Compilación de un kernel custom con Buildroot

Alumnos

- Ulises Jeremias Cornejo Fandos - 13566/7
- Agustin Vanzato - 14499/8

La entrega consta del presente informe y los siguientes archivos:

- bzImage
- rootfs.cpio.xz
- rootfs.ext2

Convenciones

En el presente informe se muestran los pasos a seguir para compilar un kernel custom. Para esto se muestran comandos que el usuario deberá ejecutar en el cli de su computadora y comandos que deben ejecutarse en alguna VM detallada en cada paso.

Los comandos a ejecutar en el cli personal tendrán como prefijo el signo `$`, similar al estandar de la shell `sh`. Por otro lado, con animos de diferenciarlos de la computadora personal, los comandos a ejecutar en las VM planteadas se verán con el prefijo `%`, conocido del estandar de shell `csh`. Esto no nos indica que estemos utilizando *csh* para ejecutar los comandos sino que es solo una herramienta que utilizamos para diferenciar los contextos de ejecución de los mismos.

Adicionalmente, a lo largo de la guía se supone un usario `user` el cual tiene su *home* en el path `/home/user`.

Dependencias

En esta sección se detallan algunas de las dependencias que el sistema utilizado debería tener al momento de iniciar el proceso de compilación y ejecución de las imagenes generadas.

- cpio
- gcc
- libelf
- make
- qemu

Pasos

En esta sección se detallan los pasos a seguir para disponer de un kernel custom de Linux. Los mismos se probaron en una computadora con un linux basado en *Arch GNU/Linux* con kernel versión `x86_64 Linux 5.0.13-arch1-1-ARCH`.

Cabe destacar que en el proceso de compilación de buildroot se generan archivos de salida los cuales se deben modificar para su correcto funcionamiento en la plataforma. De igual forma los mismos se especifican en la presente guía por lo que no debería ser un gran problema.

Para un correcto entendimiento de cada uno de los pasos se aconseja haber leido el estandar definido en la sección de convenciones.

Ver sección de convenciones.

Clonar repositorio

Obtener buildroot clonando la rama `2018.08` de buildroot ejecutando el siguiente comando:

```
$ cd /home/user
$ git clone --depth=1 --branch=2018.08 git:git.busybox.net/buildroot
$ cd buildroot # Los pasos siguientes se ejecutarán en este directorio
```

Menu de configuración

Entrar al menu de configuración utilizando el siguiente comando

```
$ make menuconfig
```

Una vez dentro del menú seguir los siguientes pasos:

- En la opción *Linux Kernel*,
  - Seleccionar la versión más reciente del Kernel Linux,en este caso *4.17*, en el campo `Kernel Version`.
  - Seleccionar la opción *Use a custom config file* en el campo `Kernel Configuration`.
  - Ingresar el path `$(TOPDIR)/board/qemu/x86_64/linux.config` en el campo `Configuration File Path`.
- En la opción *Filesystem Images*,
  - Seleccionar la opción `cpio the root filesystem`. Dentro de la misma elegir el método de compresión `xz`.
  - Seleccionar la opción `ext2/3/4 root filesystem` con `ext4` como variante a utilizar.
- En la opción *Bootloaders* seleccionar la opción `syslinux -> install mbr`.
- En la opción *Target options* seleccionar `x86_64` en el campo `Target Architecture`.

Compilación

Utilizar el siguiente comando para comenzar el proceso de compilación

```
$ make
```

El mismo puede paralelizarse de las siguientes dos formas:

```
# make --jobs[=N]
$ make -j [N] # no recomendado en la documentación de buildroot
```

o mediante variables de configuración de buildroot `BR2_JLEVEL[=N]`, e.g.

```
$ echo "BR2_JLEVEL=2" >> .config
```

Notar que N indica la cantidad de Jobs a ejecutar.

