

INSTITUTO POLITECNICO NACIONAL  
UNIDAD PROFESIONAL INTERDISCIPLINARIA EN  
INGENIERIA Y TECNOLOGIAS AVANZAS

Sistemas Operativos en Tiempo Real

Practica:  
Instalación de un Sistema Operativo XV6

Profr.: M. Lamberto Maza Casas

Alumno: Plata Colín César Iván

01-Diciembre-2019

## Introduccion

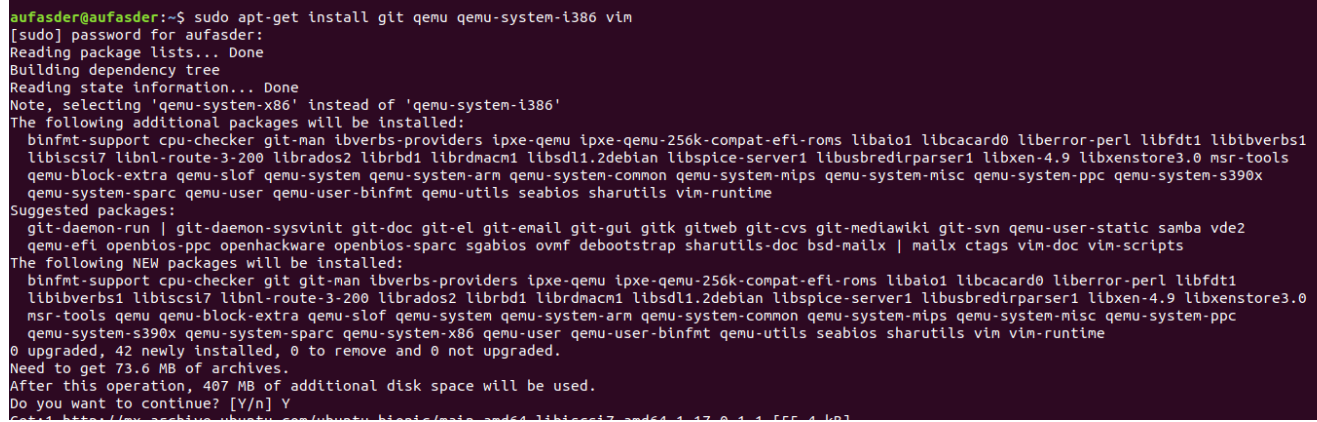
Como el nombre lo indica esta practica refiere a como obtener las fuentes de un sistema operativo compilar dichas fuentes y poder iniciar una computadora en el sistema operativo previamente compilado.

## Desarrollo

Como primer paso antes de descargar las fuentes del sistema operativo primero debemos instalar algunos programas que fungiran como herramientas para poder compilar y ejecutar el sistemaXV6.

En la linea de comandos (base Debian) escribir:

```
$ sudo apt-get install git make gcc qemu qemu-system-i386 vim
```



```
aufasder@aufasder:~$ sudo apt-get install git qemu qemu-system-i386 vim
[sudo] password for aufasder:
Reading package lists... Done
Building dependency tree
Reading state information... Done
Note, selecting 'qemu-system-x86' instead of 'qemu-system-i386'
The following additional packages will be installed:
  binfmt-support cpu-checker git-man ibverbs-providers ipxe-qemu ipxe-qemu-256k-compat-efi-roms libaio1 libcacard0 liberror-perl libfdt1 libibverbs1
  libiscsi7 libnl-route-3-200 librados2 librbdl1 librdmacm1 libsd1.2debian libspice-server1 libusbredirparser1 libxen-4.9 libxenstore3.0 msr-tools
  qemu-block-extra qemu-slof qemu-system qemu-system-arm qemu-system-common qemu-system-mips qemu-system-misc qemu-system-ppc qemu-system-s390x
  qemu-system-sparc qemu-user qemu-user-binfmt qemu-utils seabios sharutils vim-runtime
Suggested packages:
  git-daemon-run | git-daemon-sysvinit git-doc git-el git-email git-gui gitk gitweb git-cvs git-mediawiki git-svn qemu-user-static samba vde2
  qemu-efi openbios-ppc openhacking openbios-sparc sgabios ovmf debootstrap sharutils-doc bsd-mailx | mailx ctags vim-doc vim-scripts
The following NEW packages will be installed:
  binfmt-support cpu-checker git git-man ibverbs-providers ipxe-qemu ipxe-qemu-256k-compat-efi-roms libaio1 libcacard0 liberror-perl libfdt1
  libibverbs1 libiscsi7 libnl-route-3-200 librados2 librbdl1 librdmacm1 libsd1.2debian libspice-server1 libusbredirparser1 libxen-4.9 libxenstore3.0
  msr-tools qemu qemu-block-extra qemu-slof qemu-system qemu-system-arm qemu-system-common qemu-system-mips qemu-system-misc qemu-system-ppc
  qemu-system-s390x qemu-system-sparc qemu-system-x86 qemu-user qemu-user-binfmt qemu-utils seabios sharutils vim vim-runtime
0 upgraded, 42 newly installed, 0 to remove and 0 not upgraded.
Need to get 73.6 MB of archives.
After this operation, 407 MB of additional disk space will be used.
Do you want to continue? [Y/n] Y
Get:1 https://mirrors.ubuntu.com/mirrors/bionic/pool/main/a/and64/libiscsi7 and64-libiscsi7_1.17.0-1.1 [55.4 kB]
```

Despues nos indicara: "despues de la operacion #MB de espacio adicional sera usado ¿Desea continuar?" Daremos "Y" y presionamos enter y se descargarán los paquetes necesario para la instacion.

