (*link*) a otro directorio, por ejemplo **linux-2.2.14**. Esto se hace para que podamos tener más de una versión de los fuentes. Es decir, luego podríamos crear un directorio **linux-2.2.16**, donde colocaríamos esa versión, y cambiamos el enlace simbólico para que apunte al nuevo directorio. Recordemos que los programas que compilan los fuentes los buscan en el directorio / usr/src/linux, no importa si es un enlace o no.

Descompresión de los fuentes

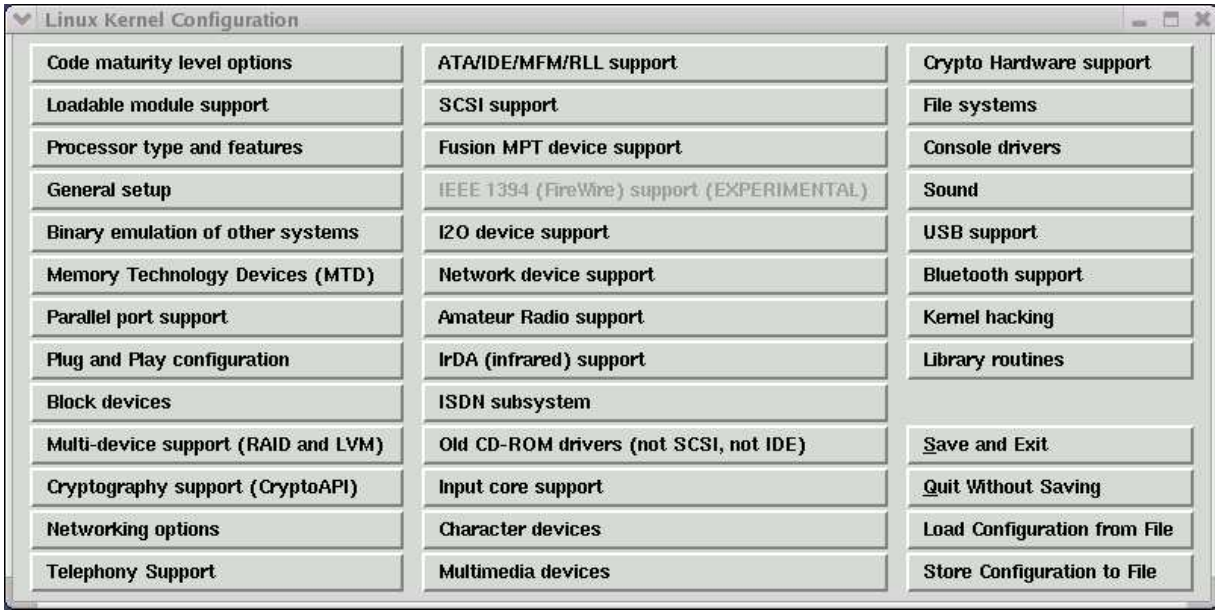
A partir de ahora, toda la tarea la debe realizar el superusuario **root**. Deberemos cambiarnos al directorio / usr/src. Si ya existe el directorio **linux**, habrá que renombrarlo como explicábamos antes. Si los fuentes están en formato .tar.gz, deberemos descompactarlo con los programas **gzip** y **tar**, como ya sabemos, o con **tar** y la opción **z**. El contenido del paquete se expandirá en /usr/src/linux. Y una vez creado, si queremos podemos optar por transformarlo en un enlace o no.

Configuración del núcleo

En el directorio /usr/src/linux hay ahora un archivo README, que es muy importante que lo leamos. La configuración se hace con el programa **make**, que acepta una de tres opciones en la línea de comando: **config**, **menuconfig** o **xconfig**. La primera invoca una sesión de preguntas y respuestas en la consola. En caso de que nos equivoquemos o cambiemos de idea, habrá que abortar con CTRL/C y empezar de nuevo desde el principio.

