



INSTITUTO POLITÉCNICO NACIONAL
UNIDAD PROFESIONAL INTERDISCIPLINARIA EN
INGENIERÍA Y TECNOLOGÍAS AVANZADAS



SISTEMAS OPERATIVOS EN TIEMPO REAL

CONSTRUCCIÓN Y EJECUCIÓN DE XV6

PROFESOR: MAZA CASAS LAMBERTO

GRUPO: 3MV11

ALUMNA:

RAMIREZ PERALTA LILLYAN YAZMIN

Pasos previos a la instalación de xv6

Antes de comenzar la instalación del sistema operativo xv6 es necesario tener un subsistema o distribución de Linux para poder trabajar. En este caso, se muestra la instalación del sistema operativo utilizando Debian. Si se cuenta con Windows, es necesario realizar los pasos que se muestran a continuación para instalar Debian.

1. Correr PowerShell como administrador y escribir lo siguiente:
Enable-WindowsOptionalFeature -Online -FeatureName Microsoft-Windows-Subsystem-Linux
2. El programa tardará unos segundos realizando varias acciones y luego te pedirá que reinicies el equipo.
3. Luego de reiniciar, ya es posible instalar Debian, el cual se puede obtener de la liga:
<https://www.microsoft.com/es-mx/p/debian/9msvkqc78pk6?rtc=1&activetab=pivot:overviewtab>
4. Después de descargar Debian es necesario seguir los pasos que indica el instalador.

Una vez que se ha instalado y abierto Debian, es necesario ahora instalar los paquetes *vim*, *qemu*, *git* y *make*, ya que serán necesarios en este procedimiento. Esto se hace escribiendo el siguiente comando en Debian:

```
sudo apt-get install vim
```

Al escribir *sudo*, Debian pedirá la contraseña del usuario para poder realizar la acción. Si no se escribe esto el comando arrojará un error ya que requiere de la confirmación a través de una contraseña para instalar los paquetes. Se utilizará el mismo comando, cambiando el nombre del paquete, para cada uno de estos:

```
sudo apt-get install git
```

```
sudo apt-get install make
```

```
sudo apt-get install qemu
```

Instalación del sistema operativo xv6

El primer paso es clonar el repositorio donde se encuentra la carpeta *xv6-public*, utilizando el siguiente comando:

```
git clone https://github.com/mit-pdos/xv6-public
```

Cuando aparezca la palabra *done*, como se muestra en la figura, el programa habrá terminado de clonar el repositorio.

```

usuario@LAPTOP-TLE1M1AI:~/tarea1$ 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 | 3.33 MiB/s, done.
Resolving deltas: 100% (9541/9541), done.

```

Figura 1. Clonar repositorio con la carpeta xv6-public.

Para comprobar que el repositorio se clonó correctamente, se escribirá el comando `ls`. Con ello se mostrarán los archivos en la ubicación actual, y deberá aparecer el nombre de la carpeta `xv6-public`. Ya que se comprobó que la carpeta existe, es necesario mover la ubicación actual al interior de `xv6-public`, utilizando el comando `cd xv6-public`.

```

usuario@LAPTOP-TLE1M1AI:~/tarea1$ ls
xv6-public
usuario@LAPTOP-TLE1M1AI:~/tarea1$ cd xv6-public/

```

Figura 2. Comprobar que el repositorio se clonó correctamente.

El siguiente paso es modificar las fuentes. Para ello, se deben editar 3 archivos, el primero, será el *Makefile* de `xv6`. Entonces, se abre este código utilizando el comando `vim Makefile`, como se muestra en la figura.