Despues de la instacion se recomienda reiniciar el sistema.

Una vez instalados lo requerimientos inciales debemos crear la carpeta my apps y ahi clonar el repositorio de xv6 para lo anterior usaremos las siguientes lineas:

```
$ mkdir myapps
$ cd myapps
$ git clone https://github.com/mit-pdos/xv6-public
```

Esta ultima operacion debio crear otra carpeta llamada xv6-public.

```
aufasder@aufasder:~$ mkdir myapps
aufasder@aufasder:~$ cd myapps
aufasder@aufasder:~/myapps$ git clone https://github.com/mit-pdos/xv6-public
Cloning into 'xv6-public'...
remote: Enumerating objects: 13983, done.
remote: Total 13983 (delta 0), reused 0 (delta 0), pack-reused 13983
Receiving objects: 100% (13983/13983), 17.16 MiB | 1.36 MiB/s, done.
Resolving deltas: 100% (9541/9541), done.
aufasder@aufasder:~/myapps$
```

Ahora debemos cambiar a la carpeta de xv6. Una vez dentro veremos si es posible compilar el sistema operativo y ejecutarlo a travez de qemu haciendo uso del Makefile que esta dentro de la carpeta para hacer todo lo anterior escribimos las siguientes lineas:

```
$ cd xv6-public
$ make
$ make qemu
```

```
aufasder@aufasder: ~/myapps/xv6-public
File Edit View Search Terminal Help
aufasder@aufasder:~/myapps$ cd xv6-public/
aufasder@aufasder:~/myapps/xv6-public$ make qemu
```

El programa comenzara a crear los archivos objeto y luego procedera a enlazarlos para despues generar el kernel del sistema operativo. Una vez terminada la anterior operacion nos mostrara una ventana con el sistemas operativo ejecutandose.

```
QEMU
SeaBIOS (version 1.10.2-1ubuntu1)

iPXE (http://ipxe.org) 00:03.0 C980 PCI2.10 PnP PMM+1FF8DDDD+1FECDDDD C980

Booting from Hard Disk...
cpu1: starting 1
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$ _
```

Ahora que ya podemos construir el sistema operativo es hora de implementar una funcion de usuario, proponemos como primer programa un hola mundo sencillo.

Como primer paso tomaremos la funcion de usuario echo.c la cual precedemos a copiar y renombrarla.

```
$ cp echo.c prueba.c
```

Ahora que ya tenemos un programa base lo modificaremos para hacer el hola mundo deseado.

```
$ vim prueba.c
```

```
#include "types.h"
#include "stat.h"
#include "user.h"

int
main(void)/*int argc, char *argv[]*/
{
    printf(1, "Hola mundo desde xv6\n");
    exit();
}
~
~
~
~
~
```

Antes de continuar debemos resaltar algunos puntos:

- No se necesitan variables de entrada debido a eso se comenta esta parte.
- El formato de imprimir es diferente al comunmente usado
- No se usa un return 0, mas bien la llamada al sistema exit();
- La libreria stdio.h debe estar incluida en alguna de las librerias de echo probablemente en user.h

ya con el programa realizado podemos incluirlo en el sistema operativo. Para ello debemos antes incluirlo en la seccion del Makefile correspondiente.

Por lo tanto modificaremos el make file con la linea

```
$ vim Makefile
```

Una vez dentro del Makefile nos dirigimos a las linea 168, esta linea comienza definiendo la variable "UPROGS" correspondiente a user programs este es el lugar donde debemos agregar el nombre de nuestro programa, en mi caso es prueba.c pero eliminaremos el .c cambiandolo por una diagonal invertida y le escribiremos antes un guion bajo.

```
UPROGS=\
    _cat\
    _echo\
    _forktest\
    _grep\
    _init\
    _kill\
    _ln\
    _ls\
    _mkdir\
    _rm\
    _sh\
    _stressfs\
    _usertests\
    _wc\
    _zombie\
    _prueba\
```

Una vez terminado guardamos y salimos (ESC, :wq ,enter). Ya estado fuera del editor VIM escribimos las siguientes lineas para volver a compilar y construir xv6

```
$ make clean
$ make
$ make qemu
```

