UNIVERSIDAD NACIONAL DE LA PLATA

Sistemas Operativos

Práctica 2

Conceptos teóricos

El propósito de esta primera sección de la práctica es introducir los conceptos preliminares que necesitará el alumno, para desarrollar la actividad práctica del apartado B de la presente guía de estudio.

1. ¿Qué es el *kernel* de GNU/Linux? ¿Cuáles son sus funciones principales dentro del Sistema Operativo?
2. Indique una breve reseña histórica acerca de la evolución del *kernel* de GNU/Linux
3. Explique brevemente la arquitectura del *kernel* de GNU/Linux teniendo en cuenta: tipo de *kernel*, módulos, portabilidad, etc.
4. ¿Cuáles son los cambios que se introdujeron en el *kernel* a partir de la versión 3.0? ¿ Cuál fue la razón por la cual se cambió de la versión 2 a la 3? ¿Y la razón para el cambio de la versión 3 a la 4?
5. ¿Cómo se define el versionado de los *kernel*s de GNU/Linux?
6. ¿Cuáles son las razones por las cuáles los usuarios de GNU/Linux recompilan sus kernels?
7. ¿Cuáles son las distintas opciones para realizar la configuración de opciones de compilación de un *kernel*? Cite diferencias, necesidades (paquetes adicionales de software que se pueden requerir), pro y contras de cada una de ellas.
8. Nombre al menos 5 opciones de las más importantes que encontrará al momento de realizar la configuración de un *kernel* para su posterior compilación.
9. Indique que tarea realiza cada uno de los siguientes comandos durante la tarea de configuración/compilación del *kernel*:
   (a) `make menuconfig`
   (b) `make clean`
   (c) `make` (investigue la funcionalidad del parámetro `-j`)
   (d) `make modules` (utilizado en antiguos kernels, actualmente no es necesario)
   (e) `make modules_install`
   (f) `make install`
10. Una vez que el *kernel* fue compilado, ¿dónde queda ubicada su imagen? ¿dónde debería ser reubicada? ¿Existe algún comando que realice esta copia en forma automática?
11. ¿A qué hace referencia el archivo `initramfs`? ¿Cuál es su funcionalidad? ¿Bajo qué condiciones puede no ser necesario?

12. ¿Cuál es la razón por la que una vez compilado el nuevo *kernel*, es necesario reconfigurar el gestor de arranque que tengamos instalado?

13. ¿Qué es un módulo del *kernel*? ¿Cuáles son los comandos principales para el manejo de módulos del kernel?

14. ¿Qué es un parche del *kernel*? ¿Cuáles son las razones principales por las cuáles se deberían aplicar parches en el kernel? ¿A través de qué comando se realiza la aplicación de parches en el *kernel*?

Ejercicio taller: Compilación del *kernel* Linux

El propósito del siguiente ejercicio es el de guiar al alumno en el proceso de compilación del kernel de GNU/Linux. Si bien los siguientes ejercicios permiten agregar determinada funcionalidad al kernel, es aconsejable que los alumnos investiguen las distintas opciones con el fin de adquirir experiencia adicional. Para la realización de este taller se ha utilizado la versión 4.15 del kernel de GNU/Linux. Aquel alumno que decida utilizar otra versión deberá descargar el código fuente y los parches del kernel correspondientes de acuerdo a la versión utilizada.
A través de los siguientes pasos agregaremos nueva funcionalidad a nuestro *kernel* de GNU/Linux asumiendo que la misma no se encuentra soportada en nuestro *kernel* actual. Dentro de la funcionalidad a agregar se encuentran:

- El soporte para sistemas de archivos `minix` y `ext4`
- Soporte para la utilización de dispositivos de *loopback*
- Actualizaremos una versión de nuestro *kernel*, a través de la aplicación de parches