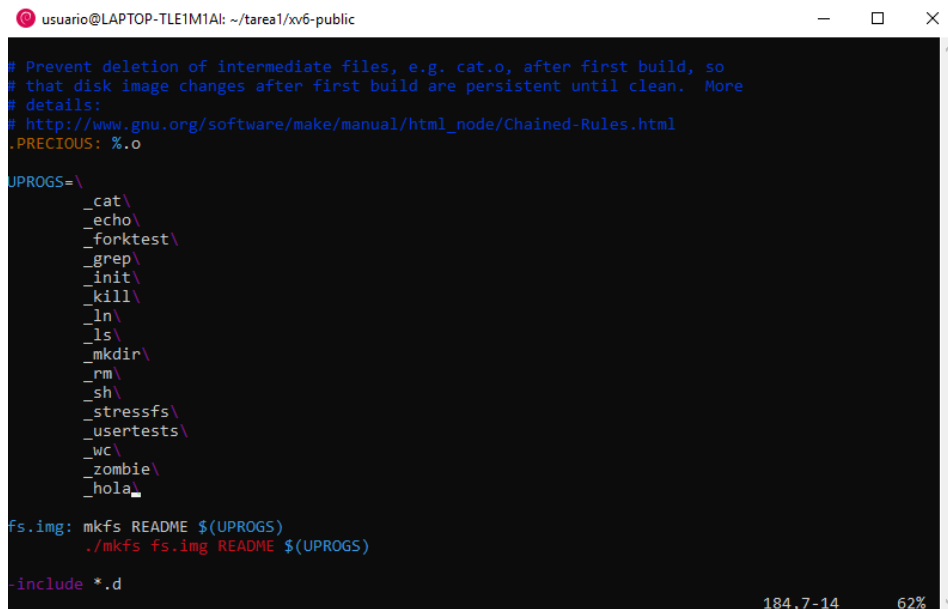
```

usuario@LAPTOP-TLE1M1AI:~/tarea1/xv6-public$ vim Makefile
usuario@LAPTOP-TLE1M1AI:~/tarea1/xv6-public$ vim init.c
usuario@LAPTOP-TLE1M1AI:~/tarea1/xv6-public$ vim hola.c
usuario@LAPTOP-TLE1M1AI:~/tarea1/xv6-public$ make clean
rm -f *.tex *.dvi *.idx *.aux *.log *.ind *.ilg \
*.o *.d *.asm *.sym vectors.S bootblock entryother \
initcode initcode.out kernel xv6.img fs.img kernelmemfs \
xv6memfs.img mkfs .gdbinit \
_cat _echo _forktest _grep _init _kill _ln _ls _mkdir _rm _sh _stressfs _usertests _wc _zombie _h
ola

```

Figura 3. Editar archivos utilizando el comando vim.

