http://lsi.ugr.es

Diseño de Sistemas Operativos
5º Curso de Ingeniería Informática

Guión de Prácticas – Curso 2010-11

Práctica 1:

Configuración y compilación del kernel de Linux

Objetivo de la práctica:

El objetivo de la presente práctica es configurar y compilar un kernel de Linux utilizando para ello las diferentes herramientas que suministra el sistema. También, se verá como pasar un parche al sistema.

1 Introducción

Si bien hoy en día existen distribuciones GNU/Linux que dispone de kernels, o núcleos, con módulos precompilados que hace que el proceso de instalación y detección de hardware más fácil, debemos de recordar que una de las principales ventajas que nos ofrece Linux es poder adaptarlo específicamente a la máquina donde se va a ejecutar.

Además, nos puede interesar recompilar el núcleo para incluir manejadores de dispositivos de terceros, la aplicación de parches no oficiales, o la instalación de un nuevo núcleo no incluido en nuestra distribución.

2 Desarrollo de la práctica

Para la realización de la práctica se pide configurar y compilar un kernel 2.4 de Linux para una de las máquinas del laboratorio de la Escuela donde se realizan las prácticas en el que se eliminaran los subsistemas innecesarios de forma que sea lo más pequeño posible y reducir así el tiempo de compilación.

3 Material necesario

Las lista que sigue a continuación muestra los pasos genéricos para la construcción de un nuevo kernel. En nuestras máquinas, no son necesarios algunos pasos pues ya se ha realizado pero lo indicamos en el guión para su conocimiento. Los pasos son:

1. Obtener el código fuente (nuestra máquina ya lo tiene).
2. Preparar el árbol de archivos del código fuente (parte del trabajo esta realizado).
3. Configurar el código fuente.
4. Construir el kernel a partir del código fuente e instalar los módulos.
5. Reubicar el kernel.
6. Configurar y ejecutar LILO (no necesario en nuestros laboratorios, al final del guión esplicamos la especificidad de nuestro sistema).
7. Verificar el soporte de módulos.

8. Reiniciar el sistema y probar el nuevo kernel.

Es necesario llevar a cabo todos los pasos en el orden correcto. En algún momento puede ser necesario parchear el kernel en lugar de generar una nueva versión completa. Parchear el kernel es más sencillo de lo que parece y lo explicare más adelante. La utilización de un parche permite ahorrar gran cantidad de tiempo.

- **Paso 1 - Obtener el código fuente**

Si bien la distribución de Linux incluye el código fuente, puede que este no sea el más actual. La forma más directa de obtener la versión actual del kernel es el sitio http://www.kernel.org. Una vez localizada la versión más actual la descargaremos en */usr/src*. Podemos conocer la versión de nuestro sistema mediante la orden `uname -a`.

Debemos crear un directorio para el código fuente del kernel. Normalmente, es de la forma *linux-2.X.YY*, donde el número (2.XX.YY) indica la versión del kernel. Luego será necesario crear un enlace a este directorio llamada *linux* (`ln -l linux-2.4.22`). Si ya existe, es porque se ha instalado previamente el RPM, por lo que es recomendable eliminarlo antes de comenzar.

Para una instalación genérica necesitaremos suministrar todos los detalles del equipo en el que vamos a instalar el núcleo. Para ello, es conveniente disponer de los manuales del hardware que tenemos. También podemos conocer los detalles de los buses y dispositivos PCI del sistema mediante la orden */sbin/lscpi*.

Nota: Este paso no es necesario en nuestro laboratorio pues en la imagen que tenemos disponible ya tenemos instalados los fuentes del un kernel 2.4.

- **Paso 2 - Preparar el árbol de archivos del código fuente**

Completado el paso uno, construiremos el árbol de archivos del código fuente para poder utilizarlo. La orden para hacerlo va a depender del formato del archivo que hayamos descargado. El archivo con el código fuente puede tener el formato .tar.gz ó tar.bz2.

