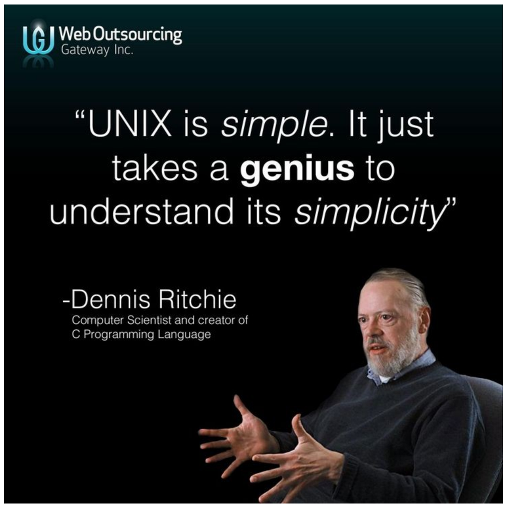
CS 370 – Project #1, Part A

Purpose: Become familiar with the **xv6** Teaching Operating System, shell organization, and system calls
Points: 50

Introduction:
The **xv6** OS is an educational OS designed by MIT for use as a teaching tool to give students hands-on experience with topics such as shell organization, virtual memory, CPU scheduling, and file systems. Although **xv6** lacks the full functionality of a modern OS, it is based on Sixth Edition Unix (also called V6) and is very similar in design and structure.


Web Outsourcing Gateway Inc.
"UNIX is simple. It just takes a genius to understand its simplicity"
-Dennis Ritchie
Computer Scientist and creator of C Programming Language

Resources:
The following resources provide more in depth information regarding **xv6**. They include the **xv6** reference book, tools for installing, and guidance to better understand **xv6**.

1. **xv6** Reference Book:
   https://pdos.csail.mit.edu/6.828/2020/xv6/book-riscv-rev1.pdf
2. Lab Tools Guide: https://pdos.csail.mit.edu/6.828/2020/tools.html
3. Lab Guidance: https://pdos.csail.mit.edu/6.828/2020/labs/guidance.html

Project:
Complete the following steps.

- Install an Ubuntu development environment
  - You may use a virtual machine (VM) from a previous class, install a new VM from scratch, use the UNLV Ubuntu VM image, or install a dual-boot.
- Install and become familiar with **xv6** Teaching Operating System
  - See installation instructions
  - Focus on shell organization

Submission
When complete, submit:

- A screen shot of the hello world program (PNG file) via the class web page (assignment submission link) by class time.
  - *Your hello program output must include your name.* Refer to the example submission for an example of what should be submitted.
  - *Note*, In Ubuntu there is a pre-installed program named *screenshot* that will be useful to capture the terminal window.
  - The full source code will be submitted in a subsequent Part.

Submissions received after the due date/time will not be accepted.

<u>Installing xv6</u>
The following instructions walk you through the process of setting up **xv6** within Ubuntu.

As previously noted, you may choose to use the UNLV Ubuntu image or install a dual-boot environment. If desired, you may also use VMware or Parallels virtualization software (which may cost money) instead of VirtuaBox. You may also attempt to use Windows Subsystem for Linux (WSL), however this has not been fully tested.

1. Download and Install VirtualBox

   a. Use default/recommended RAM during installation.
   b. IMPORTANT: Allocate at least 40GB of disk space to the machine during set-up. Set up Ubuntu in VirtualBox. If you use the default 8 GB setting, you are may run out of space at some point.
   c. IMPORTANT: Allocate as many CPUs as possible, before installing Ubuntu in VirtualBox the amount of CPUs can be changed.
   d. Under Display settings, increase display memory to 128 MB.
   e. Use all other recommendations during installation.

2. Download Ubuntu 22.04.1 LTS 64-bit

   a. Do ***<u>not</u>*** install higher or lower version of Ubuntu
   b. *Note*, to simplify the VM install, you may use the UNLV Ubuntu 22 Image for Virtual Box.

