



Instituto Politécnico Nacional
Unidad Profesional Interdisciplinaria
en Ingenierías y Tecnologías
Avanzadas

INSTALACIÓN DE UN SOTR

“Instalación de Xv6 en la terminal Debian desde Windows 10”

BARRERA ANGELES DIEGO IVÁN

Sistemas Operativos en Tiempo Real

3MV11



Profesor:
Lamberto Maza Casas

Ciudad de México, México

Diciembre 2019

Introducción

xv6 es una reimplementación moderna de Sixth Edition Unix en ANSI C para sistemas multiprocesador x86 y RISC-V. Se utiliza con fines pedagógicos en el curso de Ingeniería de Sistemas Operativos del MIT, así como en el Curso de Diseño de Sistemas Operativos de Georgia Tech, así como en muchas otras instituciones.

Desarrollo

Primeramente debemos tener instalado la herramienta git para poder clonar el link donde se encuentran los archivos de xv6. Para corroborar esto escribimos *which git* y si nos aparece la siguiente pantalla significa que efectivamente está instalado.

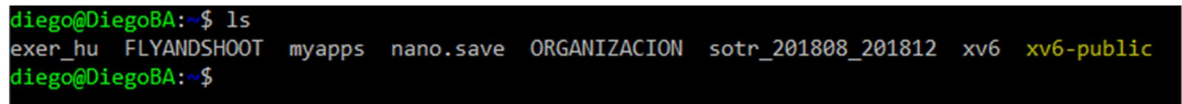


```
diego@DiegoBA: ~  
diego@DiegoBA:~$ which git  
/usr/bin/git  
diego@DiegoBA:~$
```

Figura 1. Git instalado.

En dado caso que no, escribimos en la terminal *sudo apt-get install git*. Nos pedirá la contraseña del usuario root, posterior a ello dejaremos instalar la herramienta.

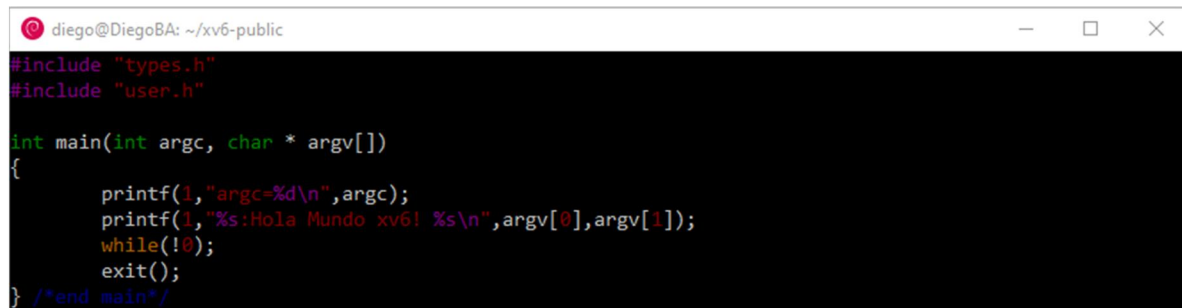
El siguiente paso es clonar el repositorio en donde se encuentran los archivos de xv6. Para ello ejecutaremos el comando *git clone https://github.com/mit-pdos/xv6-public*. Nos cercioramos de que la carpeta se clonó correctamente enlistando el directorio raíz y deberíamos encontrar la carpeta “xv6-public”.



```
diego@DiegoBA:~$ ls  
exer_hu  FLYANDSHOOT  myapps  nano.save  ORGANIZACION  sotr_201808_201812  xv6  xv6-public  
diego@DiegoBA:~$
```

Figura 2. Carpeta clonada.

Nos dirigimos a la carpeta tecleando *cd xv6-public/* y creamos un archivo en c que diga “Hola Mundo! Desde xv6”, para ello tecleamos *vim hola.c*, nos abrirá el editor vim y copiamos el siguiente código. Recordemos que para editar debemos teclear *i* y para guardar pulsamos la tecla *Esc* y *: + x* lo cual nos permite guardar cambios y cerrar.