> *"You should never use make -jN with Buildroot: top-level parallel make is currently not supported. Instead, use the BR2_JLEVEL option to tell Buildroot to run the compilation of each individual package with make -jN."*
>
> *Documentación de buildroot.*

Errores de compilación

En el proceso de compilación surgieron los siguientes errores,

Header file de sistema erroneo

El primer intento de compilación resulta en el siguiente error

```
/usr/bin/ld: main.o: in function `find_device_sysfs':
/home/user/buildroot/output/build/syslinux-6.03/extlinux/main.c:1131:
undefined reference to `minor'
/usr/bin/ld: /home/user/buildroot/output/build/syslinux-
6.03/extlinux/main.c:1131: undefined reference to `major'
/usr/bin/ld: main.o: in function `sysfs_get_offset':
/home/user/buildroot/output/build/syslinux-6.03/extlinux/main.c:133:
undefined reference to `minor'
/usr/bin/ld: /home/user/buildroot/output/build/syslinux-
6.03/extlinux/main.c:133: undefined reference to `major'
collect2: error: ld returned 1 exit status.
```

Este problema se debe a que se utilizan referencias las cuales no existen en ninguno de los header files de sistema requeridos. Una solución es agregar la siguiente linea de código al archivo `./output/build/syslinux-6.03/extlinux/main.c`

```
#include <sys/sysmacros.h>
```

Esto podría, o no, generar errores dado que redefine macros definidas en el header file `<sys/types.h>` sin correcta verificación. Sin embargo, alcanza con cambiar la linea `#include <sys/types.h>` por `#include <sys/sysmacros.h>` en el archivo `.c` anteriormente mencionado para solucionar el problema.

Este problema no hubiese tenido lugar si se utilizara una versión más actual de buildroot como puede ser la presente en la rama `master` o `2019.02.x` del repositorio github.com/buildroot/buildroot.

Dependencias no instaladas

Otro error que puede aparecer es el siguiente

```
Makefile:970: *** "Cannot generate ORC metadata for CONFIG_UNWINDER_ORC=y,
please install libelf-dev, libelf-devel or elfutils-libelf-devel". Stop.
make[1]: *** [/home/user/buildroot/output/build/linux-
4.17.19/.stamp_built] Error 2
make: *** [_all] Error 2
```

Como se observa en la captura del error, faltan dependencias. Una de ellas es la ya mencionada `libelf`.

Ver sección de dependencias.

El problema actual se soluciona instalando alguna variante de `libelf`. En nuestro caso, utilizando *arch linux* instalamos como sigue:

```
$ sudo pacman -S libelf
```

Test

Una vez finalizada la etapa de compilación, se cuenta con los archivos `rootfs.cpio` y `rootfs.ext4` en el directorio `./output/images/`.

Se puede probar el correcto funcionamiento de las imagenes generadas ejecutando los siguientes comandos:

```
$ cd output/images
$ kvm -m 512 -kernel bzImage -initrd rootfs.cpio rootfs.ext4
```

o en el caso de arch, utilizando `qemu` donde *kvm* no está definido,

```
$ cd output/images
$ qemu-system-x86_64 -m 512 -kernel bzImage -initrd rootfs.cpio
rootfs.ext4
```

Si la prueba se ejecuta en forma exitosa podemos determinar que `rootfs.cpio` sirve como *initrd* y, una vez iniciada la VM podría montar el filesystem `rootfs.ext4`, encontrado en `/dev/sda`, utilizando los siguientes comandos:

```
% mkdir /mnt/ext4FS
% mount /dev/sda /mnt/ext4FS
```

Generar imagen modificada

Generar una imagen modificada a partir de `rootfs.cpio` la cual monte `rootfs.ext4` como `/`.