Una vez que se está dentro del editor `vim`, se debe buscar la parte del código donde está escrita la palabra en mayúsculas `UPROGS`. Para comenzar a editar se presiona la tecla `i`. Se hace un salto de línea en el último nombre en esa lista (`_zombie\`), y se agrega la palabra `_hola\`. Esto servirá para agregar al inicio del sistema operativo el programa *Hola Mundo* que se mostrará más adelante. Para guardar los cambios y salir, se presiona `Esc`, y posteriormente se escribe `:wq`.



```
usuario@LAPTOP-TLE1M1A1: ~/tarea1/xv6-public

# Prevent deletion of intermediate files, e.g. cat.o, after first build, so
# that disk image changes after first build are persistent until clean. More
# details:
# http://www.gnu.org/software/make/manual/html_node/Chained-Rules.html
.PRECIOUS: %.o

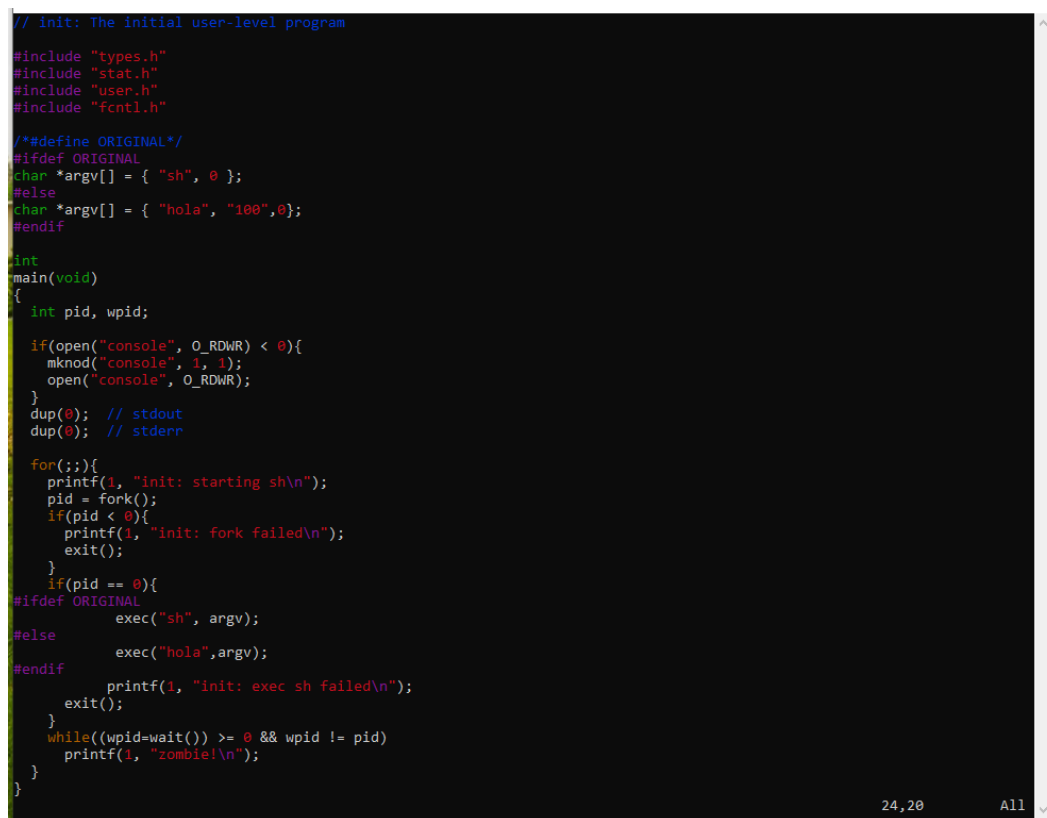
UPROGS=\
    _cat\
    _echo\
    _forktest\
    _grep\
    _init\
    _kill\
    _ln\
    _ls\
    _mkdir\
    _rm\
    _sh\
    _stressfs\
    _usertests\
    _wc\
    _zombie\
    _hola\

fs.img: mkfs README $(UPROGS)
    ./mkfs fs.img README $(UPROGS)

-include *.d
```

Figura 4. Modificación del código de Makefile.

Después, se escribe el comando `vim init.c` para editar este archivo, y se agregan las líneas que se muestran en la siguiente imagen.



```
// init: The initial user-level program

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

/*#define ORIGINAL*/
#ifdef ORIGINAL
char *argv[] = { "sh", 0 };
#else
char *argv[] = { "hola", "100", 0 };
#endif

int
main(void)
{
    int pid, wpid;

    if(open("console", O_RDWR) < 0){
        mknod("console", 1, 1);
        open("console", O_RDWR);
    }
    dup(0); // stdout
    dup(0); // stderr

    for(;;){
        printf(1, "init: starting sh\n");
        pid = fork();
        if(pid < 0){
            printf(1, "init: fork failed\n");
            exit();
        }
        if(pid == 0){
#ifdef ORIGINAL
            exec("sh", argv);
#else
            exec("hola", argv);
#endif
            printf(1, "init: exec sh failed\n");
            exit();
        }
        while((wpid=wait()) >= 0 && wpid != pid)
            printf(1, "zombie!\n");
    }
}
```

Figura 5. Modificación del código de init.c.

