

Compilación del núcleo de GNU/Linux

Objetivo:

El objetivo principal de este apunte es que el alumno aprenda a configurar y compilar un kernel GNU/Linux utilizando para ello las diferentes herramientas existentes con el fin de adaptar el sistema operativo a sus necesidades y adecuarlo a las funciones que el mismo cumplirá.

Introducción:

Si bien hoy en día existen distribuciones GNU/Linux que disponen de kernels, o núcleos, con módulos precompilados que hacen que el proceso de instalación y detección de hardware sea más fácil, debemos de recordar que una de las principales ventajas que nos ofrece GNU/Linux es la de 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.

Puede que aún alguien dude de porqué es necesario compilar el kernel. Para aclarar un poco más la situación se propone este ejemplo:

Comparemos el kernel GNU/Linux con un automóvil. Podemos ir al concesionario y comprar un modelo estándar proporcionado por el fabricante. Sin embargo, nosotros somos muy selectivos y tenemos absolutamente claro qué es lo que queremos y qué no. Para ello, luego de haber adquirido el automóvil, modificamos su frente, dándole un estilo más agresivo, cambiamos todos sus faros, y como si esto fuera poco le agregamos llantas deportivas. Tras un tiempo tendremos un auto de acuerdo a nuestros gustos y necesidades. Pues bien, este proceso es exactamente el mismo que realizamos cuando compilamos el kernel.

Debemos ser conscientes de que el mundo es muy grande y la gente posee máquinas muy dispares. Como informáticos, preferimos tomar una idea general y adaptarla a nuestras necesidades que tomar algo muy particular que luego no nos sirva. Este proceso es conocido como ajuste o tuning y consiste en adaptar el software al hardware sobre el que se ejecuta.

Qué explicaremos en este apunte:

A lo largo de este manual, nos encontraremos con un caso de ejemplo todos los aspectos que tendremos que tener en cuenta a la hora de encarar la recompilación de un nuevo núcleo. Para ello configuraremos y compilaremos la versión 2.6.18 del kernel de GNU/Linux obtenida de su sitio oficial <http://www.kernel.org> explicando paso a paso todas las opciones de este proceso.

Herramientas necesarias para compilar el kernel:

Las siguientes son las utilidades con las que debemos contar para compilar el kernel, así como las que podamos necesitar para su utilización posterior.

Lo primero, y más importante, es disponer del código fuente completo del kernel que queremos compilar. Normalmente buscaremos la versión más actual que estará más pulida, incluye nueva funcionalidad o soporta más dispositivos hardware. El código fuente del núcleo de GNU/Linux lo podemos encontrar en su página oficial <http://www.kernel.org>.

A continuación se detalla la lista de programas necesarios para la compilación del núcleo a partir del código fuente, así como la versión mínima que debemos disponer (con versiones mayores funcionará de todos modos):

- Compilador C de GNU 2.95.3 `gcc`.
- Utilidad `Make` de GNU 3.77.
- Ensamblador, enlazador, y otras utilidades GNU: `binutils 2.9.1.0.25`. Puesto que actualmente se utiliza el ensamblador `gas`, se necesita una versión bastante moderna de este paquete de utilidades.
- Archivos de cabeceras y bibliotecas de desarrollo de GNU, `libc6` (el paquete se denomina `libc6-dev`).
- Archivos de cabeceras y bibliotecas de desarrollo de `ncurses`. Solo necesarios si queremos realizar la configuración con `make menuconfig`. La versión empaquetada se denomina `libncurses5-dev`.
- El procesador de macros de GNU C o preprocesador de C, `cpp`. Necesitamos la misma versión que la que usemos de `gcc`.

La mayoría de las herramientas necesarias para la compilación del núcleo son GNU, por ello se habla a menudo de GNU/Linux.

Preparándonos para la compilación:

Es necesario evitar dos grandes errores a la hora de realizar este proceso:

1. Tener miedo de las consecuencias
2. Tomárselo a la ligera (para aquel que esté acostumbrado a pulsar `enter` antes de leer, será mejor que pierda el hábito).

