UNIVERSIDAD NACIONAL DE LA PLATA

Sistemas Operativos

Práctica 1

A - Introducción

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

1. ¿Qué es GCC?
2. ¿Qué es make y para que se usa?
3. La carpeta /home/so/practica1/ejemplos/01-make de la VM contiene ejemplos de uso de make. Analice los ejemplos, en cada caso ejecute `make` y luego `make run` (es opcional ejecutar el ejemplo 4, el mismo requiere otras dependencias que no vienen preinstaladas en la VM):
   a. Vuelva a ejecutar el comando `make`. ¿Se volvieron a compilar los programas? ¿Por qué?
   b. Cambie la fecha de modificación de un archivo con el comando `touch` o editando el archivo y ejecute `make`. ¿Se volvieron a compilar los programas? ¿Por qué?
   c. ¿Por qué “run” es un target “phony”?
   d. En el ejemplo 2 la regla para el target `dlinkedlist.o` no define cómo generar el target, sin embargo el programa se compila correctamente. ¿Por qué es esto?

   Nota, si no usás la VM podés descargar los ejemplos desde:
   https://gitlab.com/unlp-so/codigo-para-practicas
4. ¿Qué es el *kernel* de GNU/Linux? ¿Cuáles son sus funciones principales dentro del Sistema Operativo?
5. Explique brevemente la arquitectura del *kernel* Linux teniendo en cuenta: tipo de *kernel*, módulos, portabilidad, etc.
6. ¿Cómo se define el versionado de los *kernel*s Linux en la actualidad?
7. ¿Cuáles son los motivos por los que un usuario/a GNU/Linux puede querer re-compilar el kernel?

8. ¿Cuáles son las distintas opciones y menús 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.
9. Indique qué 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*?
15. Investigue la característica Energy-aware Scheduling incorporada en el kernel 5.0 y explique brevemente con sus palabras:
    a. ¿Qué característica principal tiene un procesador ARM big.LITTLE?
    b. En un procesador ARM big.LITTLE y con esta característica habilitada. Cuando se despierta un proceso ¿a qué procesador lo asigna el scheduler?
    c. ¿A qué tipo de dispositivos opinás que beneficia más esta característica?

    Ver https://docs.kernel.org/scheduler/sched-energy.html
16. Investigue la system call memfd_secret() incorporada en el kernel 5.14 y explique brevemente con sus palabras

a. ¿Cuál es su propósito?
b. ¿Para qué puede ser utilizada?
c. ¿El kernel puede acceder al contenido de regiones de memoria creadas con esta system call?

El siguiente artículo contiene bastante información al respecto: https://lwn.net/Articles/865256/

B - Ejercicio taller: Compilación del *kernel* Linux

El propósito de este ejercicio es que las y los estudiantes comprendan los pasos básicos del proceso de compilación del kernel de GNU/Linux.

Si bien esta práctica es guiada es aconsejable que las y los alumnas/os investiguen las distintas opciones y comandos utilizados.

Para la realización de este taller compilaremos la versión 6.13.7 del *kernel* Linux. Pero en lugar de descargar la versión deseada descargaremos la 6.7 y la actualizaremos a 6.8 mediante la aplicación un parche (patch) a modo de práctica.

Compilaremos un *kernel* Linux con las siguientes funcionalidades:

Soporte para sistemas de archivos `BTRFS`.

Soporte para la utilización de dispositivos de bloques *loopback*

Nota: Máquina virtual

La cátedra provee una máquina virtual que ya cuenta con el software requerido para la compilación así cómo el código fuente del kernel, patch y el archivo *btrfs.image.xz* necesario para esta práctica. Estos archivos se encuentran en el directorio `/home/so/kernel`.

Se sugiere utilizar la máquina virtual para esta práctica a fin de evitar problemas de booteo si se configura mal el bootloader.

En caso de no utilizar la máquina virtual provista, se deberá realizar la instalación del software requerido para la instalación (librerías, compiladores, etc.)

Credenciales de acceso de la máquina virtual:

	Usuario	Contraseña
Normal	so	so
Administrador	root	toor

El usuario `so` no cuenta con privilegios para ejecutar `sudo` por lo cuál para ejecutar comandos que requieran estos privilegios será necesario ejecutar `su -` o `su -c 'comando arg ...'`.

Nota: Ejemplos de la shell

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 sin privilegios. En el caso de la máquina virtual, es el usuario `so`. El símbolo `#` significa que el comando debe ejecutarse con privilegios usando el usuario `root`.

1. Descargue los siguientes archivos en un sistema GNU/Linux moderno, sugerimos descargarlo en el directorio `$HOME/kernel/` (donde $HOME es el directorio del usuario no privilegiado que uses):
   a. El archivo *btrfs.image.xz* publicado en la página web de la cátedra.
   b. El código fuente del kernel 6.13 (https://mirrors.edge.kernel.org/pub/linux/kernel/v6.x/linux-6.13.tar.xz).
   c. El parche para actualizar ese código fuente a la versión 6.13.7 (https://cdn.kernel.org/pub/linux/kernel/v6.x/patch-6.13.7.xz).
2. Preparación del código fuente:
   a. Posicionarse en el directorio donde está el código fuente y descomprimirlo:

```
$ cd $HOME/kernel/
$ tar xvf /usr/src/linux-6.13.tar.xz
```

   b. Emparchar el código para actualizarlo a la versión 6.8 usando la herramienta *patch*:

```
$ cd $HOME/kernel/linux-6.13
$ xzcat /usr/src/patch-6.13.7.xz | patch -p1
```