Para crear el programa *Hola Mundo*, se utiliza nuevamente el comando *vim*. Entonces, se escribe:

```
vim hola.c
```

Como el archivo no existe se abrirá uno nuevo, en el cual se escribirán las siguientes líneas de código:

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

int main(int argc, char *argv[])
{
    printf(1, "argc=%d\n", argc);
    printf(1, "%s:Hola Mundo xv6! %s\n", argv[0], argv[1]);
    while(!0);
    exit();
}/*end main*/
```

Figura 6. Edición del código del programa *hola.c*.

Se guarda el archivo y se sale del editor como se hizo anteriormente, con *Esc :wq*. Después, se compila utilizando el comando *make clean*, y posteriormente el comando *make*. Cuando el programa termine de compilar, se deberá ver la pantalla como se muestra en la siguiente figura.

```
usuario@LAPTOP-TLE1M1AI: ~/tarea1/xv6-public
gcc -m32 -gdwarf-2 -Wa,-divide -c -o entry.o entry.S
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-pointer -fno-
stack-protector -fno-pic -nostdinc -I. -c entryother.S
ld -m elf_i386 -N -e start -Ttext 0x7000 -o bootblockother.o entryother.o
objcopy -S -O binary -j .text bootblockother.o entryother.asm
objdump -S bootblockother.o > entryother.asm
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-pointer -fno-
stack-protector -nostdinc -I. -c initcode.S
ld -m elf_i386 -N -e start -Ttext 0 -o initcode.out initcode.o
objcopy -S -O binary initcode.out initcode
objdump -S initcode.o > initcode.asm
ld -m elf_i386 -T kernel.ld -o kernel entry.o bio.o console.o exec.o file.o fs.o ide.o ioapic.o kalloc.o kbd.o lap
ic.o log.o main.o mp.o picirq.o pipe.o proc.o sleeplock.o spinlock.o string.o swtch.o syscall.o sysfile.o sysproc.o t
rapasm.o trap.o uart.o vectors.o vm.o -b binary initcode entryother
objdump -S kernel > kernel.asm
objdump -t kernel | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' > kernel.sym
dd if=/dev/zero of=xv6.img count=10000
10000+0 records in
10000+0 records out
5120000 bytes (5.1 MB, 4.9 MiB) copied, 0.198291 s, 25.8 MB/s
dd if=bootblock of=xv6.img conv=notrunc
1+0 records in
1+0 records out
512 bytes copied, 0.0024602 s, 208 kB/s
dd if=kernel of=xv6.img seek=1 conv=notrunc
350+1 records in
350+1 records out
179204 bytes (179 kB, 175 KiB) copied, 0.0103381 s, 17.3 MB/s
usuario@LAPTOP-TLE1M1AI:~/tarea1/xv6-public$
```

Figura 7. Compilación de *xv6* con *make*.

Una vez que el programa termine la acción anterior, se escribe el comando *make qemu-nox*, y posteriormente *sudo make kernelmemfs*, ingresando la contraseña del usuario actual.

```
usuario@LAPTOP-TLE1M1AI: ~/tarea1/xv6-public
usuario@LAPTOP-TLE1M1AI:~/tarea1/xv6-public$ sudo make kernelmemfs
[sudo] password for usuario:
gcc -Werror -Wall -o mkfs mkfs.c
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-pointer -fno-stack-protector -fno-pie -no-pie -c -o ulib.o ulib.c
gcc -m32 -gdwarf-2 -Wa,-divide -c -o usys.o usys.S
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-pointer -fno-stack-protector -fno-pie -no-pie -c -o printf.o printf.c
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-pointer -fno-stack-protector -fno-pie -no-pie -c -o umalloc.o umalloc.c
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-pointer -fno-stack-protector -fno-pie -no-pie -c -o cat.o cat.c
ld -m elf_i386 -N -e main -Ttext 0 -o _cat cat.o ulib.o usys.o printf.o umalloc.o
objdump -S _cat > cat.asm
objdump -t _cat | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' > cat.sym
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-pointer -fno-stack-protector -fno-pie -no-pie -c -o echo.o echo.c
ld -m elf_i386 -N -e main -Ttext 0 -o _echo echo.o ulib.o usys.o printf.o umalloc.o
objdump -S _echo > echo.asm
objdump -t _echo | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' > echo.sym
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-pointer -fno-stack-protector -fno-pie -no-pie -c -o forktest.o forktest.c
# forktest has less library code linked in - needs to be small
# in order to be able to max out the proc table.
ld -m elf_i386 -N -e main -Ttext 0 -o _forktest forktest.o ulib.o usys.o
objdump -S _forktest > forktest.asm
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-pointer -fno-stack-protector -fno-pie -no-pie -c -o grep.o grep.c
ld -m elf_i386 -N -e main -Ttext 0 -o _grep grep.o ulib.o usys.o printf.o umalloc.o
objdump -S _grep > grep.asm
```

Figura 8. Comando make kernelmemfs.

Cuando se termine esta acción la pantalla deberá verse como se muestra en la siguiente figura.