Aquel alumno que utilice la máquina virtual provista por la Cátedra, no tendrá que instalar el software necesario para realizar la compilación, ya que el mismo se encuentra incluido. **El nombre del usuario a utilizar es `so`, y su contraseña `sistemasoperativos`**. Este usuario tiene permitido utilizar `sudo`, por lo cual para obtener privilegios de `root` basta con ejecutar el comando `sudo su`.
En caso de no utilizar la máquina virtual provista, se deberá realizar la instalación del software requerido para la instalación (librerías, compiladores, etc.)
En los comandos de ejemplo de esta práctica se verá que algunos comandos empiezan con `$` y otros con `#`. Estos símbolos representan el *prompt* del usuario y no deben escribirse cuando se copie el comando. El símbolo `$` significa que el comando debe ejecutarse con un usuario normal. En el caso de la máquina virtual, es el usuario `so`. El símbolo `#` significa que el comando debe ejecutarse con privilegios de usuario `root`.

1. Descargue los archivos publicados en el sitio web de la cátedra, que se llaman *minixFS* y *ext4FS.gz* (en la máquina virtual provista por la cátedra se encuentran en el directorio `/home/so`).

   Estos archivos representan lo que se conoce como *loop device*. Básicamente son archivos regulares que pueden ser tratados como dispositivos de bloques, donde uno puede crear un sistema de archivos dentro de ellos y montarlos como si montáramos cualquier otro dispositivo de bloques como, por ejemplo, particiones de un disco rígido. La particularidad que tienen estos archivos, es que han sido formateados con distintos tipos de sistemas de archivos. El archivo denominado *minixFS* se formateó con un tipo de sistema de archivos denominado `minix`. Como ya sabrán, para poder acceder a la información de este pseudo dispositivo, tendremos que soportar el sistema de archivos `minix` en nuestro *kernel*. Por otro

lado, el archivo llamado *ext4FS.gz* está comprimido con la utilidad `gzip` (para descomprimirlo utilizaremos `gzip -d ext4FS.gz`). En este archivo comprimido, encontraremos otro dispositivo de bloques, formateado con el sistema de archivos *extended 4* (`ext4`).

2. La tarea que tendremos que realizar en este ejercicio consiste en intentar montar los archivos con formato `minix` y `ext4` que se encuentran publicado en el sitio de la cátedra y que ya hemos descargado, con el fin de poder visualizar la información que en él se encuentra. Para ello, seguiremos los siguientes pasos:
   (a) Descargaremos este archivo en algún directorio determinado de nuestro *File System*, como por ejemplo nuestro directorio personal, `$HOME`.
   (b) Verificaremos que dentro del directorio `/mnt` existan al menos dos directorios donde podamos montar nuestros pseudo dispositivos. Si no existen los directorios, los crearemos. Por ejemplo podemos utilizar el nombre `/mnt/minix/` y `/mnt/ext4/`.
   (c) A continuación montaremos nuestros dispositivos utilizando los siguientes comandos:

```
$ sudo su
# mount -t minix $HOME/minixFS /mnt/minix/ -o loop
# mount -t ext4 $HOME/ext4FS /mnt/ext4/ -o loop
```

   (d) ¿Puede explicar qué sucedió? **Nota**: *si estás usando la máquina virtual de la cátedra, el comportamiento esperado es que los comandos anteriores fallen.*

3. Descargue el código fuente del *kernel* a compilar, el parche correspondiente a la versión a la que se quiere llevar el código fuente, arme la estructura de directorios y aplique el parche (en la máquina virtual provista por la cátedra se encuentran en el directorio `/usr/src`).
   (a) Descargar el código fuente:

```
$ cd /usr/src/
$ sudo su
# wget https://kernel.org/pub/linux/kernel/v4.x/linux-4.15.tar.xz
```

   (b) Descargar el parche correspondiente a la versión a la que quiere llevar su código fuente. Si se está usando la máquina virtual provista por la cátedra, el parche también se encuentra en `/usr/src`:

```
# wget https://kernel.org/pub/linux/kernel/v4.x/patch-4.15.15.xz
```