3. Open a terminal and execute the following commands:

   a. **`sudo apt-get update`**
   b. **`sudo apt-get upgrade`**
   c. **`sudo apt-get install git zip build-essential`**
   d. **`sudo apt-get install gdb-multiarch qemu-system-misc`**
   e. **`sudo apt-get install gcc-riscv64-linux-gnu`**
   f. **`sudo apt-get install binutils-riscv64-linux-gnu`**
   g. **`git clone https://github.com/mit-pdos/xv6-riscv.git xv6`**
   h. **`chmod 700 -R xv6`**

You are now ready to run QEMU and xv6. To run **xv6** on QEMU execute the following commands in the terminal:

1. **cd xv6** (if not already there)
2. **make qemu** (this is how you start **xv6**)

You should get a **$** prompt which is the **xv6** prompt.

For example, in xv6, the **ls** command should show:

```
$ ls
.              1 1 512
..             1 1 512
README         2 2 2170
cat            2 3 13604
echo           2 4 12616
forktest       2 5 8044
grep           2 6 15480
init           2 7 13196
kill           2 8 12668
ln             2 9 12564
ls             2 10 14752
mkdir          2 11 12748
rm             2 12 12724
sh             2 13 23212
stressfs       2 14 13396
usertests      2 15 56328
wc             2 16 14144
zombie         2 17 12388
console        3 18 0
$
```

Note: To close xv6 use **ctrl-a x**

Using GDB:

At some point, you may need to run xv6 on QEMU with GDB. Open two terminals as follows.

In terminal 1, execute the following commands:

1. **`cd xv6`** *(if not already in the xv6 directory)*
2. **`make qemu-gdb`**

In terminal 2, execute the following commands:

1. **`cd xv6`** *(if not already in the xv6 directory)*
2. **`gdb-multiarch -iex "set auto-load safe-path /" kernel/kernel`**

Helpful Information:

When attempting to exit **xv6**, use **`ctrl-a x`**
To quit GDB: **`q`**
To terminate an action being executed in GDB: **`ctrl-c`**
If your mouse and/or keyboard becomes trapped in QEMU, typing **ctrl-alt** will exit the mouse grab.

If you are debugging and you make a change to the code, make sure to close both QEMU and GDB. Otherwise they will get out of sync and cause problems.

To find the exact path required for the **`gdb`** command you can run **`gdb`** kernel in the second terminal. The path will be listed in the error provided.

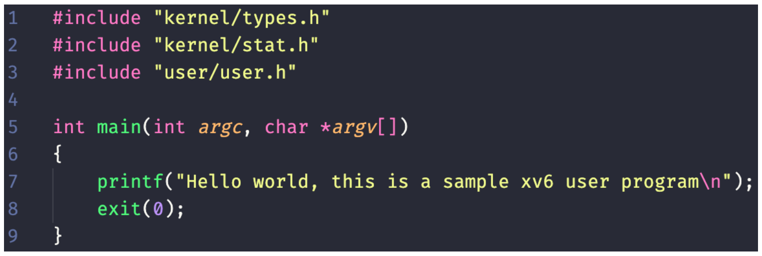
Inserting a User Program Within xv6

Overview

One of the first steps in navigating **xv6** is learning to insert a user level C program into the system. This will eventually allow you to test and implement more advanced features. These features include additional system calls such as ***time***. The following tutorial demonstrates how to add a basic user program and display it at the shell prompt.

Tutorial

1. Create a new C program (**`xv6/user/hello.c`**) in your chosen text editor. The contents of the program can be as simple as that shown in Figure 1.



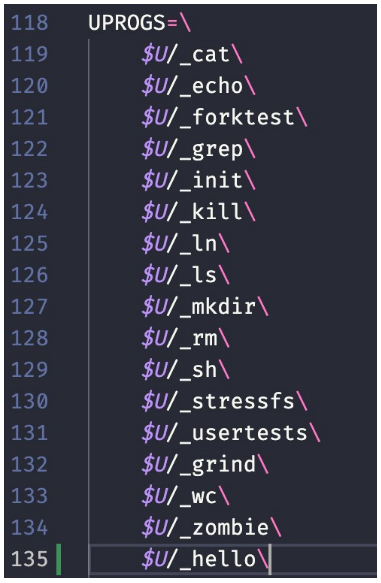
```c
#include "kernel/types.h"
#include "kernel/stat.h"
#include "user/user.h"

int main(int argc, char *argv[])
{
    printf("Hello world, this is a sample xv6 user program\n");
    exit(0);
}
```

Figure 1: Sample **xv6** User Program (**hello.c**)

Note, you should add your ***own name*** to the output string (as shown in the example below).

