Home

Instructions for building xv6 on Openlab (andromeda) servers

NoteUp until summer 2018, Openlab machines were running 32bit operating systems. In that environment it was very hard (lots of dependencies) to build the Qemu emulator required to run the xv6 operating system. Therefore, previously we relied on a Vagrant VM, i.e. a full 64bit Linux distribution that was running inside a hardware-assisted virtualization container (VT-x), inside which we built Qemu and ran xv6. However, as Openlab machines were updated, it no longer makes sense to start another VM. Now we can simply build Qemu on the andromeda machines and run xv6 without Vagrant. The instructions below are now the recommend way to work with this class.

Xv6 is a real operating system kernel, and hence, it needs real hardware to boot. Fortunately, today we can emulate hardware in software. Programs like QEMU can emulate functionality of the real physical CPU in software. I.e., QEMU implements the normal CPU loop similar to the one we discussed in class: fetches an instruction pointed by the instruction pointer register (EIP), decodes it, performs all permission and condition checks, computes the outcome, increments EIP and continues to the next instruction. Like a real PC platform, QEMU emulates hardware boot protocol. QEMU starts by loading the disk sector number 0 into the memory location 0x7c00 and jumping to it. Xv6 takes it from there. At a high level, for xv6 it does not matter if it runs on the real hardware or under QEMU. Of course, emulation is slower than real hardware, but besides that as long as QEMU implements the logic of the CPU correctly we do not see any deviations from a real baremetal execution. Surprisingly, QEMU is reasonably fast, so you as a human barely notice the difference.

SSH into Openlab (andromeda-XX)

Currently vagrant is installed on Andromeda machines 1 through 75: **andromeda-1.ics.uci.edu** to **andromeda-75.ics.uci.edu**.

Before you begin, you need to select an andromeda server for yourself. We are following the following method to select a server :

- Find out your student ID
  (Let's say: 66541280)
- Evaluate serverNumber = (studentIDNumber mod 74) +1
  Ex: 66541280 => 37
- Your server name is : **andromeda-{serverNumber}.ics.uci.edu**
  Ex: andromeda-37.ics.uci.edu

To configure your xv6 environment, login to your server

```
$ ssh UCInetID@andromeda-XX.ics.uci.edu
```

Troubleshooting your Andromeda ssh connection

If you get something like

```
 Permission denied
(publickey,gssapi-keyex,gssapi-with-mic,password)
```

message from ssh and you're connecting from outside of campus, you need to need to either use AnyConnect VPN to tunnel in or setup an SSH keypair for passwordless login. Openlab machines do not accept outside password login attempts.

Clone and build Qemu

To run xv6 we need to compile and install a version of the QEMU emulator. Default QEMU's debugging facilities, while powerful, are somewhat immature, so it is highly recommend you use a patched version of Qemu which is maintained by MIT instead of the stock version that may come with your distribution. From inside your home folder create a new directory for this class. I suggest you start in your home folder (unless you know what you're doing this is a good way to go)

```
 cd ~
mkdir cs238p
```

Change into this new directory

```
cd cs238p
```

Clone the MIT's qemu distribution into the qemu folder

```
git clone https://github.com/mit-pdos/6.828-qemu.git qemu
```

Change into qemu folder and initialize another git submodule

```
cd qemu
git submodule update --init pixman
```

Configure qemu with minimal settings and the prefix pointing to the install folder

```
./configure --disable-kvm --disable-werror --prefix=/home/<YourUCInetID>/cs238p/qemu-install --target-list="i386-softmmu x86_64-softmmu"
```

For example, for me this becomes:

```
./configure --disable-kvm --disable-werror --prefix=/home/aburtsev/cs238p/qemu-install --target-list="i386-softmmu x86_64-softmmu"
```

Make and install qemu (this will take some time)

```
make -j 8
make install
```

To make qemu accessible from other programs add it to your path. You can either add it every time you log in to the andromeda machine by exporting the PATH variable, or you can add this line to .bash_profile file in your home folder.

```
export PATH=$PATH:$HOME/cs238p/qemu-install/bin
```