Si el archivo tiene el formato .tar.gz, la orden que debemos de utilizar es:

```
% tar xzfv linux-2.4.22.tar.gz
```

Para el formato tar.bz2 utilizaremos

```
% bzcat linux-2.4.22.tar.bz2 | tar xv
```

ó bien

```
% tar xvfj linux-2.4.22.tar.bz2
```

Ambas órdenes extraen el contenido de los respectivos archivos en el directorio al que apunta el enlace *linux*.

Ahora, cambiaremos al directorio (`cd linux`). Este directorio contiene varios subdirectorio, de los cuales el más importante, permite conocer más detalles del kernel, es *Documentation*. Es el primer lugar donde debemos acudir si deseamos tener más información sobre el kernel.

Para comenzar el proceso de compilación de un nuevo kernel es necesario ejecutar la siguiente orden:

```
% make mrproper
```

que configura el entorno para el kernel. Si se recompila el kernel no será necesario volver a ejecutar esta orden salvo que se cambie el tipo de kernel de uniprocesador a SMP, o a la inversa. Si sólo se parchea el kernel tampoco será necesario ejecutar esta orden. Esta orden elimina el archivo *.config* creado en el proceso de configuración, por tanto, si deseamos guardar la configuración de una construcción anterior no debemos ejecutar la orden, o guardar el archivo de configuración en otro lugar.

- **Paso 3 - Configurar el código fuente**

En este paso, debemos configurar las opciones deseadas para nuestro nuevo kernel. La lista de opciones es bastante grande por lo que suele ser recomendable confiar en las opciones por defecto, salvo que deseemos un kernel con características especiales. Si no se comprende algún parámetro es preferible dejarlo por defecto. El tema se complica debido a las dependencias entre parámetros.

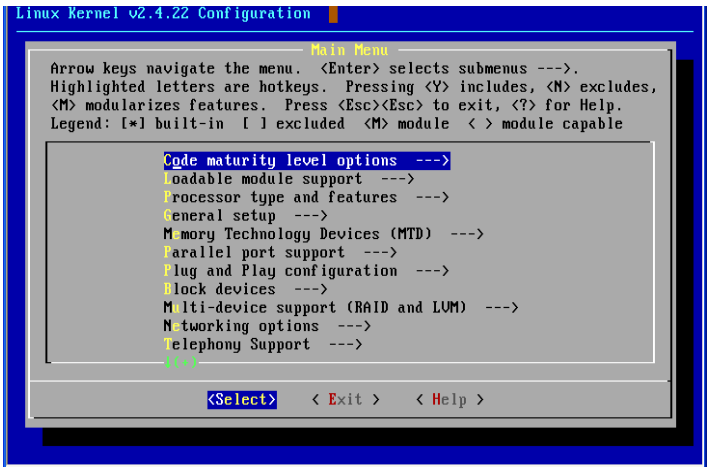
Los tres métodos alternativos para la configuración son:

```
% make config
% make menuconfig
% make xconfig
```