Pasos necesarios:

Los pasos para la construcción de un nuevo kernel son los siguientes:

1. Obtener el código fuente.
2. Preparar el árbol de archivos del código fuente.
3. Configurar el código fuente.
4. Construir el kernel a partir del código fuente e instalar los módulos.
5. Creación del disco RAM inicial o `initrd`
6. Reubicar el kernel.
7. Configurar y ejecutar el gestor de arranque (LILO, GRUB, etc).
8. Reiniciar el sistema y probar el nuevo kernel.

Es necesario llevar a cabo todos los pasos en el orden correcto.

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

Si bien la gran mayoría de las distribuciones 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> (o algunos de sus mirrors). Una vez localizada la versión más actual, la descargaremos en `/usr/src`. Podemos conocer la versión de nuestro kernel actualmente instalado mediante el comando `uname -a`. Cuando se descarga el código fuente de un kernel, es una práctica habitual ubicarlo dentro del directorio `/usr/src`.

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 utilizando el comando `/sbin/lspci`. Es aconsejable anotar los datos provistos en la salida de este comando, ya que serán los que nos provean la información acerca del modelo y fabricante del hardware que tengamos instalado.

Un aspecto importante al momento de encarar la compilación de nuestro kernel es estar logueados en el sistema como el superusuario `root`.

Para este apunte supondremos que la versión del kernel que compilaremos es la 2.6.18 descargada del sitio oficial y que el nombre del archivo descargado es `linux-2.6.18.tar.bz2`

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

Completado el paso uno, construiremos el árbol de archivos del código fuente para poder utilizarlos. 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-2.6.18.tar.bz2
```

Para el formato `tar.bz2` utilizaremos

```
% bzipcat linux-2.6.18.tar.bz2 | tar -xv
```

ó bien

```
% tar xvfj linux-2.6.18.tar.bz2
```

Ambas órdenes extraen el contenido de los respectivos archivos en el directorio actual. Por lo cuál tendremos que tener en cuenta que nuestro archivo con los fuentes del kernel deberá estar ubicado en el directorio `/usr/src`.

Ahora, ingresaremos al directorio creado por el descompresor donde se alojan los archivos del código fuente del kernel (`cd linux-2.6.18`). Este directorio contiene varios subdirectorios, de los cuales el más importante, que permite conocer detalles del kernel, es `Documentation`. Es el primer lugar donde debemos acudir si deseamos tener más información sobre el kernel que manipularemos.

➤ **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

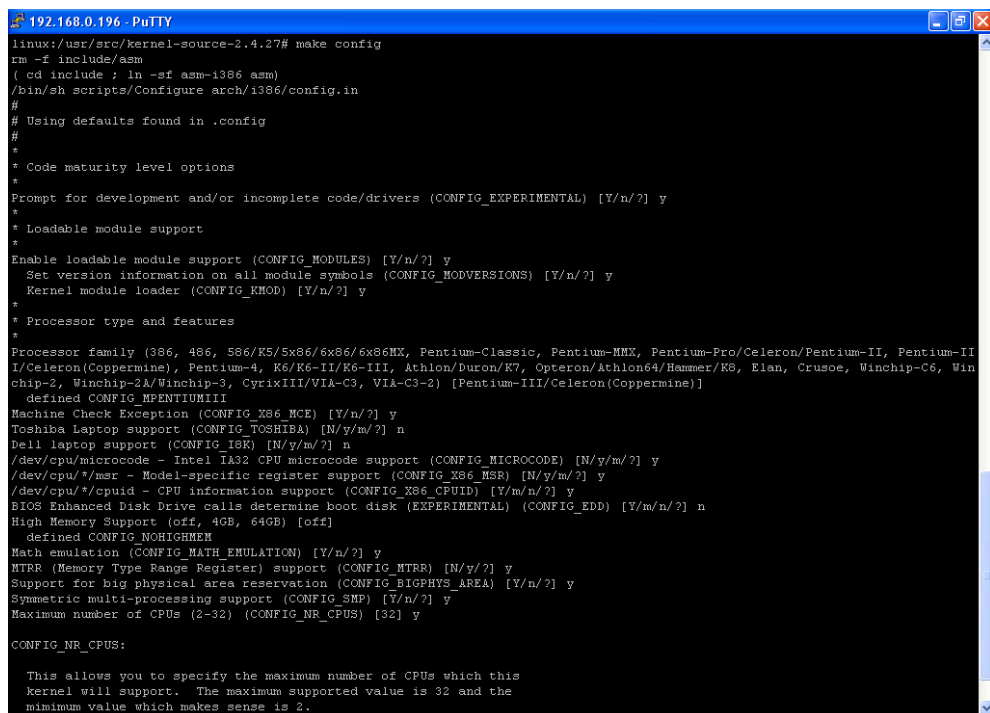
comprende algún parámetro es preferible dejarlo por defecto. Para la gran mayoría de los parámetros, existe una ayuda que indica la funcionalidad que el mismo provee.

Los tres métodos para la configuración son:

```
# make config
# make menuconfig
# make xconfig
```

Los tres métodos realizan la misma tarea, crear el archivo `.config` (que es el archivo de donde se leerán las opciones seleccionadas al momento de compilar el kernel), pero lo hacen de tres formas diferentes.

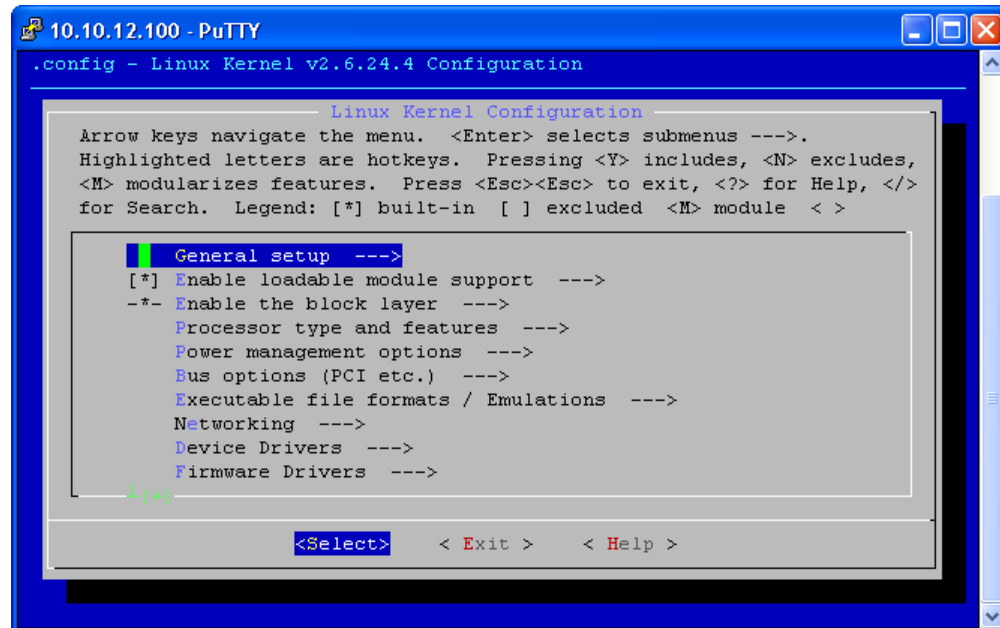
La primera, `make config`, es un script secuencial que se ejecuta en modo texto que nos va solicitando las diferentes opciones. El principal inconveniente es que no podemos retroceder.