   (c) Descomprimir el *kernel* utilizando usuario sin privilegios de `root` (en la virtual, el usuario `so`), en un nuevo directorio en el `$HOME` del usuario.

```
# exit
$ mkdir $HOME/kernel/
$ cd $HOME/kernel/
$ tar xvf /usr/src/linux-4.15.tar.xz
```

   (d) Aplique el parche descargado a nuestro código fuente. Esta tarea la haremos ubicados en el directorio de nuestro código fuente y a través de la herramienta `patch`, verificando que no arroje errores al finalizar.

```
$ cd $HOME/kernel/linux-4.15
$ xzcat /usr/src/patch-4.15.15.xz | patch -p1
```

4. Configurar el código fuente del *kernel* para su compilación. Esta tarea se realiza a través de los comandos vistos en el ejercicio 8 de la primera parte de esta práctica. En primer lugar, se deberá partir de la configuración del *kernel* que está corriendo actualmente. Por convención, esta configuración se encuentra ubicada en `/boot`. La copiaremos al directorio donde se encuentra el *kernel* parcheado.

   ```
   $ cp /boot/config-$(uname -r) $HOME/kernel/linux-4.15/.config
   ```

   En este caso utilizaremos la herramienta `menuconfig`. Para ello ejecutaremos:

   ```
   $ cd $HOME/kernel/linux-4.15
   $ make menuconfig
   ```

   Este comando pondrá en pantalla un menú de configuración del código fuente del *kernel* donde podremos seleccionar diversas opciones dependiendo de nuestro hardware y necesidades. Dentro de las opciones que debemos seleccionar con el fin de cumplir nuestro objetivo se encuentran:

   (a) Soporte para sistemas de archivos `minix`: En la opción *File Systems* → *Miscellaneous filesystems*, tendremos que seleccionar *Minix fs support*.

   (b) Soporte para sistemas de archivos `ext4`: En la opción *File Systems*, tendremos que seleccionar *The Extended 4 (ext4) filesystem*.

   (c) Soporte para dispositivos de *Loopback*: En la opción *Device Drivers* → *Block Devices*, tendremos que seleccionar *Loopback device support*.

   Adicionalmente puede ser necesario deshabilitar el firmado criptográfico de módulos, para ello:

   (a) Deshabilitar *Module signature verification* en la opción *Enable loadable module support*.

   (b) Deshabilitar *Provide system-wide ring of trusted keys* en la opción *Cryptographic API* → *Certificates for signature checking*.

   Estas últimas opciones deben deshabilitarse si al intentar compilar recibimos el error: `make[2]: *** No rule to make target 'debian/certs/benh@debian.org.cert.pem [...]'`.

   La forma de movernos a través de este menú es utilizando las flechas del cursor, la tecla *Enter* y la barra espaciadora. A través de esta última podremos decidir sobre determinada opción si la misma será incluida en nuestro *kernel*, si será soportada a través de módulos, o bien si no se dará soporte a la funcionalidad (`<*>`, `<M>`, `<>` respectivamente). Una vez seleccionadas las opciones necesarias, saldremos de este menú de configuración a través de la opción `Exit`, guardando los cambios.

5. Luego de configurar nuestro *kernel*, nos dispondremos a realizar la compilación del mismo y sus módulos.

   Para realizar la compilación deberemos ejecutar:

   ```
   $ make -jX
   ```

   X deberá reemplazarse por la cantidad de procesadores con los que cuente su máquina. En máquinas con más de un procesador o núcleo, la utilización de este parámetro puede acelerar

mucho el proceso de compilación, ya que ejecuta **X** *jobs* o procesos para la tarea de compilación en forma simultánea. Este comando debemos ejecutarlo ubicados dentro del directorio donde se encuentra el código fuente del *kernel* descargado (`$HOME/kernel/linux-4.15`). La ejecución de este último puede durar varios minutos, o incluso horas, dependiendo del tipo de hardware que tengamos en nuestra PC y las opciones que hayamos seleccionado al momento de la configuración. Una vez finalizado este proceso, debemos verificar que no haya arrojado errores. En caso de que esto ocurra debemos verificar de qué tipo de error se trata y volver a la configuración de nuestro *kernel* para corregir los problemas. Una vez corregidos tendremos que volver a compilar nuestro *kernel*. Previo a esta nueva compilación debemos correr el comando `make clean` para eliminar pasos inconclusos de compilaciones anteriores.