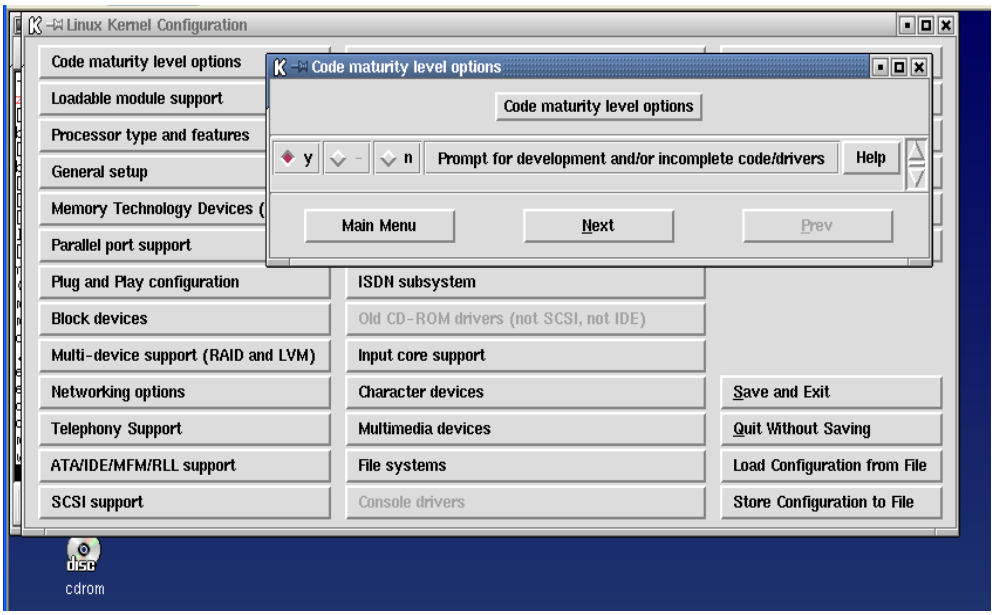
Los tres métodos realizan la misma tarea, crear el archivo *.config*, pero lo hacen de tres formas diferentes.

La primera, `make config`, es un script secuencial de texto que nos solicitando las diferentes opciones, en el que el principal inconveniente es que no podemos retroceder.

La segunda, `make menuconfig`, ya no es un script secuencial y no depende tampoco de ningún entorno de ventanas. Presenta un tipo especial de ventanas, denominadas *ncurses* ó *curses*, si tenemos instalada la biblioteca *ncurses*.



El último método, `make xconfig`, es el más fácil pues solo debemos seleccionar con el ratón los parámetros deseados pero para su uso debemos tener instalado el entorno de ventanas.



Cada opción de las que aparece en la lista permite dos o tres valores. Si indicamos:

- "y" (o "*" con *menuconfig*) se incluye en el kernel el soporte para esta opción,
- "n" (o "" con *menuconfig*) no se suministra el soporte;
- "m" indica que la funcionalidad correspondiente se suministra como módulo de carga dinámica (debemos saber que no todas las opciones están disponibles como módulos).

Sea cual sea el método, indicar que existen muchas opciones posibles. Nosotros citaremos algunas de las más críticas para arrancar el sistema:

- **Procesor type and features** (Processor family) – no deberá seleccionar un procesador superior al instalado.
- **General configuration** – se realizará de la siguiente forma:
  1. **PCI support** – obligatorio para un sistema PCI.
  2. **Sysctl support** – activado salvo en sistemas empotrados en lo que generalmente no dispondremos de usuarios.
  3. **Kernel core** – Debe tener formato ELF (con a.out no arrancará el sistema).
  4. **Kernel support for ELF binaries** – esta opción debe compilarse, no dejarse como módulo.
- **Block devices** – será necesario compilar cualquier dispositivo desde donde se arranque el kernel y que vaya a ser montado como una partición raíz (no es posible compilarlo como un módulo).
- **SCSI support** – es necesario compilarlo si se va a arrancar el sistema desde un dispositivo de este tipo.
- **Character devices** – deberemos seleccionar al menos los tres dispositivos siguientes: virtual terminal, support for console on virtual terminal, UNIX98 PTY support.
- **Mouse devices** – se seleccionará esta opción si disponemos de ratón (PS/2 o de bus) pero también se puede compilar como módulo.
- **Filesystems** – También es obligatorio compilar el soporte para los sistemas de archivos (no como módulo) tipo ext2 como el raíz. Se debe incluir soporte para los sistemas de archivos /proa y /dev/pts.
- **Console drivers** – deberemos compilar dos elementos para los controladores de la consola: VGA text console, y video mode selection support.

En el laboratorio podéis ver las diferentes opciones para familiarizarse con ellas. Esto también nos permitirá volver a las opciones adecuadas cuando falle el proceso de compilación y nos indique que parte del kernel estaba compilando.