3. Pre-configuración del kernel:
   a. Usaremos como base la configuración del kernel actual, esta configuración por convención se encuentra en el directorio `/boot`. Copiaremos y

renombraremos la configuración actual al directorio del código fuente con el comando:

```
$ cp /boot/config-$(uname -r) $HOME/kernel/linux-6.13/.config
```

b. Generaremos una configuración adecuada para esta versión del kernel con olddefconfig. olddefconfig toma la configuración antigua que acabamos de copiar y la actualiza con valores por defecto para las opciones de configuración nuevas.

```
$ cd $HOME/kernel/linux-6.13
$ make olddefconfig
```

c. A fin de construir un kernel a medida para la máquina virtual usaremos a continuación localmodconfig que configura como módulos los módulos del kernel que se encuentran cargados en este momentos deshabilitando los módulos no utilizados. Es probable que make pregunte por determinadas opciones de configuración, si eso sucede presionaremos la tecla *Enter* en cada opción para que quede el valor por defecto hasta que make finalize.

```
$ make localmodconfig
```

4. Configuración personalizada del kernel. Utilizaremos la herramienta menuconfig para configurar otras opciones. Para ello ejecutaremos:

```
$ make menuconfig
```

a. Habilitar las siguientes opciones para poder acceder a *btrfs.tar.xz*:
   i. File Systems -> Btrfs filesystem support.
   ii. Device Drivers -> Block Devices -> Loopback device support.
b. Deshabilitar las siguientes opciones para reducir el tamaño del kernel y los recursos necesarios para compilarlo:
   i. General setup -> Configure standard kernel features (expert users).
   ii. Kernel hacking -> Kernel debugging.

Tip: Uso de menuconfig

La forma de movernos a través de este menú es utilizando las flechas del cursor, la tecla *Enter* y la barra espaciadora. La barra espaciadora permite decidir si la opción

seleccionada 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, realizaremos la compilación del mismo y sus módulos.
    a. Para realizar la compilación deberemos ejecutar:

```
$ make -jX
```

i.

Tip: Sobre la compilación

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.

El comando *lscpu* permite ver la cantidad de CPUs y/o cores disponibles. **Recordar que la cantidad de CPUs de las máquinas virtuales es configurable.**

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 cambiada la configuración tendremos que volver a compilar nuestro kernel. Previo a esta nueva compilación debemos correr el comando *make clean* para eliminar los archivos generados con la configuración vieja.

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. En general todo esto se puede hacer de forma automatizada con los siguientes comandos.

```
$ make modules_install
$ make install
```

Tip: Instalación en otras distribuciones

En algunas distribuciones GNU/Linux el comando *make install* sólo instala la imagen del kernel pero no genera la imagen *initramfs* ni configura el gestor de arranque. En esos casos será necesario hacer esas tareas de forma manual.

Los comandos a utilizar varían de acuerdo a la distribución usada.

Tip: Instalación manual en distribuciones basadas en Debian

En caso de querer entender mejor el proceso de instalación en lugar de ejecutar *make install* es posible instalar la imagen del kernel manualmente de la siguiente forma:

```
# cp  $HOME/kernel/linux-6.13/arch/x86_64/boot/bzImage\
    /boot/vmlinuz-6.13.7
# cp $HOME/kernel/linux-6.13/System.map /boot/System.map-6.13.7
# cp $HOME/kernel/linux-6.13/.config /boot/config-6.13.7
# mkinitramfs -o /boot/initrd.img-6.13.7 6.13.7
# update-grub2
```

7. Como último paso, a través del comando `reboot`, reiniciaremos nuestro equipo y probaremos el nuevo *kernel* recientemente compilado.
    a. En el 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.
    b. En caso de que el sistema no arranque con el nuevo *kernel*, podemos reiniciar el equipo y *bootear* con nuestro *kernel* anterior para corregir los errores y realizar una nueva compilación.
    c. Para verificar qué kernel se está ejecutando en este momento puede usar el comando:

```
$ uname -r
```

C - Poner a prueba el kernel compilado

btrfs.image.xz es un archivo de 110MiB formateado con el filesystem `BTRFS` y luego comprimido con la herramienta `xz`. Dentro contiene un script que deberás ejecutar en una máquina con acceso a Internet (puede ser la máquina virtual provista por la cátedra) para realizar la entrega obligatoria de esta práctica.

Para acceder al script deberás descomprimir este archivo y montarlo como si fuera un disco usando el driver “Loopback device” que habilitamos durante la compilación del kernel.

Usando el kernel 6.13.7 compilado en esta práctica:

1. Descomprimir el filesystem con:

```
$ unxz btrfs.image.xz
```

2. Verificaremos que dentro del directorio /mnt exista al menos un directorio donde podamos montar nuestro pseudo dispositivo. Si no existe el directorio, crearlo. Por ejemplo podemos crear el directorio */mnt/btrfs/*.
3. A continuación montaremos nuestro dispositivo utilizando los siguientes comandos:

```
$ su -
# mount -t btrfs -o loop $HOME/btrfs.image /mnt/btrfs/
```

4. Diríjase a */mnt/btrfs* y verifique el contenido del archivo README.md.