```
192.168.0.196 - PuTTY
linux:/usr/src/kernel-source-2.4.27# make config
rm -f include/asm
(cd include ; ln -sf asm-i386 asm)
/bin/sh scripts/Configure arch/i386/config.in
#
# Using defaults found in .config
#
*
* Code maturity level options
*
Prompt for development and/or incomplete code/drivers (CONFIG_EXPERIMENTAL) [Y/n/?] y
*
* Loadable module support
*
Enable loadable module support (CONFIG_MODULES) [Y/n/?] y
Set version information on all module symbols (CONFIG_MODVERSIONS) [Y/n/?] y
Kernel module loader (CONFIG_KMOD) [Y/n/?] y
*
* Processor type and features
*
Processor family (386, 486, 586/K5/5x86/6x86/6x86MX, Pentium-Classic, Pentium-MMX, Pentium-Pro/Celeron/Pentium-II, Pentium-II
I/Celeron(Coppermine), Pentium-4, K6/K6-II/K6-III, Athlon/Duron/K7, Opteron/Athlon64/Hammer/K8, Elan, Crusoe, Winchip-C6, Win
chip-2, Winchip-2A/Winchip-3, CyrixIII/VIA-C3, VIA-C3-2) [Pentium-III/Celeron(Coppermine)]
defined CONFIG_MPENTIUMIII
Machine Check Exception (CONFIG_X86_MCE) [Y/n/?] y
Toshiba Laptop support (CONFIG_TOSHIBA) [N/y/m/?] n
Dell laptop support (CONFIG_I8K) [N/y/m/?] n
/dev/cpu/microcode - Intel IA32 CPU microcode support (CONFIG_MICROCODE) [N/y/m/?] y
/dev/cpu/*/msr - Model-specific register support (CONFIG_X86_MSR) [N/y/m/?] y
/dev/cpu/*/cpuid - CPU information support (CONFIG_X86_CPUID) [Y/n/m/?] y
BIOS Enhanced Disk Drive calls determine boot disk (EXPERIMENTAL) (CONFIG_EDD) [Y/n/m/?] n
High Memory Support (off, 4GB, 64GB) [off]
defined CONFIG_NOHIGHMEM
Math emulation (CONFIG_MATH_EMULATION) [Y/n/?] y
MTRR (Memory Type Range Register) support (CONFIG_MTRR) [N/y/?] y
Support for big physical area reservation (CONFIG_BIGPHYS_AREA) [Y/n/?] y
Symmetric multi-processing support (CONFIG_SMP) [Y/n/?] y
Maximum number of CPUs (2-32) (CONFIG_NR_CPUS) [32] y