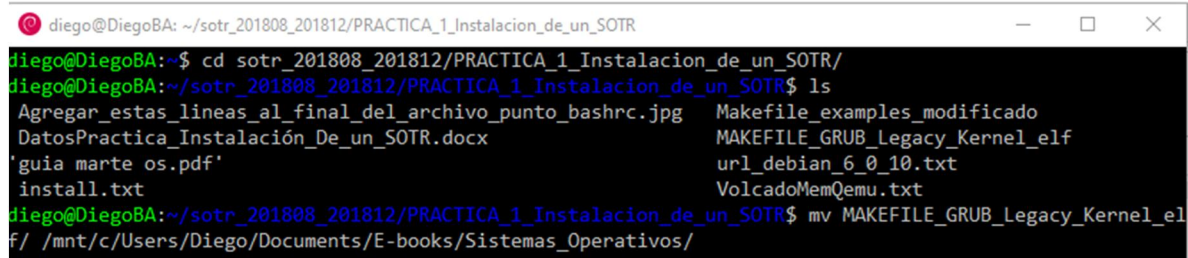


```
diego@DiegoBA: ~/xv6-public  
#include "types.h"  
#include "user.h"  
  
int main(int argc, char * argv[])  
{  
    printf(1, "argc=%d\n", argc);  
    printf(1, "%s:Hola Mundo xv6! %s\n", argv[0], argv[1]);  
    while(!0);  
    exit();  
} /*end main*/
```

Figura 3. Hola.c

Posteriormente clonamos el repositorio desde la siguiente liga con la misma instrucción del primer paso *git clone https://github.com/sotrteacher/sotr_201808_201812*. Nos movemos dentro de la misma a la siguiente dirección *cd sotr_201808_201812/PRACTICA_1_Instalacion_de_un_SOTR/* enlistamos los elementos y encontraremos una carpeta con el nombre *MAKEFILE_GRUB_Legacy_Kernel_elf*, la moveremos en un directorio que conozcamos, por ejemplo una carpeta dedicada a la materia. Un ejemplo de ello es el siguiente:

```
mv MAKEFILE_GRUB_Legacy_Kernel_elf/ /mnt/c/Users/Diego/Documents/E-books/Sistemas_Operativos/
```