Una vez configurado el núcleo, podemos modificar la variable `linux_banner` del archivo *init/version.c* para que aparezca un mensaje puesto por nosotros. De esta forma, conoceremos si es nuestro núcleo el que esta arrancando cuando finalicemos todos los pasos (si no lo vemos entre los cientos de mensajes del arranque podemos utilizar la orden `dmesg`). También, podemos saber si el sistema ha arrancado con nuestro kernel pues el banner incluye la fecha en la que el kernel ha sido construido (que debe ser la fecha actual).

- **Paso 4 - Construir el kernel a partir del código fuente e instalar los módulos**

Una vez definida la configuración, debemos realizar los siguientes pasos:

```
% make dep
% make clean
% make [ bzImage | bzdisk | bzlilo ]
% make modules
% make modules_install
```

El primer paso (1) crea los archivos de dependencias entre componentes,*.depend*. El segundo, (2) eliminará los pasos no finalizados de construcciones previas para que no interfieran con la

actual. Ojo: si no ejecutamos esta orden, no se borraran los archivos ".o" de compilaciones previas que no se hayan modificado y el proceso de compilación será más rápido.

Para la orden 3 (lista de arriba) existen tres posibilidades, para construir la partes estática del kernel, que pasamos a describir:

make bzImage – su misión es compilar nuevo un kernel. Es el paso normal. Si todo tiene éxito, podemos encontrar el kernel nuevo en */usr/src/arch/i386/boot/bzImage*. Para arrancar el sistema con este nuevo kernel, sólo nos quedaría ajustar el cargador (LILO ó GRUB) para que lo localice, o reubicar la imagen a una nueva posición, pero eso lo veremos después.

make bzdisk – Como la orden anterior, pero al finalizar al compilación copia el kernel resultante en un disquete. Este disquete se puede utilizar por ejemplo como disco de emergencia.

make bzlilo – Este es el método que se recomienda normalmente para crear e instalar un nuevo kernel pero necesita de una preparación previa de LILO, ya que puede sobrescribir el kernel "actual" con los riegos que esto conlleva. Con este método, al final de la compilación el kernel que acabamos de construir se denomina */vmlinux* (se hace una copia de seguridad del archivo *vmlinux* actual), y se ejecuta LILO para volver al instalarlo.

Las dos últimas órdenes que nos quedan son `make modules` y `make modules_install`. La primera de ellas construye los módulos de carga dinámica para aquellas partes que se indicaron como tales en la configuración, con "m". La segunda mueve los módulos a su ubicación final, es decir, */lib/modules/<versión kernel>/kernel/<tipo de modulo>*.

- **Paso 5 - Reubicar el kernel**

Una vez creado el kernel, debemos dejarlo en el directorio adecuado. Actualmente, se esta abandonando la política de situarlo en el directorio raíz, y se tiene a ubicarlo en el directorio /boot, junto con el archivo del mapa del kernel. Para ello, solo debemos ejecutar una simple copia:

```
% cp System.map /boot/System.map-2.4.YY
% cp arch/i386/boot/bzImage /boot/bzImage-2.4.YY
```

Nota: Ver Apartado 6 sobre las peculiaridades de la configuración de la Escuela

- **Paso 6 - Configurar y ejecutar LILO**

Damos aquí este paso por completitud, ya que en los equipos de la Escuela no es necesario ya que utilizamos un cargador especial, denominado Rembo.

El cargador de arranque nos permitirá especificar varias imágenes de arranque. Si el cargador es LILO, podemos insertar otra imagen de arranque insertando otra sección en el archivo /etc/lilo.conf. Por ejemplo, un ejemplo del contenido de este archivo es:

```
boot=/dev/sda
map=/boot/map
install=/boot/boot.b
prompt
timeout=50
message=/boot/message
linear
default=linux

image=/boot/vmlinuz-2.4.2-2
  label=linux
  initrd=/boot/initrd-2.4.2-2.img
```

```
  read-only
  root=/dev/sda5
```