CONFIG_NR_CPUS:
This allows you to specify the maximum number of CPUs which this
kernel will support. The maximum supported value is 32 and the
minimum value which makes sense is 2.
```

La segunda y más utilizada, `make menuconfig`, ya no es un script secuencial y depende tampoco de un entorno de ventanas provisto por un conjunto de librerías llamadas `ncurses` ó `ncurses`. Si bien este entorno corre en el modo texto o consola de Linux, es necesario contar con la biblioteca `ncurses`.



El último método, `make xconfig`, es el más fácil pues solo debemos seleccionar con el mouse los parámetros deseados pero para su uso debemos tener instalado el entorno de ventanas (xwindow, kde, gnom, etc).



En nuestro caso realizaremos la configuración del kernel a través de `make menuconfig`. Para movernos sobre este menú lo haremos utilizando las teclas del cursor, la tecla enter para entrar a cada subnivel y la barra espaciadora para seleccionar cada opción (1 vez presionada: suministrar opción como módulo, 2 veces: embeber en el kernel, 3 veces: deseleccionar)

Cada opción de las que aparece en la lista permite dos o tres valores. Si indicamos `<*>` se incluye en el kernel el soporte para esta opción; indicando `< >` no se suministra el soporte; en algunos casos es posible la opción `<m>` lo que indica que la funcionalidad correspondiente se suministrará como módulo (recordar que todas las opciones no están disponibles como módulos).

El menú de configuración del Kernel 2.6 se encuentra más ordenado y es más entendible que el del Kernel 2.4. A continuación citaremos algunas de las opciones más importantes que encontraremos en el menuconfig:

- ✓ Enable Loadable module support: Los módulos del kernel son pequeños programas compilados que pueden ser agregados o removidos del kernel que actualmente tenemos en funcionamiento. Los mismos pueden ser controladores de dispositivos, manejadores de File Systems, etc y proveen funcionalidades adicionales al kernel sin la necesidad de que este sea recompilado.

En esta pantalla seleccionaremos las 3 opciones posibles

- ✓ Procesor type and features: Se setean las opciones del procesador que tenemos instalado. En esta sección es donde le diremos al compilador de nuestro kernel para qué tipo de procesador debe compilar el mismo. Por ejemplo, si seleccionamos que tenemos un procesador "Pentium Pro", el `gcc` va a compilar las llamadas a función utilizando instrucciones especiales que apilan y desapilan varios registros a la vez, cosa que los procesadores anteriores no tenían. Si elegimos un procesador "Pentium 4", se van a compilar, por ejemplo, rutinas de RAID que usan instrucciones de SSE2, exclusivas de ese procesador y que son mucho más rápidas que las que usan instrucciones MMX (Pentium MMX y superiores), o SSE1 (Pentium II, Athlon, etc). Otros procesadores, carecen de ciertos componentes de la CPU, como el coprocesador matemático; entonces el código que se genera de ese kernel se encarga de incluir el emulador de coprocesador por software.
- ✓ Networking: En este menú encontraremos todas las opciones del Kernel referentes a dispositivos de red soportados. Esta opción deberá estar seleccionada en nuestro sistema aunque no tengamos pensado utilizarlo en Red. Muchas aplicaciones necesitan hacer referencia a direcciones de red (conocidas generalmente como de loopback) para su funcionamiento, y esto se da seleccionando esta opción.
- ✓ Device Drivers: En este menú encontraremos opciones diversas para dar soporte a puertos paralelos, dispositivos de bloques, soporte para dispositivos SCSI y SATA, soporte para dispositivos de red, usb, dispositivos de video, et.

- ✓ **Filesystems:** Ingresando a este menú daremos soporte en el kernel a los sistemas de archivos que utilicemos para almacenar nuestra información y aquí haremos un especial hincapié. Notar que existen diversos tipos de File Systems soportados en el kernel GNU/Linux. Dentro de los más importantes encontraremos: `ext2`, `ext3`, `DOS fat`, `vfat`, `ISO 9660`, `NTFS`, etc. Se deberá incluir también soporte para los sistemas de archivos `/proc` (se trata de una interfaz por medio de ficheros con la tabla de procesos del núcleo, usada por programas como `ps`) y `/dev/pts`. Tengamos en cuenta que si no se da soporte para el sistema de archivos que utiliza nuestro Sistema Operativo, no podremos arrancarlo. Otra opción que se puede habilitar en este menú es el soporte de cuotas (no todos los FileSystems las soportan).

Dentro del menú de configuración encontraremos muchas mas opciones. Todas ellas pueden ser accedidas y consultadas a través de su menú Help, acerca de las características que agregan al Kernel.

Una vez seleccionadas las opciones que queremos para la compilación, saldremos del menú de configuración a través de la opción `exit`, lo cuál nos solicitará que guardemos los cambios realizados en un archivo de configuración. Por defecto este archivo se llama `.config` y se almacenará en el directorio `/usr/src/ linux-2.6.18` (notar que para verlo, deberá utilizar el comando `ls -a` ya que el mismo es un archivo oculto)

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

Una vez definida la configuración, debemos correr las siguientes instrucciones en la línea de comandos:

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

El comando `make` es el que realiza la compilación del código fuente de nuestro Kernel. El mismo generará el archivo binario con la imagen de nuestro Kernel Si la ejecución del

mismo no arrojó ningún error, entonces en el directorio `arch/i386/boot/` encontraremos un archivo llamado `bzImage` que corresponde a la imagen del kernel

El segundo comando (`make modules`) creará los binarios correspondientes a los módulos que hayamos seleccionado. Finalmente, el tercer comando copiará todos los módulos del que hayamos seleccionado y hayan sido compilados al directorio `/lib/modules/2.6.18`

➤ Paso 5 – Creación del disco RAM inicial o initrd:

Para minimizar la cantidad de código que se carga en memoria, y para una modularidad máxima, el kernel GNU/Linux omite mucho código necesario para cargar el sistema operativo. Parte de este código se encuentra en módulos del kernel, mientras que el resto son aplicaciones en el espacio de usuario.

Para realizar un sistema de arranque que pueda servir para hardware variado, o que pueda ser cargado a partir de distintos medios incluyendo medios virtuales y transitorios como los provistos por una conexión de red, suele ser necesario para el kernel acceder al espacio de usuario y quizás a módulos del kernel de manera que pueda obtener los datos y rutinas necesarias para acceder al almacén de datos principal. El sistema de arranque `initrd` ha sido la principal solución a este problema desde hace mucho tiempo.

En el sistema `initrd`, los archivos que necesitan ser accedidos por el kernel durante el arranque se almacenan en un disco RAM, cuyos contenidos se encuentran en un sistema de archivos sobre un archivo montado como un dispositivo de bucle, o más históricamente, en un pequeño dispositivo montable como un disquette, y ocupa generalmente entre 1,4 MB y 4 MB. El sistema de archivos está comprimido con `gzip`. La localización de la imagen de este disco RAM se da al kernel en la carga por el cargador de arranque (generalmente `LILO` o `GRUB`). Este sistema de archivo es montado por el kernel para su carga y luego es desmontado una vez que el kernel ha finalizado de cargarse.

Para la creación de este disco de arranque utilizaremos el siguiente comando que recibe como primer parámetro el nombre del archivo de salida (el disco RAM a crear) y como segundo parámetro a versión del nuevo kernel.