La segunda inicia una sesión con menúes desplegables y ventanas enlazadas sencillas en formato texto y color. Todas las opciones con <> indican que son modularizables, y las que están con [] indican que no son modularizables. Si queremos seleccionar una opción, deberemos tocar la barra espaciadora y aparecerá un asterisco indicando que ha sido seleccionada para ser incluída dentro del kernel. Si tocamos otra vez la barra espaciadora (o la tecla n) desaparece el asterisco indicando que dicha opción no está seleccionada. Finalmente para las opciones que están entre <> podemos tocar la tecla m y aparecerá una M mayúscula para indicarnos que esa opción está seleccionada como módulo.

La tercera opción es para ser ejecutada dentro de una ventana **xterm** en un entorno X. Aparecerá una ventana con botones, soporte para mouse, etc. Es muy intuitiva de utilizar; las opciones que serán incluídas en el núcleo monolítico tienen una y griega, m que será compilada como módulo y n que no será compilada. Obviamente es la más cómoda y la más bonita, sin embargo no olvidemos que tanto X como la compilación del kernel son dos operaciones que consumen bastantes recursos de procesador y memoria principal. De manera que si estamos con poco tiempo, ésta no es la mejor opción. Pero podríamos utilizarla sólo para la configuración, luego apagar todos los procesos X (cambiando de nivel de ejecución, por ejemplo) y compilar después.



Estas últimas dos opciones (**menuconfig** y **xconfig**) tienen además una pequeña ayuda en línea. No son muy extensas, pero se supone de alguna manera que si está compilando el núcleo, debería saber lo que está haciendo y por lo tanto no debería necesitar muchas explicaciones, ya que las opciones son obvias. ¿No es cierto?.

Además estas opciones tienen una respuesta por omisión, y generalmente es la respuesta adecuada.

Opciones de la configuración del núcleo

Code maturity level options

Algunas de las cosas que el núcleo de Linux soporta, pueden estar en estado de desarrollo aún, y la funcionalidad, estabilidad o el grado de evaluación del mismo tal vez no sean suficientes para el uso general. Es el llamado "estado alfa" de desarrollo, y no es aconsejable su uso "en producción". Sin embargo, está ahí para todos aquellos que quieran intentar probarlo, para ver qué tal funciona, y eventualmente los más capacitados podrán hacer algún tipo de aporte. Entonces a menos que tenga intención de probar software que todavía no está maduro, debería responder que "No", con lo cual, en adelante se le presentarán muchas menos opciones, es decir, sólo la del software considerado "maduro".

Loadable module support

Los módulos del núcleo son pequeñas piezas de código compilado que pueden ser insertados o extraídos, desde el núcleo en ejecución. Los módulos pueden ser controladores de dispositivos (*device drivers*), sistemas de archivos o distintos formatos de ejecutables binarios. Un núcleo modular es más eficiente porque solamente está cargado en memoria el código necesario. De manera que generalmente responderá "y" a esta opción.

Processor type and features

Es el tipo de procesador de su CPU. Esta opción se ofrece como optimización, ya que el compilador generará código máquina para el procesador específico que usted elija aquí. Tenga en cuenta que el tipo elegido tiene compatibilidad "hacia adelante" pero no "hacia atrás", vale decir que si usted elige "Pentium", el núcleo ejecutará en un "Pentium Pro", pero no en un "80486".

General setup

Opciones generales tales como si queremos soporte en el núcleo para redes, soporte para el bus PCI, EISA, MCA o PCMCIA. A la comunicación entre procesos tipo "System V" seguramente responderá afirmativamente, ya que sino muchos programas no podrán ejecutar. La interface *sysctl* provee un medio para cambiar dinámicamente ciertos parámetros y variables del núcleo durante su ejecución sin necesidad de recompilar el núcleo o reiniciar el equipo, seguramente responderá que sí a esta facilidad. También podrá optar por tener soporte en el núcleo para distintos formatos de programas ejecutables, tales como "a.out", ELF y MISC. El primero es un tanto antiguo, ELF es el estándar de Linux y MISC se refiere a formatos de Java, DOS, Phyton o Emacs-Lisp; seguramente responderá que sí a estas opciones.

Binary emulation of other systems

Soporte en el núcleo para poder ejecutar programas binarios provenientes de SCO Unix, Unixware, Solaris, Wyse y otros. Generalmente no es necesario activar estas opciones.

