EECS 343 - xv6

A BIT ABOUT XV6

In this course you will have the chance to learn about and work with **xv6**, a simple Unix-like teaching operating system from MIT.

> "xv6 is a re-implementation of Dennis Ritchie's and Ken Thompson's Unix Version 6 (v6). xv6 loosely follows the structure and style of v6, but is implemented for a modern x86-based multiprocessor using ANSI C.
> xv6 is inspired by **John Lions's Commentary on UNIX 6th Edition** (Peer to Peer Communications; ISBN: 1-57398-013-7; 1st edition (June 14, 2000))."
>
> *From the xv6 README file.*

As with Lions' Commentary, there is a **xv6 book**. Keep in mind that the version of xv6 we use is slightly older than that of the book and so you may note some differences.

GETTING STARTED

(Project 1, Part 0)

To work on xv6, you will use two set of tools:

1. **QEMU**, an x86 emulator for running your kernel
2. A compiler toolchain (assembler, linker, C compiler, and debugger) for compiling and testing your kernel.

Some useful information about these tools can be found **here**. Fortunately, these tools are already installed for you on the T-Lab and Wilkinson Lab machines. Your TA has completed the xv6 projects on the lab machines and confirmed that everything is working, but please let us know ASAP if any problems arise with these tools.

We will now walk you through how to get started with xv6, which will involve the following steps:

1. **Download the xv6 source code.**
2. **Compile the xv6 kernel.**
3. **Run xv6 on QEMU.**
4. **Take a tour of xv6.**
5. **Exit xv6 and QEMU.**
6. **Run xv6 again, this time with GDB remote debugging enabled.**
7. **Exit xv6, QEMU and GDB.**

These instructions assume you are working from a lab machine, or at least SSH-ed into one. It might be possible to complete these instructions from another environment, but it will be tricky.

Without further ado, let's get going.

Download the source code:

```
$ wget http://www.aqualab.cs.northwestern.edu/component/attachments/download/696 -O
xv6.tar.gz
```

The -O option above allows you to specify how to name the downloaded file. You should now have a file named xv6.tar.gz. This distribution of xv6 is a copy from our friends at UW-Madison, which they have restructured to make it easier to navigate the code.

Ensure that your download worked properly by checking the SHA hash of the file:

```
$ sha256sum xv6.tar.gz
```

Confirm that the output is identical to this:
```
fee5c515f1b8092306113b96d92998f84740732b98ddae63d41b1f89f2c60371  xv6.tar.gz
```

Extract the files:

```
$ tar -xzvf xv6.tar.gz
```

In the xv6 directory you will find:

```
     FILES .gitignore include kernel Makefile Makefile~ README tools user version
```

Look at `FILES` using vim for a description of each directory.

Compile the xv6 kernel:

In the xv6 directory, type `make`. When the compilation is complete, you should now find two additional files in the xv6 directory:

```
     fs.img xv6.img
```

Run xv6 on QEMU:

The command to run xv6 on QEMU is:

```
$ make qemu-nox
```

The first time you will probably see this error:

```
***
*** Error: Couldn't find a working QEMU executable.
*** Is the directory containing the qemu binary in your
*** PATH or have you tried setting the QEMU variable in
*** Makefile?
***
```

As the error says, the QEMU executable wasn't found. This can be easily solved by either modifying your PATH or modifying the Makefile. We will show you how to modify the Makefile. First, open the Makefile in a text editor:

```
$ vim Makefile
```

Find the lines that look like this:

```
# If the makefile can't find QEMU, specify its path here
#QEMU :=
```

Uncomment the second line and modify it to look like this:

```
QEMU := /home/software/qemu/bin/qemu-system-i386
```

Save and quit out of the Makefile. Now you should be able to run xv6. Try the command again:

```
$ make qemu-nox
```

You may see a couple warnings, then you should see the following output lines as xv6 boots up:

```
xv6...
lapicinit: 1 0xfee00000
cpu1: starting
cpu0: starting
```

```
init: starting sh
$
```

Take a tour of xv6:

You are now in an xv6 shell. Type `ls` to see the list of programs that you can run. You should see a few familiar commands such as '`cat`' and '`mkdir`'. It is always good to start by reading the README:

```
$ cat README
```

Interesting stuff, eh? Now try running the rest of the provided programs to get a feel for what they do. If a program doesn't exit on its own, you can kill it by typing:

```
control+d
```

Exit xv6 and QEMU:

When you are done snooping around, you can exit xv6 by typing:

`control+a`
then release both keys and tap
`x`

Run xv6 again, this time with GDB remote debugging enabled:

The GNU Debugger (GDB) will be one of your best friends this quarter. Any operating systems programming project at any college in the universe is basically composed of three parts:

1. Stare confusedly at the provided code trying to figure out what is going on and come up with a plan of attack.
2. Write code.
3. Spend hours in GDB trying to find all the reasons your code isn't working properly.