2. Save the program within the **xv6/user** folder.

3. Open the **xv6** Makefile from **xv6** directory (**`xv6/Makefile`**). The **Makefile** should be edited to ensure that **`hello.c`** is properly added to the existing set of user programs (under UPROGS) to be compiled. Note: Makefiles are sensitive to different types of whitespace, use a TAB at the start of the line when adding your hello program. This is done as shown in Figure 2.



```
118  UPROGS=\
119  	$U/_cat\
120  	$U/_echo\
121  	$U/_forktest\
122  	$U/_grep\
123  	$U/_init\
124  	$U/_kill\
125  	$U/_ln\
126  	$U/_ls\
127  	$U/_mkdir\
128  	$U/_rm\
129  	$U/_sh\
130  	$U/_stressfs\
131  	$U/_usertests\
132  	$U/_grind\
133  	$U/_wc\
134  	$U/_zombie\
135  	$U/_hello\
```

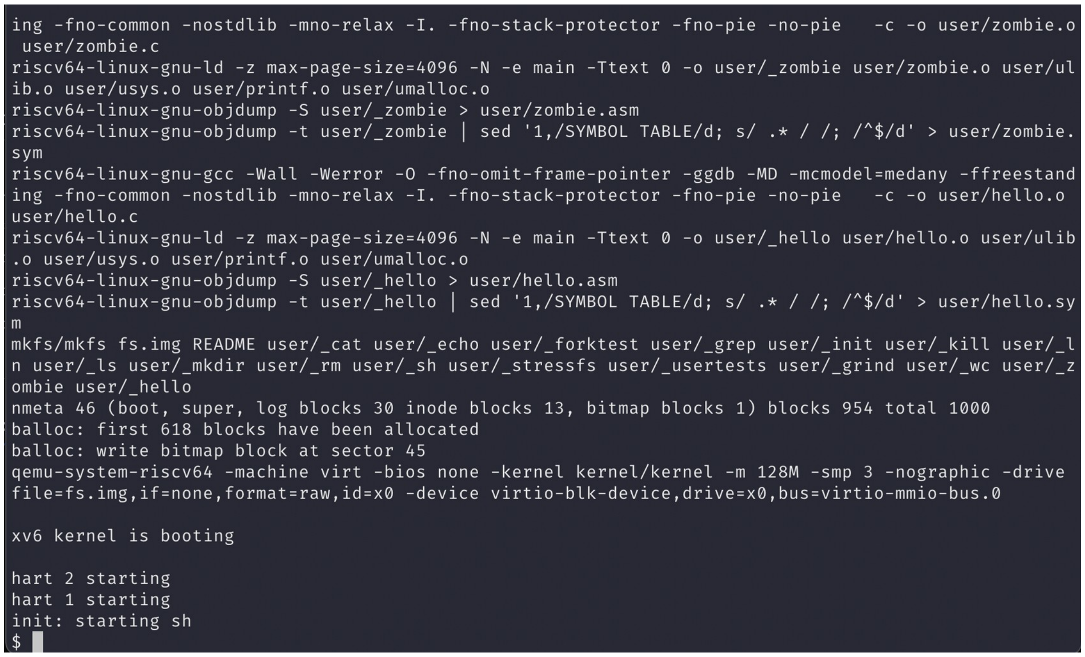
Figure 2: Add **`hello`** to the List of User Programs

4. Open the terminal to compile the system and start up **xv6** on QEMU. The commands should be the following:

```
cd xv6
make clean
make qemu
```