Memory Technology Devices (MTD)

Se refiere a chips RAM o flash, generalmente usados en sistemas embebidos (*embedded*) o de estado sólido. Generalmente no es necesario activar estas opciones.

Parallel port support

Si quiere usar dispositivos conectados al puerto paralelo de su máquina, ya sea impresoras, dispositivos ZIP o un enlace PLIP (Parallel Line Internet Protocol), entonces deberá responder que "si".

Plug and Play configuration

Plug and Play (PnP) es un estándar para periféricos que les permite ser configurados por software, asignándoles la IRQ u otros parámetros. Responda afirmativamente ("y") si le gustaría que Linux configure sus dispositivos PnP, también debería responder afirmativamente a "ISA Plug and Play support" que es para soporte PnP para el bus ISA.

Block devices

Soporte para los dispositivos especiales por bloque, tales como disketteras, PS/2 con MCA y discos rígidos ESDI (máquinas IBM), viejas controladoras de XT, dispositivos IDE externos que se conectan a través del puerto paralelo (PARIDE), controladoras Compaq Smart Array, soporte para dispositivos *loop*, que permiten usar un archivo común como si fuera un dispositivo especial por bloque, esto permite por ejemplo montar un archivo que contiene una imagen ISO 9660 antes de ser grabada en el CD, o uno que contenga una imagen de diskette con sistema de archivo DOS, vfat, Minix, ext2, etcétera. También se ofrece soporte para dispositivos por bloques de red (similar al anterior pero a través de una red en modelo cliente-servidor). Además podemos optar por tener soporte para discos RAM. Todas estas opciones pueden ser marcadas como módulos ("m").

Multi-device support (RAID and LVM)

Soporte para múltiples ejes *(spindles)* físicos a través de un único dispositivo lógico. Es requerido para RAID (combinar varias particiones de un disco rígido en un dispositivo de bloques lógico) y administración de volumenes lógicos (LVM).

Cryptography support (CryptoAPI)

Habilita en el núcleo el soporte para las API (Application Programming Interface) Criptográficas (encriptación). En caso de habilitarse luego se harán preguntas relativas a los distintos algoritmos de encriptación. No es necesario habilitar estas funciones para usar aplicaciones que utilizan encriptación tales como PGP, SSL, etc.

Networking options

Distintas opciones para soporte de redes en el núcleo, tales como filtrado y fragmentado de paquetes (útil para *firewall*), filtrado de *sockets*, soporte para TCP/IP, multicast de IP, Linux como *router*, túneles de IP sobre IP, IP móvil, túneles de IPv6 sobre IPv4, *router* de multicast, soporte para IPv6, soporte para VLAN's (IEEE 802.1Q), para protocolo IPX de Novell, para protocolo Appletalk de Apple, para DECnet de Digital Equipment Corporation (ahora comprada por Compaq), para el estándar IEEE 802.1d Ethernet Bridging, y una multitud más de protocolos y controladores considerados "experimentales", pero que pueden usarse muy bien. Casi todos pueden habilitarse como módulos.

Telephony support

Para placas de telefonía, que le permiten usar un teléfono común para aplicaciones de voz sobre IP. Puede compilarse como módulo.

ATA/IDE/MFM/RLL support

Soporte para las unidades de almacenamiento masivo de bajo costo ATA (AT Attachment), (E)IDE ((Enhanced) Integrated Disks Electronics) y ATAPI (ATA Packet Interface). Seguramente responderá que sí, salvo que su equipo solamente posea interfaces SCSI.

SCSI support

Si tiene un disco rígido, una unidad de cinta, una lectora de CD-ROM o cualquier otro dispositivo SCSI, tendrá que responder que sí, pero luego se le preguntará de que placa se trata. Se puede compilar como módulo.

Fusion MPT device support

Para SCSI de alta performance, LSI Logic Fusion Message Passing Technology (MPT). Sólo si tiene esta placa debería responder afirmativamente.

