Tools Used

You'll need the RISC-V versions of QEMU 7.2+, GDB 8.3+, GCC, and Binutils.

Ubuntu 24 (or later)

```
sudo apt-get update && apt-get install git build-essential gdb-multiarch
qemu-system-misc gcc-riscv64-linux-gnu binutils-riscv64-linux-gnu
```

Installing on Windows

Students running Windows are encouraged to either install Linux on their local machine or use WSL 2 (Windows Subsystem for Linux 2).

We also encourage students to install the Windows Terminal tool in lieu of using Powershell/Command Prompt.

To use WSL 2, first make sure you have the Windows Subsystem for Linux installed. Then add Ubuntu 24.04 from the Microsoft Store. Afterwards you should be able to launch Ubuntu and interact with the machine.

IMPORTANT: Make sure that you are running version 2 of WSL. WSL 1 does not work with the labs. To check, run `wsl -l -v` *in a Windows terminal to confirm that WSL 2 and the correct Ubuntu version are installed.*

To install all the software you need for this class, run:

```
$ sudo apt-get update && sudo apt-get upgrade
$ sudo apt-get install git build-essential gdb-multiarch qemu-system-misc
gcc-riscv64-linux-gnu binutils-riscv64-linux-gnu
```

From Windows, you can access all of your WSL files under the *"\\wsl$\"* directory. For instance, the home directory for an Ubuntu 24.04 installation should be at *"\\wsl$\Ubuntu-24.04\home\<username>\"*.

Running a Linux VM

If you're running an operating system on which it's not convenient to install the RISC-V tools, you may find it useful to run a Linux virtual machine (VM) and install the tools in the VM.

Installing a Linux virtual machine is a two step process. First, get a virtualization platform; we suggest:

- **VirtualBox** (free for Mac, Linux, Windows) — Download page

Once the virtualization platform is installed, fetch a boot disk image for the Linux distribution of your choice.

- Ubuntu Desktop is one option.
- Ubuntu Server (No GUI)

This will download a file named something like ubuntu-24.04.1-desktop-amd64.iso. Start up your virtualization platform and create a new (64-bit) virtual machine. Use the Ubuntu image as a boot disk; the procedure differs among VMs but shouldn't be too difficult.

Installing on macOS

First, install developer tools:

```
$ xcode-select --install
```

Next, install Homebrew, a package manager for macOS:

```
$ /bin/bash -c "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

Next, install the RISC-V compiler toolchain:

```
$ brew tap riscv/riscv
$ brew install riscv-tools
```

The brew formula may not link into /usr/local. You will need to update your shell's rc file (e.g. ~/.bashrc) to add the appropriate directory to $PATH.

```
PATH=$PATH:/usr/local/opt/riscv-gnu-toolchain/bin
```

Finally, install QEMU:

```
brew install qemu
```

Testing your Installation

To test your installation, you can double check your installation is correct by running the following:

```
$ qemu-system-riscv64 --version
QEMU emulator version 7.2.0
```

And at least one RISC-V version of GCC:

```
$ riscv64-linux-gnu-gcc --version
riscv64-linux-gnu-gcc (Debian 10.3.0-8) 10.3.0
...
```

```
$ riscv64-unknown-elf-gcc --version
riscv64-unknown-elf-gcc (GCC) 10.1.0
...
```

```
$ riscv64-unknown-linux-gnu-gcc --version
riscv64-unknown-linux-gnu-gcc (GCC) 10.1.0
...
```

Boot xv6

You should be able to compile and run xv6. You can try this by following the instructions:

Fetch the git repository for the xv6 source for the lab:

```
$ git clone git://g.csail.mit.edu/xv6-labs-2024
Cloning into 'xv6-labs-2024'...
...
$ cd xv6-labs-2024
```

The files you will need for this and subsequent labs are distributed using the Git version control system. For each of the labs you will check out a version of xv6 tailored for that lab. To learn more about Git, take a look at the Git user's manual, or this CS-oriented overview of Git. Git allows you to keep track of the changes you make to the code. For example, if you are finished with one of the exercises, and want to checkpoint your progress, you can *commit* your changes by running:

```
$ git commit -am 'my solution for util lab exercise 1'
Created commit 60d2135: my solution for util lab exercise 1
 1 files changed, 1 insertions(+), 0 deletions(-)
$
```

You can view your changes with git diff, which displays changes since your last commit. git diff origin/util displays changes relative to the initial util code. origin/util is the name of the git branch for this lab.

Build and run xv6:

```
$ make qemu
riscv64-unknown-elf-gcc    -c -o kernel/entry.o kernel/entry.S
riscv64-unknown-elf-gcc -Wall -Werror -O -fno-omit-frame-pointer -ggdb
-DSOL_UTIL -MD -mcmodel=medany -ffreestanding -fno-common -nostdlib
-mno-relax -I. -fno-stack-protector -fno-pie -no-pie   -c -o kernel/start.o
kernel/start.c
...
riscv64-unknown-elf-ld -z max-page-size=4096 -N -e main -Ttext 0 -o
user/_zombie user/zombie.o user/ulib.o user/usys.o user/printf.o
user/umalloc.o
riscv64-unknown-elf-objdump -S user/_zombie > user/zombie.asm
riscv64-unknown-elf-objdump -t user/_zombie | sed '1,/SYMBOL TABLE/d; s/ .*
/ /; /^$/d' > user/zombie.sym
mkfs/mkfs fs.img README  user/xargstest.sh user/_cat user/_echo
user/_forktest user/_grep user/_init user/_kill user/_ln user/_ls
user/_mkdir user/_rm user/_sh user/_stressfs user/_usertests user/_grind
user/_wc user/_zombie
nmeta 46 (boot, super, log blocks 30 inode blocks 13, bitmap blocks 1)
blocks 954 total 1000
balloc: first 591 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
$
```

If you type ls at the prompt, you should see output similar to the following:

```
$ ls
.              1 1 1024
..             1 1 1024
README         2 2 2227
xargstest.sh   2 3 93
cat            2 4 32864
echo           2 5 31720
forktest       2 6 15856
grep           2 7 36240
init           2 8 32216
kill           2 9 31680
ln             2 10 31504
ls             2 11 34808
mkdir          2 12 31736
rm             2 13 31720
sh             2 14 54168
stressfs       2 15 32608
usertests      2 16 178800
grind          2 17 47528
wc             2 18 33816
zombie         2 19 31080
console        3 20 0
```

These are the files that **mkfs** includes in the initial file system; most are programs you can run. You just ran one of them: ls.

xv6 has no ps command, but, if you type Ctrl-p, the kernel will print information about each process. If you try it now, you'll see two lines: one for init, and one for sh.

To quit qemu type: **Ctrl-a x** (press Ctrl and a at the same time, followed by x).

References:
1. https://pdos.csail.mit.edu/6.1810/2024/tools.html
2. https://www.qemu.org/docs/master/system/targets.html