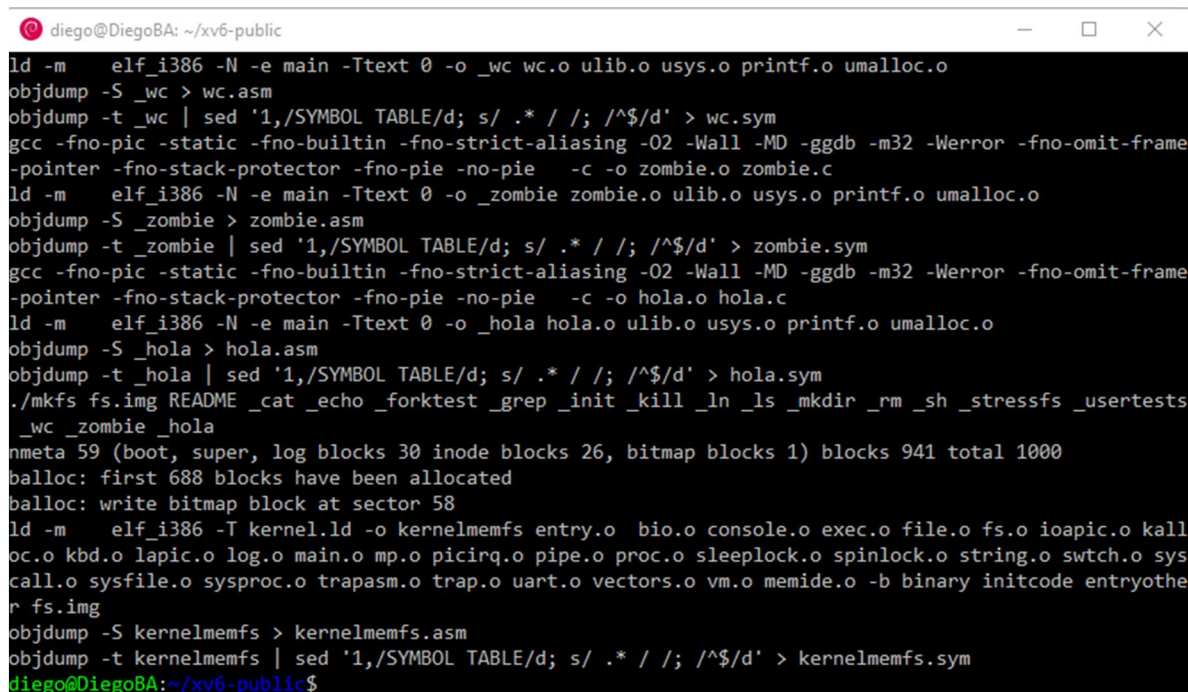


```
diego@DiegoBA: ~/sotr_201808_201812/PRACTICA_1_Instalacion_de_un_SOTR
diego@DiegoBA:~$ cd sotr_201808_201812/PRACTICA_1_Instalacion_de_un_SOTR/
diego@DiegoBA:~/sotr_201808_201812/PRACTICA_1_Instalacion_de_un_SOTR$ ls
Agregar_estas_lineas_al_final_del_archivo_punto_bashrc.jpg  Makefile_examples_modificado
DatosPractica_Instalación_De_un_SOTR.docx                 MAKEFILE_GRUB_Legacy_Kernel_elf
'guia_marte_os.pdf'                                       url_debian_6_0_10.txt
install.txt                                                VolcadoMemQemu.txt
diego@DiegoBA:~/sotr_201808_201812/PRACTICA_1_Instalacion_de_un_SOTR$ mv MAKEFILE_GRUB_Legacy_Kernel_elf/ /mnt/c/Users/Diego/Documents/E-books/Sistemas_Operativos/
```

Figura 4. MAKEFILE

Desde el ordenador nos dirigimos a la dirección especificada y corroboramos que se movió con éxito.

Nos dirigimos de nuevo al directorio base (*cd*) e instalaremos *qemu* con el siguiente comando *sudo apt-get install qemu*. Una vez realizado este paso nos movemos a la carpeta de *xv6-public* y tecleamos *make kernelmemfs* para compilar los archivos. Sabremos que el proceso terminó cuando nos aparezca la siguiente pantalla.



```
diego@DiegoBA: ~/xv6-public
ld -m elf_i386 -N -e main -Ttext 0 -o _wc wc.o ulib.o usys.o printf.o umalloc.o
objdump -S _wc > wc.asm
objdump -t _wc | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' > wc.sym
gcc -fno-pic -static -fno-builtin -fno-strict-aliasing -O2 -Wall -MD -ggdb -m32 -Werror -fno-omit-frame-pointer -fno-stack-protector -fno-pie -no-pie -c -o zombie.o zombie.c
ld -m elf_i386 -N -e main -Ttext 0 -o _zombie zombie.o ulib.o usys.o printf.o umalloc.o
objdump -S _zombie > zombie.asm
objdump -t _zombie | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/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/ .* / /; /^$/d' > hola.sym
./mkfs fs.img README_cat_echo_forktest_grep_init_kill_ln_ls_mkdir_rm_sh_stressfs_usertests _wc_zombie_hola
nmeta 59 (boot, super, log blocks 30 inode blocks 26, bitmap blocks 1) blocks 941 total 1000
ballocc: first 688 blocks have been allocated
ballocc: write bitmap block at sector 58
ld -m elf_i386 -T kernel.ld -o kernelmemfs entry.o bio.o console.o exec.o file.o fs.o ioapic.o kalloc.o kbd.o lapic.o log.o main.o mp.o picirq.o pipe.o proc.o sleeplock.o spinlock.o string.o swtch.o syscall.o sysfile.o sysproc.o trapasm.o trap.o uart.o vectors.o vm.o memide.o -b binary initcode entryother fs.img
objdump -S kernelmemfs > kernelmemfs.asm
objdump -t kernelmemfs | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' > kernelmemfs.sym
diego@DiegoBA:~/xv6-public$
```