```
mkinitramfs -o /boot/initrd.img-2.6.18 2.6.18
```

➤ Paso 6 - 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 tiende a ubicarlo en el directorio `/boot`, junto con el archivo del mapa del kernel. Para ello, solo debemos ejecutar una simple copia:

```
# cp arch/i386/boot/bzImage /boot/vmlinuz-2.6.18
# cp System.map /boot/System.map-2.6.18
```

El primer comando copia la imagen compilada del kernel a la ubicación en la cual se la accederá al momento del booteo.

El segundo comando copia el archivo de mapa del sistema al mismo directorio de booteo, ahora... *¿qué es el archivo de mapa del sistema?*:

El kernel, en vez de utilizar "symbol names" (por ejemplo `do_linuxrc`) prefiere utilizar direcciones "addresses" cuando se está ejecutando. (por ejemplo `c0105160`).

El núcleo de GNU/Linux está principalmente escrito en C. El compilador nos permite que usemos symbol names cuando escribimos y permite al kernel usar direcciones "addresses" cuando esta corriendo.

Hay ocasiones en que necesitamos saber cual es la dirección de un symbol name o al revés. Esto se logra con una tabla de símbolos denominada "symbol table"

Por ejemplo:

```
c0105000 T __stext
c0105000 T stext
c0105000 t rest_init
c0105030 t init
c0105160 t do_linuxrc
c0105270 T prepare_namespace
c01053b0 t huft_build
c0105960 t huft_free
c0105990 t inflate_codes
```

Se puede ver claramente que la variable con nombre `inflate_codes` está en la dirección del kernel `c0105990`.

El `System.map` es un archivo que se utiliza como "symbol table". Cada vez que se compila un kernel puede que las direcciones de varios "symbol names" estén obligadas a cambiar.

También existe otro "archivo" (lo pongo entre "comillas" porque realmente no es un archivo) en el sistema que se utiliza como "symbol table", es `/proc/ksyms`, este "archivo" es creado cada vez que el kernel bootea.

Un ejemplo claro de la utilización del `System.map` es cuando se produce un "oops" (error) en el kernel (producido por algún bug) y nos imprime algo así:

```
EIP: 0010:[<00000000>]
Call Trace: [c0105990>]
```

El `klogd` (kernel logging deamon) utiliza el `System.map` para traducir los "oops". Es decir, el `klogd` toma la infamación útil del EIP (por ejemplo `c0105990`) y la resuelve contra el `System.map` para luego pasarla al `syslogd` y que podamos leerla y saber de que se trata, así poder reportarla y arreglarla.

Otros programas que usan el `System.map` son `lsof`, `ps`, etc.

Si no se tiene un `System.map` correcto lo que puede llegar a ocurrir es que en los mensajes de logs veamos mensajes como "System.map does not match actual kernel", nada crítico, pero nos puede impedir la rápida resolución de un problema sencillo.

➤ Paso 7 - Configurar y ejecutar el gestor de arranque:

Llegado a este punto, nuestro nuevo kernel ya se encuentra compilado y listo para ejecutarse en la ubicación correspondiente. Solo nos resta indicarle a nuestro gestor de arranque como ejecutarlo. Veamos en este punto, entonces, que en nuestra máquina podemos tener varios kernels instalados y al momento de bootear nuestro equipo seleccionaremos con qué kernel deseamos arrancar. Vayamos entonces a la configuración del gestor de arranque GRUB que es el más utilizado hoy día:

Para realizar la configuración del mismo, editaremos el archivo `/boot/grub/menu.lst` (en algunas distribuciones de Linux este archivo puede denominarse `/etc/grub.conf`).

Un ejemplo de este archivo sería:

```
default          0
timeout          5
```

```
color cyan/blue white/blue
```

```
title          Linux-kernel-2.4.2
root           (hd0,0) #Disco en el 1° IDE, 1° partición
kernel         /vmlinuz-2.4.2-2 root=/dev/hda2 ro
initrd         /initrd.img-2.4.2-2

savedefault
boot
```

Para añadir nuestra nueva imagen del kernel, tendremos que copiar las líneas `title`, `root` y `kernel` y modificarlas con la información de nuestro nuevo archivo de imagen del kernel, con lo cuál nuestro nuevo archivo de configuración quedará de la siguiente manera:

```
default        0
timeout        5
color cyan/blue white/blue
```

```
title          Linux-kernel-2.4.2
root           (hd0,0) #Disco en el 1° IDE, 1° partición
kernel         /vmlinuz-2.4.2-2 root=/dev/hda2 ro
initrd         /initrd.img-2.4.2-2
```

```
title          Linux-kernel-2.6.18
root           (hd0,0) #Disco en el 1° IDE, 1° partición
kernel         /vmlinuz-2.6.18 root=/dev/hda2 ro
initrd         /initrd.img-2.6.18
```

```
savedefault
boot
```

Ya estamos en condiciones de reiniciar el sistema con el nuevo kernel o bien con el anterior, dependiendo de la opción (nombre de la etiqueta) que seleccionemos en el gestor de arranque, aunque aún nos queda realizar un chequeo más.

Tengamos en cuenta, y este es muy importante, que en ninguna ocasión se hizo referencia a la eliminación del archivo de imagen del kernel anterior. No se habló de eliminarlo físicamente ni tampoco lógicamente (es decir, su entrada del gestor de arranque). Esto nos permitirá que ante cualquier error que surja de la compilación de nuestro nuevo kernel que no podamos bootear con la imagen del kernel anterior (que sabemos que funciona) y corregir el inconveniente.

➤ Paso 8 - Reiniciar el sistema y probar el nuevo kernel:

Ya podemos reiniciar el sistema. En caso de que surja algún problema, volveremos a bootear el mismo seleccionando la imagen anterior del kernel. Es por esto que se recomienda NUNCA borrar la imagen anterior del kernel, hasta que estemos seguros de que la nueva imagen funciona como corresponde.

Un poco más acerca de los módulos:

Para lograr configurar un dispositivo controlado por un módulo, podemos utilizar las herramientas del paquete `modutils` o `modconf` para:

1. Asegurarnos que no haya conflictos entre el dispositivo con otros y eventualmente conocer la configuración que usa (algunos controladores autodetectan la configuración del dispositivo, pero no todos).
2. Encontrar un módulo que pueda manejar el dispositivo
3. Eventualmente pasar opciones al módulo de acuerdo a la configuración del dispositivo (por ejemplo IRQ o dirección base).

Los comandos que utilizaremos para la manipulación de los módulos son:

- ✓ lsmod: Lista los módulos cargados. De cada uno presenta nombre, tamaño, cuenta de usos y lista de módulos que lo usan (es equivalente a ejecutar el comando `cat /proc/modules`).
- ✓ rmmod modulo: Descarga uno o más módulos cargados, mientras estos no estén siendo usados. Con la opción `-r` intenta descargar recursivamente módulos de los cuales el módulo especificado dependa. El comando `rmmod -a` descarga todos los módulos que no estén siendo usados.
- ✓ `insmod modulo [opciones]`: Trata de cargar el módulo especificado. Pueden pasarse opciones específicas para el módulo, a continuación del nombre con la sintaxis `símbolo=valor` (los símbolos posibles dependen del módulo, pueden verse algunos en `/usr/share/modconf/descr.gz` que es la ayuda presentada por `modconf`. Puede indicarse una ruta no estándar para buscar módulos estableciéndola en la variable `MODPATH` o en `/etc/modules.conf`. Dado que los módulos se enlazan directamente con el kernel, deben ser compilados para una versión precisa, con la opción `-f` puede evitarse el chequeo de versiones