I2O device support

La arquitectura Intelligent Input/Output permite a controladores de hardware dividirse en dos partes: un módulo específico de sistema operativo llamado OSM y un módulo específico de hardware llamado HDM. El OSM puede "hablarle" a un gran rango de HDM's, e idealmente los HDM's son independientes del sistema operativo. Este controlador está disponible como módulo.

Network device support

Solamente deberíamos responder que no, si la computadora en cuestión no tendrá una placa de red de ningún tipo y en caso de tener un modem sólo lo utilizaríamos para UUCP (Unix to Unix Communication Protocol) o ingreso a un BBS (Bulletin Board Service), ya que si planeamos conectarnos a Internet con el mismo deberíamos responder afirmativamente. Luego vienen una inmensa cantidad de controladores para Ethernet (10 y 100 Mbps), Token Ring, FDDI, PPP, SLIP, LAN inhalámbricas, interfaces WAN y dispositivos de red PCMCIA. Estos controladores están disponibles como módulos.

Amateur Radio support

Soporte para placas de radioaficionados que funcionan bajo protocolo AX.25. Si tiene una placa de este tipo deberá responder afirmativamente.

IrDA (infrared) support

Las Infrared Data Associations especifican estándares para comunicaciones infrarojas inhalámbricas y están soportadas por la mayoría de las laptops y PDA's (Personal Digital Assistants). Pueden ser compiladas como módulos.

ISDN subsystems

Las Redes Digitales de Servicios Integrados (Integrated Services Digital Networks) es un tipo especial de servicio telefónico digital con velocidades de 64 y 128 Mbps que se utilizan para transmisión de voz, datos y video en forma totalmente digital; es ideal para teleconferencias. Puede ser compilado como módulo.

Old CD-ROM drivers (not SCSI, not IDE)

Soporte para los viejos lectores de CD-ROM, antes de que aparecieran los IDE y los SCSI, de tecnología propietaria, y que a menudo venían en conjunto con una placa de audio, tales como Creative SoundBlaster, Panasonic, Sony, Matsushita, Mitsumi, Philips o Sanyo. Debería responder afirmativamente si tiene alguno de estos viejos lectores de CD-ROM. Pueden seleccionarse como módulos.

Input core support

Responda afirmativamente sólo si pretende habilitar alguna de las opciones de USB HID (Human Interface Device). Para mayor información vea http://www.linux-usb.org/

Character devices

Soporte para todos los dispositivos especiales por caracteres: terminales, ratones, joysticks, ciertas unidades de cinta lentas, *watchdog* que son placas para “despertar” la computadora en forma remota, algunas placas de video *on-board*, dispositivos especiales PCMCIA y una gran variedad más de dispositivos. Pueden seleccionarse como módulos.

Multimedia devices

Soporte para distintas placas capturadoras de audio o video, y placas de radio FM. Pueden seleccionarse como módulos.

Crypto Hardware support

Soporte para placas encriptadoras, por ejemplo Broadcom 5820 SSL. Pueden seleccionarse como módulos.

File systems

Soporte para distintos sistemas de archivos, tales como ext2, ext3, reiserfs, adfs, proc, Amiga FFS, BeOS BeFS, MSDOS, vfat, umsdos, jffs, iso9660, joliet, jfs, minix, ntfs, os/2, qnx4, udf, los de red: nfs, smb, ncp; y otros más. Pueden seleccionarse como módulos.

Console drivers

Controladoras para distintas consolas (por ejemplo VGA) en modo texto. Hay que responder afirmativamente a la que usaremos, que en el caso de una PC es la VGA.

Sound

Soporte para distintas placas de sonido: Creative, Ensoniq, ESS, Intel, S3, Trident, Pinnacle, VIA, OSS, y muchas más menos conocidas. Pueden seleccionarse como módulos.

USB support

Universal Serial Bus (USB) es una especificación para un subsistema de bus serial que ofrece mas altas velocidades y mas prestaciones que el puerto serial tradicional de la PC. Este bus suministra voltaje a los periféricos y permite el intercambio “en caliente” (*hot swap*). Se pueden conectar hasta 127 periféricos USB a un único puerto USB en una estructura de árbol. El puerto USB es la raíz del árbol. Debe responder afirmativamente si su máquina tiene un bus USB. Los controladores específicos pueden ser compilados como módulos.