Para esto crear un directorio y descomprimir `rootfs.cpio` como sigue:

```
$ mkdir cpioFS
$ cd cpioFS
$ cat ../rootfs.cpio | sudo cpio -iv
```

El objetivo ahora es modificar `sbin/init/`. Haciendo un *cat* del mismo se observa que es un binario el cual no resulta *human readable*. Sin embargo, `sbin/init` es un link simbólico al binario `bin/busybox`. Cambiamos el `sbin/init` como sigue:

```
$ sudo rm -f sbin/init
$ touch sbin/init
$ sudo chmod +x sbin/init
```

Luego, con un editor de preferencia, e.g. nano, vi, code, ..., modificar el contenido del script quedando de la siguiente forma:

```
#!/usr/bin/env sh

/bin/mount -t proc proc /proc
/bin/mount -o remount,rw /
/bin/mkdir -p /dev/pts /dev/shm
/bin/mount -a
/sbin/swapon -a
/bin/ln -sf /proc/self/fd /dev/fd
/bin/ln -sf /proc/self/fd/0 /dev/stdin
/bin/ln -sf /proc/self/fd/1 /dev/stdout
/bin/ln -sf /proc/self/fd/2 /dev/stderr/bin/hostname -F /etc/hostname
```

```
# Looking for values of diff occurrencies of arg "root"
root=$(cat /proc/cmdline | xargs | sed 's/ /\n/g' | grep root= | cut -d'='
-f2 | sed 's/ /\n/g')

# root=<path/to/default/mountin/point> if -append not defined
if [[ "$root" == "" ]]
then
    echo "@@ Not found root argument"
    echo "@@ Setting \"root\" as default mounting point: \"/dev/sda\""
    root=${root:-"/dev/sda"}
else
    for line in $root
    do
        if [ -e "$line" ]
        then
            root="$line"
            break
        fi
    done
fi

if  [[ -e "$root" ]]
then
    echo "@@ Mounting $root"
else
    # Kernel panic when -append is defined with no valid mounting points
at "root" argument
    echo "@@ Mounting point $root not found"
    exit 1
fi

mkdir /mnt/ext4FS
mount "$root" /mnt/ext4FS
echo "Mounted rootfs"
mount --move /dev /mnt/ext4FS/dev
cd /mnt/ext4FS
exec switch_root . "/sbin/init" "$@"
```

Luego, generamos un comprimido utilizable como *initramfs* como sigue:

```
$ sudo find . | sudo cpio -H newc -o > ../rootfs.cpio
```

o generar un comprimido de la imagen con fines prácticos,

```
$ sudo find . | sudo cpio -H newc -o | xz --check=crc32 -v9 >
../rootfs.cpio.xz
```

Probar imagen generada

Una vez finalizada la etapa anterior, iniciar la VM de la siguiente forma:

```
$ qemu-system-x86_64 -m 512 -kernel bzImage -initrd rootfs.cpio.xz -append
root="/dev/sda" rootfs.ext4
```

El comando anterior inicia la VM con `rootfs.cpio.xz` como *initramfs* y luego monta `rootfs.ext4` como raíz, `/`, gracias al script `sbin/init` definido anteriormente.

Modificar init

Si la ejecución de la VM es correcta, dirigirse al directorio `/etc/init.d` y crear un script de arranque. En caso de no saber como utilizar editores como `vi`, y no contar con `nano` se recomienda utilizar el comando `cat` redireccionando la salida estandar a un archivo. En cualquier caso se puede crear el script de arranque como sigue:

```
% cd /etc/init.d
% echo "#\!/bin/sh\necho \"Hola Ulises Jeremias Cornejo Fandos y Agustin
Vanzato\"" > S60hello
% chmod +x S60hello
```

Con la ejecución de los comandos anteriores se creará un ejecutable con el siguiente contenido

```
#!/bin/sh
echo "Hola Ulises Jeremias Cornejo Fandos y Agustin Vanzato"
```

El nombre del script generado es `S60hello`. El nombre del mismo comienza con `S` indicando que es un script de arranque y sigue con un número, `60`, indicando la prioridad. Cuanto menor sea la misma, antes se ejecutará.

Probar script de arranque

En caso de haber completado el paso anterior, debería ser capaz de probarlo. Para esto, reinicie la VM y cuando vuelva a iniciar verá el mensaje `Hola Ulises Jeremias Cornejo Fandos y Agustin Vanzato`.