*(note, your **xv6** directory might be different)*

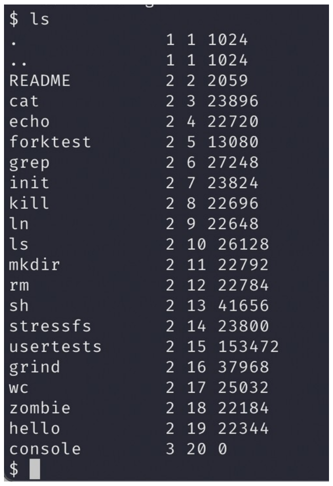
If there were no compilation errors **xv6** should have been properly launched something like as shown in Figure 4.



```
ing -fno-common -nostdlib -mno-relax -I. -fno-stack-protector -fno-pie -no-pie   -c -o user/zombie.o
 user/zombie.c
riscv64-linux-gnu-ld -z max-page-size=4096 -N -e main -Ttext 0 -o user/_zombie user/zombie.o user/ul
ib.o user/usys.o user/printf.o user/umalloc.o
riscv64-linux-gnu-objdump -S user/_zombie > user/zombie.asm
riscv64-linux-gnu-objdump -t user/_zombie | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' > user/zombie.
sym
riscv64-linux-gnu-gcc -Wall -Werror -O -fno-omit-frame-pointer -ggdb -MD -mcmodel=medany -ffreestand
ing -fno-common -nostdlib -mno-relax -I. -fno-stack-protector -fno-pie -no-pie   -c -o user/hello.o
user/hello.c
riscv64-linux-gnu-ld -z max-page-size=4096 -N -e main -Ttext 0 -o user/_hello user/hello.o user/ulib
.o user/usys.o user/printf.o user/umalloc.o
riscv64-linux-gnu-objdump -S user/_hello > user/hello.asm
riscv64-linux-gnu-objdump -t user/_hello | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$/d' > user/hello.sy
m
mkfs/mkfs fs.img README user/_cat user/_echo user/_forktest user/_grep user/_init user/_kill user/_l
n user/_ls user/_mkdir user/_rm user/_sh user/_stressfs user/_usertests user/_grind user/_wc user/_z
ombie user/_hello
nmeta 46 (boot, super, log blocks 30 inode blocks 13, bitmap blocks 1) blocks 954 total 1000
balloc: first 618 blocks have been allocated
balloc: write bitmap block at sector 45
qemu-system-riscv64 -machine virt -bios none -kernel kernel/kernel -m 128M -smp 3 -nographic -drive
file=fs.img,if=none,format=raw,id=x0 -device virtio-blk-device,drive=x0,bus=virtio-mmio-bus.0

xv6 kernel is booting

hart 2 starting
hart 1 starting
init: starting sh
$ 
```

Figure 4: Compilation and Start-Up of **xv6**

5. Use the ls command to verify that `hello.c` was added to the existing list of available user programs as shown in Figure 5. You can then type **hello** to view the output of the newly created test program.



```
$ ls
.              1 1 1024
..             1 1 1024
README         2 2 2059
cat            2 3 23896
echo           2 4 22720
forktest       2 5 13080
grep           2 6 27248
init           2 7 23824
kill           2 8 22696
ln             2 9 22648
ls             2 10 26128
mkdir          2 11 22792
rm             2 12 22784
sh             2 13 41656
stressfs       2 14 23800
usertests      2 15 153472
grind          2 16 37968
wc             2 17 25032
zombie         2 18 22184
hello          2 19 22344
console        3 20 0
$ 
```