Bluetooth support

Bluetooth es una nueva tecnología inhalámbrica de bajo costo, bajo consumo y corto alcance. Fue diseñada como reemplazo de cables y otras tecnologías de corto alcance como IrDA. Bluetooth opera en un rango de área personal (Personal Area Network, PAN) que típicamente se extiende hasta 10 metros. Para mayor información vea http://www.bluetooth.com/

Kernel hacking

Según el README de Linus Torvalds “la configuración de 'kernel hacking' usualmente resulta en un núcleo mas grande o mas lento (o ambos) y puede hacer el núcleo menos estable al configurar unas rutinas que activamente tratan de romper código malo para encontrar problemas en el núcleo. De modo que debería responder negativamente si quiere configurar un núcleo de producción”. Habría que acatar el consejo.

Library routines

Soporte para comprimir y descomprimir (zlib) en el núcleo. No es necesario para las aplicaciones que comprimen como gzip, bzip2, etc. No confundir.

Save and exit

Quit without saving

Load configuration from file

Store configuration to file

Las opciones para salir guardando los cambios o salir abandonando. Alternativamente se puede guardar la configuración en un archivo distinto del habitual, si usted quiere tener un respaldo o una versión especial, y posteriormente cargarla.

Una vez pasada esta etapa de configuración el programa indica "revise el archivo Makefile para opciones adicionales" creando un archivo Makefile. Es decir, que todo lo hecho hasta ahora, fue una pequeña "ayudita" para crear este archivo (y el oculto .config), que por otra parte, podríamos haberlo creado "a mano" con el editor de texto. Sólo haremos esto si necesitáramos realmente modificarlo a mano, pero en el 99.99% de los casos esto no es necesario. Sin embargo, es difícil resistir la curiosidad de al menos mirarlo.

Compilación del núcleo

Al terminar de configurar el programa indica las operaciones siguientes: "make dep" y "make clean", que son las que "hacen las dependencias" y "hacen la limpieza", respectivamente.

Ahora que está preparada la configuración, viene el momento de compilar, sus opciones son:

- #make zImage[38]
- #make bzImage
- #make zdisk
- #make bzdisk
- #make zlilo
- #make bzlilo

Básicamente hay tres opciones, sólo que cada una de ellas tiene (o no) la letra "b" por delante. La "b" (de *big*) al principio es para núcleos grandes, que no caben en el primer megabyte (MB) de memoria RAM. La "z" indica que está comprimido. Un núcleo comprimido se autodescomprime él mismo al arrancar.

Las primeras dos opciones son similares: compilarán el núcleo y lo dejarán en el subdirectorio arch/i386/boot/ con el nombre zImage o bzImage. Son los núcleos compilados.

Las dos siguientes (zdisk y bzdisk) además de compilarlo, se copiará a un diskette que previamente hayamos dejado (desprotegido) en la diskettera. Esto es muy útil para cuando estamos probando y no deseamos arruinar un arranque que está funcionando bien. Este diskette de arranque con el nuevo núcleo me permitirá arrancar desde el mismo, y si todo sale bien, podemos pasar a las opciones siguientes, que hacen cambios mas definitivos.

[38] Como la compilación del núcleo genera una gran cantidad de mensajes por pantalla, podemos guardar los mensajes a medida que los vemos, a través de una tubería y el comando **tee**, de esta manera:
#make zImage 2>&1 | tee archivo.txt

Las últimas dos opciones, luego de compilar el núcleo, lo copia a "/vmlinuz". Es decir, al directorio "raíz". Y ejecuta **lilo** para que la próxima vez que arranquemos, lo hagamos con este nuevo núcleo. Es muy importante ver los mensajes finales de **lilo**, que no indiquen ningún error. Las opciones de compilación sin la "b" (de *big*) tienden a desaparecer en el futuro.