Para añadir una nueva imagen, debemos copiar la entrada *image* (5 últimas líneas) y la etiqueta *label* para que contenga el nombre de la nueva imagen. Ahora, tendremos

```
boot=/dev/sda
map=/boot/map
install=/boot/boot.b
prompt
timeout=50
message=/boot/message
linear
default=linux

image=/boot/vmlinuz-2.4.22
  label=linux
  initrd=/boot/initrd-2.4.22.img
  read-only
  root=/dev/sda5

image=/boot/vmlinuz-2.4.2-2.viejo
  label=linux.viejo
  initrd=/boot/initrd-2.4.2-2.img
  read-only
```

Después de esta configuración, debemos volver a instalar el cargador de arranque con:

```
% lilo
Added linux *
Added Linux.viejo
```

Ya podemos reiniciar el sistema bien con el nuevo kernel bien con el anterior, dependiendo de la opción (nombre de la etiqueta) que le demos a LILO.

Si nos fijamos en la configuración, aparece la entrada *initrd=/boot/initrd-2.4.22.img*. La opción *initrd* se utiliza en aquellos casos que es necesario instalar ciertos manejadores de dispositivos antes de iniciar el arranque propio del kernel. Esto ocurre cuando por ejemplo arrancamos en modo gráfico.

- **Paso 7 - Verificar el soporte de módulos**

Deberemos comprobar que el archivo */etc/modules.conf* (en Red Hat) contiene los módulos que deseamos cargar en el arranque del sistema.

- **Paso 8 - Reiniciar el sistema y probar el nuevo kernel**

Ya podemos reiniciar el sistema (¡cruzamos los dedos!), y si hay algún problema, siempre podemos cargar la imagen original y comenzar de nuevo.

4 Como parchear un kernel

Tanto si aparece una versión nueva como si deseamos llevar a cabo una actualización del kernel actual, la solución más rápida y económica es parchear el sistema actual. Un parche es un archivo que modificará el código fuente del kernel de una versión a otra.

Existen diferentes tipos de parches:

- parches estables del kernel que se aplican a versiones base del kernel.
- Parches de versión del kernel base que se aplican a versiones previas del kernel base.
- Parches incrementales que modifican una versión a la siguiente.

Los parches estables y de kernel base los podemos encontrar en el mismo directorio que el árbol de código fuente. Los parches incrementales los solemos encontrar un nivel por debajo, en el directorio *incr*.

Para realizarlo, debemos descargar el parche que queremos aplicar. Los parches se aplican incrementalmente y de uno en uno (no podemos saltarnos ningún parche). Por ejemplo, deseamos pasar el parche *patch-2.4.23.bz2*.

Copiar el archivo *linux-2.4.23.bz* del parche en /usr/src y ejecutamos:

```
% bzcat patch-2.4.23.bz2 | patch –p0 –dry-run
```

Si la orden anterior no da errores (líneas `FAILED`) es posible eliminar la opción *–dry-run* y pasar definitivamente el parche.

Una ventaja de pasa así el parche es que no ahorramos los pasos de configuración y pasamos directamente a la construcción del kernel.

En ocasiones podemos necesitar eliminar un parche antes de poder pasar un parche del kernel, por ejemplo, eliminar un driver de una tarjeta de red. Para ello, ejecutamos la orden:

```
% patch –p1 –R <ethernet.patch
```

- **Qué es un parche y como construirlo**

Una vez visto cómo aplicar un parche, comentar que un parche es un archivo que describe las diferencias entre dos versiones de un archivo fuente. Para crearlo, utilizamos la herramienta *diff*, que comparar línea por línea las dos versiones de archivo e imprime las diferencias en un formato estándar. Este archivo es el que lee la utilidad *patch* y aplica los cambios a otra copia del archivo.