Fortunately, xv6 allows you to "look inside" while it is running using GDB. This requires two terminals: one to run xv6 and another to run GDB with remote debugging.

First, run xv6 with remote debugging enabled:

```
$ make qemu-nox-gdb
```

You will notice that this time xv6 hangs during the boot-up process. It is waiting for you to make a remote connection using GDB, so let's do that. In another terminal window, in the xv6 directory, type:

```
$ gdb kernel/kernel
```

The first time you will probably get a warning that says:

```
...auto-loading has been declined by your `auto-load safe-path'...
```

Fortunately, it also tells you how to fix the problem:

```
To enable execution of this file add
    add-auto-load-safe-path /<path-to-xv6>/xv6/.gdbinit
line to your configuration file "/home/<username>/.gdbinit".
To completely disable this security protection add
    set auto-load safe-path /
line to your configuration file "/home/<username>/.gdbinit".
```

Go ahead and quit out of GDB so that you can follow the message instructions. You'll probably have to create a `.gdbinit` file in your home directory (there is already a separate .gdbinit in your xv6 directory, which you should NOT touch--leave it be). You should create (or, if it already exists, modify) `/home/<username>/.gdbinit` as suggested in the message. We recommend using the `add-auto-load-safe-path` method, which we have highlighted in red above.

Once you have fixed your .gdbinit file, you can run GDB again:

```
$ gdb kernel/kernel
```

This time, when GDB starts up, you should see the following lines in the output:

```
Connecting to QEMU
(gdb) target remote localhost:25273   (<--port number may differ)
```

Now let's insert a breakpoint in the `exec` function. This kernel-level function is invoked whenever a new process is executed. In GDB, you can set the breakpoint by typing:

```
(gdb) b exec
```

We are now ready to let xv6 finish booting. In GDB, let xv6 continue by typing:

```
(gdb) c
```

In your xv6 terminal, you should see the system begin to boot:

```
xv6...
lapicinit: 1 0xfee00000
cpu1: starting
cpu0: starting
```

Then it hits the `exec` function for the first time, triggering the breakpoint. In GDB, you should see:

```
Breakpoint 1, exec (path=0x1c "/init", argv=0xff0e98) at kernel/exec.c:11
11 {
```

Notice the part that says: `path=0x1c "/init"`
This means that `exec` is about to execute the `init` process. We have just discovered the first process that xv6 executes during boot-up!

Keep typing 'c' or "continue" in GBD until xv6 reaches the shell prompt, noting the process(es) executed along the way. Once at the shell prompt, you can type an xv6 command to run a program, which should trigger your breakpoint again. You can then type 'c' in GDB to allow the program to run to completion. NOTE: Some programs may behave differently when remote debugging is in use.

You can now repeat your previous tour of xv6, but this time you can set breakpoints and use other GDB features to do some more thorough snooping around.

Exit xv6, QEMU, and GDB:

When you've had enough, you can exit xv6 and QEMU as before by typing `control+a` then `x`. You can exit GDB by typing q then `enter`.

A NOTE ON PLATFORMS AND WORKING REMOTELY

For your projects, we STRONGLY recommend using the lab machines. If the labs are full, you can still work remotely by SSH-ing into a lab machine. That being said, it may be possible to complete the projects on your personal machine. The easiest approach is to install xv6 in a recent version of the Linux operating system. If you are not running Linux, you can run Ubuntu inside a VM (like the freely available **VirtualBox**). The next easiest approach (which the TA couldn't get to work after several hours) is to try to run xv6 and QEMU on Mac OS X. There are a few websites that provide instructions (e.g., **here** and **here**). If you run Windows on your personal machine, your best bet is to SSH into the lab machines (recommended!), or use a Linux VM as mentioned above.