6. Finalizado este proceso, debemos reubicar las nuevas imágenes en los directorios correspondientes, instalar los módulos, crear una imagen `initramfs` y reconfigurar nuestro gestor de arranque.
   (a) En primer lugar, para realizar la instalación de los módulos ejecutaremos:
   ```
   $ sudo make modules_install
   ```
   Recordar que debemos estar ubicados en el directorio `$HOME/kernel/linux-4.15`. Este comando copiará los módulos compilados al directorio `/lib/modules/4.15.15`.
   (b) Para reubicar la imagen del *kernel* disponemos de al menos dos métodos.
      1. Utilizar el siguiente comando:
      ```
      $ sudo make install
      ```
      Este comando reubicará todos los archivos creados durante el proceso de compilación. Este es el método **recomendado** puesto que, entre otros detalles, no sobre-escribirá ningún archivo en caso de que el *kernel* que estemos instalando tenga la misma versión que el que está corriendo.
      2. Este método consiste en hacer lo mismo que el anterior, pero de forma manual. Se deben realizar la copia manual de cada uno de los archivos creados. Para ello debemos ejecutar los siguientes comandos:
      ```
      $ sudo su
      # cp -i $HOME/kernel/linux-4.15/arch/i386/boot/bzImage /boot/vmlinuz-4.15.15
      # cp -i $HOME/kernel/linux-4.15/System.map /boot/System.map-4.15.15
      # cp -i $HOME/kernel/linux-4.15/.config /boot/config-4.15.15
      ```
      El primer comando copiará la imagen del *kernel*, el segundo copiará el archivo de mapa de símbolos correspondiente a la nueva imagen del *kernel*. El último comando nos permitirá mantener una copia del archivo de configuración con el cual fue compilado nuestro *kernel*. Si bien no es necesario mantener este último archivo, es recomendable hacerlo, con el fin de mantener el orden y una documentación de las opciones de compilación utilizadas en la imagen compilada.
   (c) Si en el inciso anterior se utilizó la opción de `make install`, entonces podremos saltear este paso. De lo contrario, para la creación de la imagen `initramfs` utilizaremos el comando:
   ```
   # mkinitramfs -o /boot/initrd.img-4.15.15 4.15.15
   ```

(d) El último paso previo a realizar la prueba de nuestra imagen del *kernel*, es reconfigurar nuestro gestor arranque de manera tal que podamos *bootear* con la nueva imagen. Suponiendo que el gestor de arranque que tenemos instalado es GRUB en su versión 2, ejecutaremos el comando:

```
# update-grub2
```

Este comando se encargará de agregar las entradas correspondientes en el archivo de configuración `/boot/grub/grub.cfg`. Nuevamente, si utilizamos `make install` para instalar el *kernel*, entonces es posible saltear este paso.

7. Como último paso, a través del comando `reboot`, reiniciaremos nuestro equipo y probaremos el nuevo *kernel* recientemente compilado. Al momento de cargarse nuestro gestor de arranque, veremos una nueva entrada que hace referencia al nuevo *kernel*. Para *bootear*, seleccionamos esta entrada y verificamos que el sistema funcione correctamente. En caso de que el sistema no arranque con el nuevo *kernel*, tendremos la opción de volver a *bootear* con nuestro *kernel* anterior para corregir los errores y realizar una nueva compilación.

8. Repita el procedimiento del inciso dos. ¿Qué sucede ahora?

   Como nuestro *kernel* tiene soporte para este tipo de *File System*, entonces, ingresando a `/mnt/minix`, encontraremos la información contenida en este pseudo dispositivo. Nos encontraremos con una situación similar al querer montar el sistema de archivos `ext4`.