```
usuario@LAPTOP-TLE1M1AI: ~/tarea1/xv6-public
objdump -t _zombie | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' > zombie.sym
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-pointer -fno-stack-protector -fno-pie -no-pie -c -o hola.o hola.c
ld -m elf_i386 -N -e main -Ttext 0 -o _hola hola.o ulib.o usys.o printf.o umalloc.o
objdump -S _hola > hola.asm
objdump -t _hola | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' > hola.sym
./mkfs fs.img README _cat _echo _forktest _grep _init _kill _ln _ls _mkdir _rm _sh _stressfs _use
rtests _wc _zombie _hola
nmeta 59 (boot, super, log blocks 30 inode blocks 26, bitmap blocks 1) blocks 941 total 1000
balloc: first 595 blocks have been allocated
balloc: write bitmap block at sector 58
ld -m elf_i386 -T kernel.ld -o kernelmemfs entry.o bio.o console.o exec.o file.o fs.o ioapic.o
kalloc.o kbd.o lapic.o log.o main.o mp.o picirq.o pipe.o proc.o sleeplock.o spinlock.o string.o
swtch.o syscall.o sysfile.o sysproc.o trapasm.o trap.o uart.o vectors.o vm.o memide.o -b binary
initcode entryother fs.img
objdump -S kernelmemfs > kernelmemfs.asm
objdump -t kernelmemfs | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' > kernelmemfs.sym
```

Figura 9. Fin de las acciones con el comando make kernelmemfs.

Después, para correr el sistema operativo sobre *qemu*, se escribe lo siguiente:

qemu-system-i386 -kernel kernelmemfs -nographic -m 512

Mientras el sistema operativo corre sobre *qemu*, la pantalla debe verse como se muestra a continuación.

```
usuario@LAPTOP-TLE1M1AI:~/tarea1/xv6-public$ qemu-system-i386 -kernel kernelmemfs -nographic -m 512
xv6...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
argc=1
sh:Hola Mundo xv6! (null)
```

Figura 10. Sistema operativo xv6 corriendo sobre qemu.

Para salir de la ejecución, se presiona *Ctrl+A* y posteriormente la tecla *X*, con lo cual aparecerá el mensaje de que la ejecución ha sido terminada, como se muestra en la figura siguiente.

```
usuario@LAPTOP-TLE1M1AI:~/tarea1/xv6-public$ qemu-system-i386 -kernel kernelmemfs -nographic -m 512
xv6...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
argc=1
sh:Hola Mundo xv6! (null)
QEMU: Terminated
usuario@LAPTOP-TLE1M1AI:~/tarea1/xv6-public$
```

Figura 11. Finalizar ejecución de xv6 sobre qemu.

El siguiente paso es descargar o clonar la carpeta *MAKEFILE_Grub_Legacy_Kernel* del siguiente repositorio:

https://github.com/sotrteacher/sotr_201808_201812/tree/master/PRACTICA_1_Instalacion_de_un_SOTR/MAKEFILE_GRUB_Legacy_Kernel_elf

Una vez hecho esto, mover la carpeta al escritorio u otro lugar donde se quiera conservar y revisar la ubicación exacta en el explorador de archivos. Copiar el archivo *kernelmemfs* a la carpeta de *MAKEFILE_Grub_Legacy_Kernel*, utilizando la ubicación de esta, como se muestra a continuación.