Figura 5, make kernelmemfs

Posteriormente enlistamos los elementos y veremos un archivo con el nombre *kernelmemfs*.

```
cat.o      fs.d      kbd.d      memide.d   rm.d      syscall.c  usys.S
cat.sym    fs.h      kbd.h      memide.o   rm.o      syscall.d  vectors.o
console.c  fs.img    kbd.o      memlayout.h rm.sym    syscall.h  vectors.pl
console.d  fs.o      kernel.ld  _mkdir     runoff    syscall.o  vectors.S
console.o  gdbutil   kernelmemfs  mkdir.asm  runoff1   sysfile.c  vm.c
cuth       _grep     kernelmemfs.asm  mkdir.c    runoff.list sysfile.d  vm.d
date.h     grep.asm  kernelmemfs.sym  mkdir.d    runoff.spec sysfile.o  vm.o
defs.h     grep.c    _kill      mkdir.o     _sh       sysproc.c  _wc
dot-bochsrc grep.d    kill.asm   mkdir.sym   sh.asm    sysproc.d  wc.asm
_echo      grep.o    kill.c     mkfs        sh.c      sysproc.o  wc.c
echo.asm   grep.sym  kill.d     mkfs.c      sh.d      toc.ftr    wc.d
echo.c     _hola    kill.o     mmu.h       sh.o      toc.hdr    wc.o
echo.d     hola.asm  kill.sym   mp.c        show1     trapasm.o  wc.sym
echo.o     hola.c    lapic.c    mp.d        sh.sym    trapasm.S  x86.h
echo.sym   hola.d    lapic.d    mp.h        sign.pl   trap.c     _zombie
elf.h      hola.o    lapic.o    mp.o        sleep1.p  trap.d     zombie.asm
entry.o    hola.sym  LICENSE    Notes       sleeplock.c trap.o     zombie.c
entryother ide.c     _ln        param.h     sleeplock.d traps.h    zombie.d
entryother.asm _init    ln.asm     picirq.c    sleeplock.h TRICKS     zombie.o
entryother.d  init.asm ln.c       picirq.d    sleeplock.o types.h    zombie.sym
entryother.o  init.c   ln.d       picirq.o    spinlock.c uart.c
entryother.S  initcode ln.o       pipe.c      spinlock.d uart.d
entry.S       initcode.asm ln.sym     pipe.d      spinlock.h uart.o
exec.c        initcode.d log.c      pipe.o      spinlock.o ulib.c
diego@DiegoBA:~/xv6-public$
```

Figura 6. Kernelmemfs

Dicho archivo tendremos que copiarlo a la carpeta *MAKEFILE_GRUB_Legacy_Kernel_elf* ejecutando el siguiente comando. Hay que tomar en cuenta la dirección en la que está la carpeta.

```
cp -v kernelmemfs /mnt/c/Users/Diego/Documents/E-
books/Sistemas_Operativos/MAKEFILE_GRUB_Legacy_Kernel_elf/
```

```
diego@DiegoBA:~/xv6-public$ cp -v kernelmemfs /mnt/c/Users/Diego/Documents/E-books/Sistemas_Operativos/
MAKEFILE_GRUB_Legacy_Kernel_elf/
'kernelmemfs' -> '/mnt/c/Users/Diego/Documents/E-books/Sistemas_Operativos/MAKEFILE_GRUB_Legacy_Kernel_
elf/kernelmemfs'
diego@DiegoBA:~/xv6-public$
```