Figure 5: Successful Sample Program Output

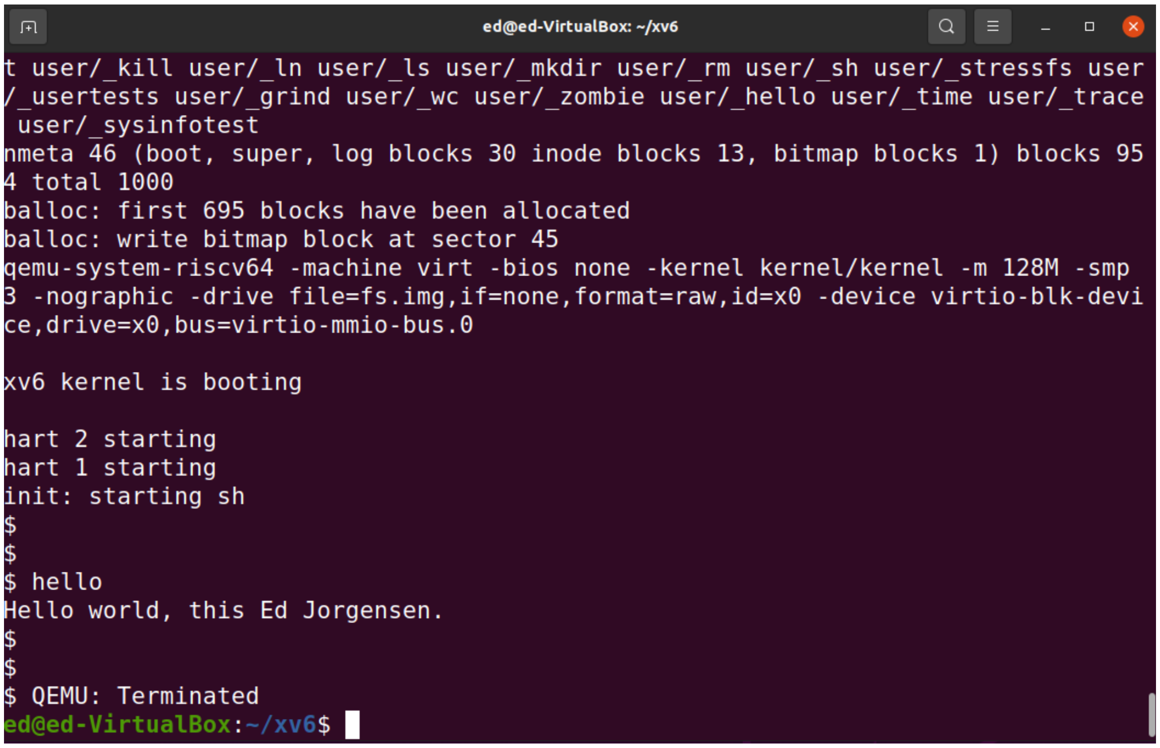
Committing Changes

The files you will need for this and subsequent lab assignments are distributed using the Git version control system. To learn more about Git, take a look at the Git user's manual, or, you may find this CS-oriented overview of Git useful. Git allows you to keep track of the changes you make to the code. For example, if you are finished with one of the project exercises, and want to checkpoint your progress, you can commit your changes by running:

```
$ cd xv6
xv6 $ git add Makefile
xv6 $ git add user/hello.c
xv6 $ git commit -am 'my solution for hello xv6'
Created commit 60d2135: my solution for hello xv6
 1 files changed, 1 insertions(+), 0 deletions(-)
xv6 $
```

Example Submission

The final submission should include a capture of current window (via built-in screenshot utility "Grab current window). As noted, your name should be included in the hello world output. The following is an example of the PNG file that should be submitted.



```
t user/_kill user/_ln user/_ls user/_mkdir user/_rm user/_sh user/_stressfs user
/_usertests user/_grind user/_wc user/_zombie user/_hello user/_time user/_trace
 user/_sysinfotest
nmeta 46 (boot, super, log blocks 30 inode blocks 13, bitmap blocks 1) blocks 95
4 total 1000
balloc: first 695 blocks have been allocated
balloc: write bitmap block at sector 45
qemu-system-riscv64 -machine virt -bios none -kernel kernel/kernel -m 128M -smp
3 -nographic -drive file=fs.img,if=none,format=raw,id=x0 -device virtio-blk-devi
ce,drive=x0,bus=virtio-mmio-bus.0

xv6 kernel is booting

hart 2 starting
hart 1 starting
init: starting sh
$
$
$ hello
Hello world, this Ed Jorgensen.
$
$
$ QEMU: Terminated
ed@ed-VirtualBox:~/xv6$
```