```
usuario@LAPTOP-TLE1M1AI:~/tarea1/xv6-public$ cp -v kernelmemfs /mnt/c/Users/lilly/OneDrive/Escritorio/MAKEFILE_GRUB_Legacy_Kernel_elf/
'kernelmemfs' -> '/mnt/c/Users/lilly/OneDrive/Escritorio/MAKEFILE_GRUB_Legacy_Kernel_elf/kernelmemfs'
usuario@LAPTOP-TLE1M1AI:~/tarea1/xv6-public$
```

Figura 12. Copiar kernelmemfs a carpeta MAKEFILE_Grub_Legacy_Kernel.

Después de realizar lo anterior, escribir el siguiente comando:

```
cp -vi kernelmemfs kernel.elf
```

```

usuario@LAPTOP-TLE1M1AI:/mnt/c/Users/lilly/OneDrive/Escritorio/MAKEFILE_GRUB_Legacy_Kernel_elf$ cp -vi kernelmemf
s kernel.elf
cp: overwrite 'kernel.elf'? y
'kernelmemfs' -> 'kernel.elf'

```

Figura 13. Sobrecribir kernel.elf.

Se vuelve a utilizar *make* para compilar, y posteriormente se comprueba que el archivo *os.iso* se encuentre en la carpeta de *MAKEFILE_Grub_Legacy*, con el comando *ls*.

```

usuario@LAPTOP-TLE1M1AI:/mnt/c/Users/lilly/OneDrive/Escritorio/MAKEFILE_GRUB_Legacy_Kernel_elf$ make
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
usuario@LAPTOP-TLE1M1AI:/mnt/c/Users/lilly/OneDrive/Escritorio/MAKEFILE_GRUB_Legacy_Kernel_elf$ ls
hello_world_c.exe  kernel.elf  kernelmemfs  leds_parallel_port  Makefile  os.iso

```

Figura 14. Comprobar que se encuentra el archivo os.iso.

Se comprueba que se pueda correr el sistema operativo sobre *qemu*, como se hizo anteriormente.

```

usuario@LAPTOP-TLE1M1AI:/mnt/c/Users/lilly/OneDrive/Escritorio/MAKEFILE_GRUB_Legacy_Kernel_elf$ qemu-system-i386
-kernel kernelmemfs -nographic -m 512
xv6...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
argc=1
sh:Hola Mundo xv6! (null)
QEMU: Terminated
usuario@LAPTOP-TLE1M1AI:/mnt/c/Users/lilly/OneDrive/Escritorio/MAKEFILE_GRUB_Legacy_Kernel_elf$

```

Figura 15. Comprobar nuevamente que xv6 corra sobre qemu.

Finalmente, se utiliza el software *Linux Live USB Creator* para crear una *liveUSB* con el sistema operativo *xv6*. Este software se puede descargar del siguiente enlace:

<https://www.linuxliveusb.com/en/download>

Para grabar el archivo *iso* en la USB, se elige la unidad como indica el software, y posteriormente, se elige la fuente, es decir, el archivo *iso* de la carpeta de *MAKEFILE_Grub_Legacy*. Seguir con los otros pasos que indica el software para crear la *liveUSB*.



Figura 16. Elegir unidad y fuente en Linux Live USB Creator. // Figura 17. Pasos 3 a 5 para crear la liveUSB en el software Linux Live USB Creator.

Luego de realizar todo lo anterior, se puede utilizar este sistema operativo insertando la USB en una computadora, previamente a encenderla. Después, presionar el botón de encendido y presionar F11 o F12 hasta que inicie la ejecución de xv6.