A continuación nos movemos al directorio donde copiamos tecleando *cd* y después *cd /mnt/c/Users/Diego/Documents/E-books/Sistemas_Operativos/MAKEFILE_GRUB_Legacy_Kernel_elf/*. Posteriormente reemplazamos un archivo dentro de esa carpeta llamado *kernel.elf* con el que copiamos anteriormente. Para ello escribimos *cp -vi kernelmemfs kernel.elf*

```
diego@DiegoBA:/mnt/c/Users/Diego/Documents/E-books/Sistemas_Operativos/MAKEFILE_GRUB_Legacy_Kernel_elf$
cp -vi kernelmemfs kernel.elf
cp: overwrite 'kernel.elf'? Y
'kernelmemfs' -> 'kernel.elf'
diego@DiegoBA:/mnt/c/Users/Diego/Documents/E-books/Sistemas_Operativos/MAKEFILE_GRUB_Legacy_Kernel_elf$
```

Figura 7. Kernelmemfs

Ejecutamos *make* y encontraremos en la carpeta un archivo *os.iso*.

```

diego@DiegoBA: /mnt/c/Users/Diego/Documents/E-books/Sistemas_Operativos/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
diego@DiegoBA: /mnt/c/Users/Diego/Documents/E-books/Sistemas_Operativos/MAKEFILE_GRUB_Legacy_Kernel_elf$

```

Figura 8. Creación de os.iso

iso	03/09/2019 03:08 ...	Carpeta de archivos	
alarma_tarea	03/12/2019 01:04 ...	C Source File	1 KB
alarma_tarea	03/12/2019 01:02 ...	Aplicación	922 KB
hello_world_c	20/09/2019 03:44 ...	Aplicación	905 KB
kernel.elf	03/12/2019 03:59 ...	Archivo ELF	704 KB
kernelmemfs	03/12/2019 03:54 ...	Archivo	704 KB
leds_parallel_port	30/08/2019 08:12 ...	Archivo	1,015 KB
Makefile	30/08/2019 08:12 ...	Archivo	1 KB
os	03/12/2019 04:00 ...	Archivo ISO	1,168 KB

Figura 9. Archivo os.iso

Posteriormente corremos sobre qemu tecleando `qemu-system-i386 -nographic -cdrom os.iso -m 512`

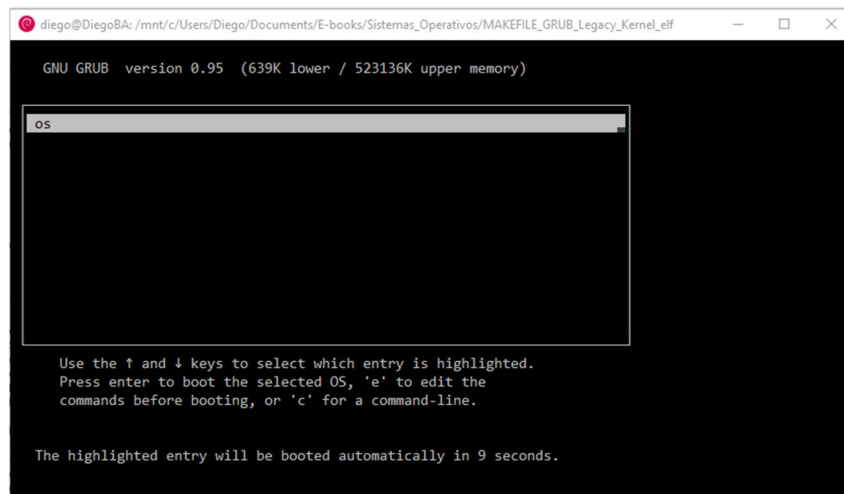


Figura 10. QEMU

```
diego@DiegoBA: /mnt/c/Users/Diego/Documents/E-books/Sistemas_Operativos/MAKEFILE_GRUB_Legacy_Kernel_elf
Booting 'os'

kernel /boot/kernel.elf
[Multiboot-elf, <0x100000:0x778e:0x0>, <0x108000:0x7f516:0xaf52>, shtab=0x19
32f8, entry=0x10000cxv6...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
argc=2
hola:Hola Mundo xv6! 100
```

Figura 11. Programa corriendo sobre QEMU.

Posteriormente con el software LiLi quemamos el os.iso para poder correrlo sobre hardware.



Recordemos que la memoria hay que formatearla así que es importante no tener archivos importantes ahí. Por último podremos correr sobre hardware.