- ✓ depmod: Como un módulo puede requerir otros, hay dependencias que deben respetarse al cargar y descargar módulos. `depmod` permite calcular tales dependencias entre varios módulos o entre todos los disponibles con la opción `-a`. Por defecto `depmod -a` escribe las dependencias en el archivo `/lib/modules/version/modules.dep`. Cada línea de ese archivo tiene el nombre de un módulo seguido del carácter ':' y los módulos de los cuales depende, separados por espacios.
- ✓ modprobe módulo opciones: Emplea la información de dependencias generada por `depmod` e información de `/etc/modules.conf` para cargar el módulo especificado, cargando antes todos los módulos de los cuales dependa. Para especificar el módulo basta escribir el nombre (sin la ruta, ni la extensión `.o`) o uno de los alias definidos en `/etc/modutils/alias` (o en otro archivo del directorio `/etc/modutils`). Si hay líneas `pre-install` o `post-install` en `/etc/modules.conf`, `modprobe` puede ejecutar un comando antes y/o después de cargar el módulo. Como opciones para cargar el módulo usa prioritariamente las dadas en la línea de comandos y después las especificadas en líneas de la forma `options módulo opciones` en el archivo `/etc/modules.conf`.

Se pueden utilizar estos programas para configurar los módulos y se pueden hacer permanentes los cambios, agregando el módulo y las opciones en el archivo `/etc/modules`.

Para hacer más fácil la configuración de módulos, algunas distribuciones ofrecen las siguientes herramientas:

- ✓ modconf: Para listar, cargar y descargar módulos con menús. Este programa muestra los módulos disponibles en categorías y con ayudas sobre su uso y permite cargarlos o descargarlos del kernel, actualizando automáticamente los archivos `/etc/modules` y `/etc/modules.conf` (cambiando los archivos apropiados de `/etc/modutils`) para que los módulos configurados sean cargados automáticamente en el siguiente arranque. La información sobre los módulos disponibles la obtiene del directorio `/lib/modules`, los módulos cargados y sus parámetros los lee de `/etc/modutils` y `/etc/modules.conf` y la ayuda y la información interna de los archivos en `/usr/share/modules.conf`. `---modconf` es un script para el intérprete de comandos.
- ✓ update-modules: Actualiza el archivo `/etc/modules.conf` a partir de la información de los archivos del directorio `/etc/modutils`.

Descarga del Software utilizado:

Gcc	ftp://ftp.gnu.org/pub/gnu/gcc
Make	ftp://ftp.gnu.org/gnu/make
Binutils	ftp://ftp.vaLinux.com/pub/support/hjl/binutils/
libc6	ftp://sourceware.cygnus.com/pub/glibc/releases/
ncurses	ftp://ftp.gnu.org/gnu/ncurses
Cpp	ftp://sourceware.cygnus.com/pub/glibc/releases/
Fileutils	ftp://ftp.gnu.org/gnu/fileutils
shellutils	ftp://alpha.gnu.org/gnu/fetish/
Kernel	http://www.kernel.org

Bibliografía consultada:

➤ Sitios de InterNet:

- ✓ <http://www.kernel.org>
- ✓ <http://es.tldp.org/>
- ✓ <http://tldp.org/docs.html#howto>
- ✓ <http://www.tldp.org/HOWTO/mini/LILO.html>
- ✓ <http://www.gnu.org/manual/grub-0.92/grub.html>
- ✓ <http://www-lsi.ugr.es/~jagomez/disisop.html>
- ✓ <http://www.digitalhermit.com/~kwan/kernel.html>
- ✓ <http://perso.wanadoo.es/exter/aarg/arg-kernel.html>
- ✓ <http://www.insflug.org/COMOs/Kernel-Como/Kernel-Como-1.html>
- ✓ http://structio.sourceforge.net/guias/AA_Linux_colegio/AA_Linux_colegio.html
- ✓ <http://acm.asoc.fi.upm.es/documentacion/lpractico/node1.html>
- ✓ <http://tldp.org/LDP/lkmpg/>
- ✓ http://www.thc.org/papers/LKM_HACKING.html
- ✓ <http://es.wikipedia.org/>

➤ Recursos escritos:

- ✓ Manual de Administración de Linux – Steve Shah – Editorial McGraw-Hill
- ✓ Linux Máxima Seguridad –Editorial Prentice Hall

Cambios realizados

- Año 2005: Versión Inicial
- Año 2006: Se corrigió la redacción, se agregaron aclaraciones adicionales
- Año 2008: Se modificó el apunte en general con el fin de explicar el proceso de compilación de un Kernel 2.6
- Año 2009: Se corrigieron errores de redacción y omisiones sobre la versión anterior