Por ejemplo, sean dos archivos que se diferencias en una línea:

```
val@evilcat <~>$ cat old/file.txt
  This
  is
  a
  simple
  file.
  val@evilcat <~>$ cat new/file.txt
  This
  is
  a
  slightly more complex
  file.
  val@evilcat <~>$ diff -uNr old new
  diff -uNr old/file.txt new/file.txt
  --- old/file.txt        Tue May 28 23:00:21 2002
  +++ new/file.txt        Tue May 28 23:01:01 2002
  @@ -1,5 +1,5 @@
   This
   is
   a
  -simple
  +slightly more complex
   file.
```

Como podemos observar el parche contiene información sobre que línea hay que quitar del fichero origen (-) y cual hay que añadir en el nuevo (+).

5 Herramientas necesarias para compilar el kernel

En general, la mayoría de las distribuciones permiten la instalación de los paquetes necesarios para la configuración y construcción de un kernel. Si nuestra distribución, lo permite, es más fácil instalarlos de ella que andar buscando los paquetes y las versiones correctas de los mismos en la red.

Solo son necesarios tres paquetes para construir un kernel:

- el compilador – debemos utilizar el compilador gcc. Suele estar en nuestra distribución pero sino es así podemos descargarlo de http://gcc.gnu.org. Debemos tener presente la versión del mismo, ya que podríamos no poder construir el kernel. Por ejemplo, para construir un kernel 2.6.18 la versión más vieja del compilador ha de ser la 3.2. A veces, tampoco es recomendable hacerlo con la más moderna.
- El enlazador – el compilador necesita para completar su labor un conjunto de herramientas que reciben el nombre conjunto de binutils que realizan en enlazado ensamblador de los archivos. Estas utilidades se encuentran en el paquete binutils de la distribución, o podemos descargarlas de http://www.gnu.org/software/binutils.
- Make – es la herramienta que guía el proceso de compilación. Podemos encontrar esta herramienta en la distribución o en http://www.gnu.org/software/make.

Nota de versiones: Todo el proceso de compilación descrito es válido para kernel 2.4, para kernel 2. 6 tanto el proceso como las herramientas necesarias para la compilación varian respecto a lo descrito.

Por último, comentar que si bien en general la versión del kernel que estemos usando no afecta a ninguna aplicación de usuario hay un pequeño conjunto de herramientas que si se ven afectadas por la versión. De esta forma, si modificamos la versión del kernel, necesitamos modificar también estos paquetes. Estos son:

- Paquete util-linux – conjunto de utilidades diferentes que van desde funciones relacionadas con el manejo de disco hasta el reloj. Podemos obtenerlo de http://www.kernel.org/pub/linux/utils/utils-linux.
- Module-init-tools – necesario si deseamos usar módulos del kernel de linux. Podemos obtenerlo de http://www.kernel.org/pub/linux/utils/kernel/module-init-tools.
- Herramientas específicas para sistemas de archivos particulares, como ext2, ext3 y ext4, JFS, ReiserFS, XFS, Quotas, y NFS
- Otras herramientas como pueden ser udev, procps y pcmiatools.

6 Adaptación a los equipos de prácticas de la Escuela

Los equipos de nuestra Escuela arrancan las imágenes de los sistemas con un sistema denominado *Rembo* que exige algunas peculiaridades en la forma de arrancar la imagen del kernel generado en las prácticas. Básicamente las que nos afectan son:

(a) no se utiliza Lilo para el arranque por lo que no será necesario configurarlo, y

(b) la imagen de Linux que generamos en el Paso 5 de Apartado 3 "Reubicar el kernel" debe tener un nombre específico para poder arrancar en local.

Estas peculiaridades, además de los códigos para utilizar la imagen adecuada como el *password* para poder trabajar como *root* los veremos en el laboratorio de prácticas.

7 Bibliografía y referencias de interés

- Kwan Love, "Kernel Rebuild Guide", en http://www.digitalhermit.com/~kwan/kernel.html. Como parte de *The Linux Documentation Project*, en http://tldp.org/docs.html#howto.
- Fernando Sánchez y Rocío Arango, *El kernel 2.4 de Linux*, Prentice Hall, 2003.
- Greg Kroah-Hartman, *Linux Kernel in a Nutshell. A Desktop Quick Reference*, O'Reilly, 2007