Usando el comando ls dentro de xv6 podemos notar que el penultimo programa de la lista es nuestro programa de prueba.

```
init: starting sh
$ ls
.          1 1 512
..         1 1 512
README    2 2 2170
cat       2 3 13624
echo      2 4 12636
forktest  2 5 8064
grep      2 6 15500
init      2 7 13216
kill      2 8 12688
ln        2 9 12584
ls        2 10 14772
mkdir     2 11 12768
rm        2 12 12744
sh        2 13 23232
stressfs  2 14 13416
usertests 2 15 56348
wc        2 16 14164
zombie    2 17 12408
prueba    2 18 12400
console   3 19 0
$ _
```

Ahora podemos ejecutar el programa:

```
forktest 2 5 8064
grep      2 6 15500
init      2 7 13216
kill      2 8 12688
ln         2 9 12584
ls         2 10 14772
mkdir     2 11 12768
rm         2 12 12744
sh         2 13 23232
stressfs  2 14 13416
usertests 2 15 56348
wc         2 16 14164
zombie    2 17 12408
prueba    2 18 12400
console   3 19 0
$ prueba
Hola mundo desde xv6
$ _
```

y podemos notar que se ejecuta correctamente.

Como ultimo ejercicio crearemos una imagen del sistema XV6 que contenga el programa creado para dar la posibilidad de hacer una USB booteable y asi probar XV6 sobre hardware. Primero obtendremos el programa que nos genera la imagen del SO si le colocamos un kernel. Para esto escribimos la siguiente linea dentro de myapps

```
$ git clone https://github.com/GRUP03MV11SOTR-2020-1/3MV11SOTR.git
```

Despues regresamos a la carpeta de XV6 y ejecutamos los siguientes comandos

```
$ cd ../xv6-public
$ make clean
$ make
$ make kernelmemfs
```

Esta serie de comandos creara un archivo .elf (este es el kernel del sistema operativo) dentro de la carpeta xv6-public llamado kernelmemfs el cual deberemos copiar dentro de la

carpeta de:

3MV11SOTR/PRACTICA\_1\_Instalacion\_de\_un\_SOTR/MAKEFILE\_GRUB\_Legacy\_Kernel\_elf

Asi pues procedemos a copiar nuestro archivo "kernelmemfs" a la carpeta mencionada con el nombre de "kernel.elf" esto se logra con la siguiente linea (estando dentro de la carpeta de xv6):

```
$ cp kernelmemfs
../3MV11SOTR/PRACTICA_1_Instalacion_de_un_SOTR/MAKEFILE_GRUB_Legacy_Kernel_elf/kernel.elf
```

Ya con el kernel dentro de la carpeta de "MAKE . . .elf" nos cambiamos a dicha carpeta con

```
$ cd
../3MV11SOTR/PRACTICA_1_Instalacion_de_un_SOTR/MAKEFILE_GRUB_Legacy_Kernel_elf
```

Ahora solo queda usar el archivo Makefile para que nos genere el archivo "os.iso" siendo este el que usaremos con un programa para booteables de sistemas operativos.

```
$ make all
```

y asi nos genera el archivo solicitado.

```
aufasder@aufasder:~/myapps/3MV11SOTR/PRACTICA_1_Instalacion_de_un_SOTR/MAKEFILE_GRUB_Legacy_Kernel_elf$ make all
cp -v kernel.elf iso/boot/kernel.elf
'kernel.elf' -> 'iso/boot/kernel.elf'
genisoimage -R
               \
               -b boot/grub/stage2_eltorito\
               -no-emul-boot
               -boot-load-size 4
               -A os
               -input-charset utf8
               -quiet
               -boot-info-table
               -o os.iso
               \
               iso
aufasder@aufasder:~/myapps/3MV11SOTR/PRACTICA_1_Instalacion_de_un_SOTR/MAKEFILE_GRUB_Legacy_Kernel_elf$ ls
iso  kernel.elf  Makefile  os.iso
aufasder@aufasder:~/myapps/3MV11SOTR/PRACTICA_1_Instalacion_de_un_SOTR/MAKEFILE_GRUB_Legacy_Kernel_elf$
```

Con el archivo os.iso deberia ser posible usarlo con cualquier tipo de programa que haga booteables USB partiendo de archivos ".iso". Sin embargo nosotros usamos el programa LinuxLive USB y nos condujo a resultados satisfactorios.

## conclusiones

El ejercicio fue muy ilustrativo acerca de como usar la terminal de linux asi como sus comando modo de funcionamiento y demas particularidad de linux, pero mas aun la forma en que se estructura un archivo Makefile el cual, entre otras cosas, permite compilar y ejecutar ciertos programas con ciertos parametros de tal manera que ahorra tiempo de escritura y tiempo de ejecucion pues al momento de recompilar y reconstruir puede reconocer programas que no tuvieron cambios y permitirles permanecer asi. Respecto a xv6 se puede notar que no siempre se tienen las mismas librerias en todos los sistemas operativos, pese a que el lenguaje c tiene una alta difusion y una librería estandarizada.