Como este proceso tarda mucho, podemos enviar el trabajo al *background*:

nohup make bzImage &

Y obtener nuevamente el *prompt*, luego para ver como va la compilación:

tail -f nohup.out

El comando **nohup**, ejecuta un comando de manera inmune a la señal SIGHUP, es decir, que aunque salieramos de nuestra sesión **bash**, el comando a la derecha de **nohup** seguiría ejecutando hasta finalizar. Toda su salida, en vez de ir a pantalla, está redirigida al archivo nohup.out.

Instalación del núcleo

El paso anterior funcionará si **lilo** está bien configurado en su sistema, es decir, el núcleo es /vmlinuz, **lilo** está en /sbin, y su archivo de configuración en /etc/lilo.conf no tiene fallas ni incoherencias. Además renombra el núcleo anterior a /vmlinuz.old.

Si, por ejemplo, no está de acuerdo en dejar el núcleo en /vmlinuz, sino en el directorio /boot/ y con un nombre más adecuado, deberá instalar el núcleo "a mano", moviendo /vmlinuz a /boot/vmlinuz-2.4.2-2, por ejemplo, si ése fuera el número de versión. Y modificar /etc/lilo.conf como ya hemos visto antes:

```
image=/boot/vmlinuz-2.4.2-2
        label=nuevokernel
        ...
```

Y ejecutando **lilo** para que efectúe los cambios necesarios.

Trabajando con módulos

La órdenes

```
#make modules
#make modules_install
```

compilarán los módulos que hayamos seleccionado, y los instalarán en /lib/modules/(versión), es decir en nuestro ejemplo en /lib/modules/2.4.2-2/, creando la estructura de directorios necesaria. No olvidar ejecutar

#depmod -a

en cuanto hayamos arrancado con dicho núcleo (generalmente este comando está incluído en los *scripts* de arranque del sistema).

El núcleo de Linux está organizado siguiendo una arquitectura monolítica, en la cual, todas las partes del núcleo del sistema operativo (sistemas de archivos, manejadores de dispositivos o *device drivers*, protocolos de red, etc.) están enlazadas como una sola imagen (normalmente el archivo /vmlinuz) que es la que se carga y ejecuta en el arranque del sistema.

Esta estructura podría dar lugar a un sistema poco flexible, ya que cualquier funcionalidad que se le quisiera añadir al núcleo del sistema requeriría una recompilación completa del mismo. Aún así, la filosofía de fuentes libres hace a Linux mucho más flexible que otros sistemas operativos en la que los fuente no están disponibles.

No obstante, la recompilación total del núcleo puede resultar engorrosa en las fases de desarrollo de nuevos manejadores de dispositivo, ampliaciones no oficiales del núcleo, etc.

Esta limitación desapareció con la incorporación, en la versión 2.0 de Linux, del soporte para la carga dinámica de módulos en el núcleo. Esta nueva característica permite la incorporación "en caliente" de nuevo código al núcleo del sistema operativo, sin necesidad de reinicializar el sistema.

Los módulos son "trozos de sistema operativo", en forma de archivos objeto (.o), que se pueden insertar y extraer en tiempo de ejecución. Dichos archivos .o se pueden obtener directamente como resultado de la compilación de archivo .c :

```
# gcc -c prog.c
```

Una vez desarrollado un módulo e insertado en el núcleo, su código pasa a ser parte del propio núcleo, y por lo tanto se ejecuta en el modo supervisor del procesador (nivel de privilegio 0 en la arquitectura i386), con acceso a todas las funciones del núcleo, a las funciones exportadas por módulos previamente insertados, y a todo el hardware de la máquina sin restricciones.

La única diferencia con código enlazado en el núcleo es la posibilidad de extraer el módulo una vez que ha realizado su labor o ha dejado de ser útil, liberando así todos los recursos utilizados.

Los comandos para trabajar con módulos

Ya hemos hablado del comando **depmod**, que maneja las descripciones de dependencias para módulos cargables del núcleo. La configuración general de los módulos se encuentra en el archivo /etc/modules.conf, que puede contener por ejemplo:

```
alias sound sb
alias midi opl3
alias eth0 ne2k-pci
options opl3 io=0x388
options sb io=0x220 irq=5 dma=1 dma16=5 mpu_io=0x330
options ad1848 io=0x530 irq=11 dma=0,0
```

Las líneas "alias" le dan un nombre adicional a los módulos, por ejemplo, en la primera para dirigirnos al módulo **sb** (por SoundBlaster) podemos hacerlo por ese nombre o por el nombre "sound". El nombre "eth0" para la primera interfaz Ethernet, es un sinónimo del módulo "ne2k-pci". Esto es muy útil porque si nuestra placa NE2000 con bus PCI fuera nuestra tercera interfaz Ethernet podríamos haber puesto "eth2". Todas las directivas "options" especifican las opciones necesitadas por un módulo.

El comando **modprobe** hace un manejo de alto nivel de módulos cargables, intentando hacer un "sondeo" del mismo. Todos estos comandos que veremos leen el archivo de configuración de módulos /etc/modules.conf. Por ejemplo, el comando

```
#modprobe sound
```

Intentará cargar en memoria el módulo cargable para el manejo de la placa de audio SoundBlaster, de acuerdo con la configuración que le hemos dado en el archivo /etc/modules.conf, este comando es equivalente a:

```
#modprobe sb io=0x220 irq=5 dma=1 dma16=5 mpu_io=0x330
```

El comando **insmod** instala un módulo cargable del núcleo, intenta enlazar un módulo en el núcleo que se está ejecutando al resolver todos los símbolos de la tabla de símbolos exportada del núcleo.

El comando **lsmod** lista los módulos cargados, también podemos verlos con

```
#cat /proc/modules
```

Con el comando **rmmod** descarga módulos previamente cargados.

Estos son básicamente (hay otros) los comandos que manipulan los módulos cargables del núcleo. Sin embargo, hay un programa que ejecuta como *daemon* o "demonio" que se llama **kerneld**[39], y está encargado de la carga y descarga de módulos a medida que se los necesita.

La carga y descarga de módulos genera mensajes de registro (**logs** o bitácoras), que de acuerdo a cómo hayamos configurado al programa que administra los mensajes de error, precaución e información (programa **syslogd** y **klogd**), archivándolos en archivos específicos. Generalmente podremos ver estos **logs**, con el comando **dmesg**.

Notas bibliográficas

El sistema Linux es un producto de Internet; en consecuencia, la mayor parte de la documentación existente sobre Linux se puede obtener en alguna forma a través de Internet. Los sitios clave siguientes son referencia a la información más útil disponible:

- Las Páginas de Referencias Cruzadas de Linux en http://lxr.linux.no/ mantienen listados actualizados del núcleo de Linux, que pueden navegarse a través de la Web y contienen referencias cruzadas completas.
- Linux-HQ en http://www.linuxhq.com/ es sede de una gran cantidad de información relacionada con los núcleos Linux 2.0 en adelante. Este sitio también incluye vínculos a las páginas base de la mayor parte de las distribuciones, así como archivos de las principales listas de correo.
- El Proyecto de Documentación de Linux en http://www.tldp.org/ lista muchos libros sobre Linux que están disponibles en formato fuente como parte del Proyecto de Documentación de Linux. El proyecto también es sede de las guías Linux HOWTO: una serie de sugerencias y consejos relacionados con aspectos de Linux.
- La Kernel Hackers' Guide es una guía basada en Internet para los detalles internos del núcleo en general.

[39] Obsoleto.

Trabajo Dirigido Nº 1

1. Compilaremos el núcleo tal como se ha indicado, es decir, instalando los fuentes desde un paquete **tar**, **rpm** o **deb**.
2. Elejimos el procesador en el cual vamos a compilar o uno anterior, soporte para módulos, y dejamos en el núclo sólo las funcionalidades imprescindibles, el resto lo seleccionamos como módulo.
3. Al momento de compilar, podemos elejir las opciones que generan un diskette con el nuevo núcleo, para poder arrancar con él y no alterar el arranque actual.
4. Si funciona bien nuestro núcleo del diskette, podemos ahora generar un núcleo que reconozca **lilo** para que quede automáticamente configurado.
5. Si logramos arrancar bien, procedemos a compilar e instalar los módulos.

Trabajo Dirigido Nº 2

A continuación ofrecemos un sencillo programa – el clásico "Hola, mundo" de Brian Kernighan & Dennis Ritchie – pero desde el núcleo de Linux. La función init_module nos va a permitir inicializar el módulo al insertarlo en el núcleo (equivaldría a la función main de un programa en **C**). Complementariamente, cleanup_module se usará para liberar los recursos utilizados cuando se vaya a extraer.

```
/* hello.c
 * Copyright (C) 1998 by Ori Pomerantz
 *
 * "Hello, world" - the kernel module version.
 */

/* The necessary header files */

/* Standard in kernel modules */
#include <linux/kernel.h>   /* We're doing kernel work */
#include <linux/module.h>   /* Specifically, a module */

/* Deal with CONFIG_MODVERSIONS */
#if CONFIG_MODVERSIONS==1
#define MODVERSIONS
#include <linux/modversions.h>
#endif

/* Initialize the module */
int init_module()
{
  printk("Hello, world - this is the kernel speaking\n");

  /* If we return a non zero value, it means that
   * init_module failed and the kernel module
   * can't be loaded */
  return 0;
}

/* Cleanup - undid whatever init_module did */
void cleanup_module()
{
  printk("Short is the life of a kernel module\n");
}
```

Para compilar:
$ gcc -Wall -DMODULE -D__KERNEL__ -DLINUX -c hello.c
Se inserta como superusuario:
insmod hello.o
Y se extrae:
rmmod hello
El núcleo no dispone de salida estándar, por lo que no podemos utilizar la función printf(). A cambio, el núcleo ofrece una versión de ésta, llamada printk(), que funciona casi igual.

Autoevaluación

1. Investigue: ¿En qué costos extra se incurre al crear y planificar un proceso, en comparación con el costo de un hilo clonado?.
2. Investigue: ¿Qué ventajas tiene el enlazado dinámico (compartido) de bibliotecas en comparación con el enlazado estático? Cite dos casos en los que sea preferible el enlazado estático.
3. Compare el uso de *sockets* para trabajo con redes con el uso de memoria compartida como mecanismo para comunicar datos entre procesos dentro de un mismo computador. ¿Qué ventajas tiene cada método? ¿Cuándo podría ser preferible cada uno?
4. El código fuente de Linux está ampliamente disponible en forma gratuita por Internet o de proveedores de CD-ROM. ¿Qué implicaciones tiene esta disponibilidad para la seguridad del sistema Linux?
5. La arquitectura del núcleo de Linux es
    - monolítica
    - cliente-servidor
    - microkernel
6. Y esto es así porque:
    - el desempeño de una arquitectura monolítica es mucho mas eficiente
    - una ventaja del modelo cliente-servidor es su adaptabilidad para usarse en sistemas distribuidos.
    - Los sistemas operativos modernos tienden a quitarle lo más que se pueda al núcleo llevandolo a procesos de usuario para hacerlo mas eficiente y ordenado.
7. ¿Cuál es el sitio web oficial de los fuentes de Linux?
8. ¿En qué directorio deben quedar los fuentes?
9. ¿Cuales son las tres opciones en la línea de comando para la configuración del núcleo? (¿Cuales son las tres formas para configurarlo?)
10. Una vez que está configurado ¿cómo generamos las dependencias?
11. ¿Cómo compilamos una imagen grande comprimida y que quede almacenada en disquette?
12. ¿Qué son los "módulos"?
13. Si hemos desarrollado un módulo para el núcleo, debemos reiniciar el equipo para que tome los cambios
    - verdadero
    - falso
14. ¿Cómo compilamos e instalamos los módulos?
15. ¿En qué archivo encontramos la configuración general de los módulos?
16. ¿Con qué comando listamos los módulos que están cargados?
17. ¿Cómo puedo sondear un módulo, para ver si puede cargarse, por ejemplo si quisiera detectar mi placa SoundBlaster?
18. ¿Cómo puedo eliminar un módulo cargado en memoria, por ejemplo descargando el controlador de sonido SoundBlaster anteriormente cargado?