If you decide to add the PATH permanently to your .bash_profile the line should look something like

```
export PATH=$HOME/cs238p/qemu-install/bin:/sbin:/bin:/usr/sbin:/usr/bin:/usr/games:/usr/local/sbin:/usr/local/bin:/usr/X11R6/bin:$HOME/bin:$PATH
```

Clone, build, and boot xv6

Change back into the `cs238p` folder and clone xv6 repository.

```
cd ~/cs238p
git clone git://github.com/mit-pdos/xv6-public.git
Cloning into xv6...
...
```

Build xv6 (you only will see a bunch of output from the make command that invokes the gcc compiler on all kernel files, links the kernel with ld, creates the kernel file system with all user-level programs that will be available inside xv6, and so on):

```
$ cd xv6-public
$ make
```

You're now ready to run xv6.

```
$ make qemu-nox ...
```

You are now running xv6 on top a hardware platform that is emulated by the Qemu emulator. You are now done with the xv6 setup and can continue moving to any homeworks that are currently assigned.

You can find more information about QEMU monitor and GDB debugger here, feel free to explore them.

Old and Deprecated: Instructions for setting up Vagrant xv6 environment

Note: this is an old way to run xv6 on Openlab, it's no longer recommended

Up until summer 2018, Openlab machines were running 32bit operating systems. In that environment it was very hard (lots of dependencies) to build the Qemu emulator required to run the xv6 operating system. Therefore, we decided (with Openlab admins) that the easiest way to provide xv6-compatible environment is to run it inside a Vagrant VM, i.e. a full 64bit Linux distribution that was running inside a hardware-assisted virtualization container (VT-x). However, as Openlab machines are updated, it no longer makes sense to start another VM. Now we can simply build Qemu on the andromeda machines and run xv6 without Vagrant. Now we recommend that you follow the native "xv6 on Openlab" instructions above.

Xv6 is a real operating system kernel, and hence, it needs real hardware to boot. Fortunately, today we can emulate hardware in software. Programs like QEMU can emulate functionality of the real physical CPU in software. I.e., QEMU implements the normal CPU loop similar to the one we discussed in class: fetches an instruction pointed by the instruction pointer register (EIP), decodes it, performs all permission and condition checks, computes the outcome, increments EIP and continues to the next instruction. Like a real PC platform, QEMU emulates hardware boot protocol. QEMU starts by loading the disk sector number 0 into the memory location 0x7c00 and jumping to it. Xv6 takes it from there. At a high level, for xv6 it does not matter if it runs on the real hardware or under QEMU. Of course, emulation is slower than real hardware, but besides that as long as QEMU implements the logic of the CPU correctly we do not see any deviations from a real baremetal execution. Surprisingly, QEMU is reasonably fast, so you as a human barely notice the difference.

To run xv6 we need to compile and install a version of the QEMU emulator. Due to some compatibility issues it is impossible to compile QEMU directly on the Openlab (andromeda-XX) machines. Instead, we will run yet another virtual machine called Vagrant on the Openlab servers. Vagrant is a user-friendly interface to the VirtualBox virtual machine monitor. You will be able to start Vagrant on any of the andromeda machines. Vagrant will boot into a version of the Ubuntu Linux system.

Currently vagrant is installed on Andromeda machines 1 through 75: **andromeda-1.ics.uci.edu** to **andromeda-75.ics.uci.edu**.

Before you begin, you need to select an andromeda server for yourself. We are following the following method to select a server :

- Find out your student ID
  (Let's say: 66541280)
- Evaluate serverNumber = (studentIDNumber mod 74) +1
  Ex: 66541280 => 37
- Your server name is : **andromeda-{serverNumber}.ics.uci.edu**
  Ex: andromeda-37.ics.uci.edu

To configure your xv6 environment, login to your server

```
$ ssh UCInetID@andromeda-XX.ics.uci.edu
```

Troubleshooting your Andromeda ssh connection

If you get something like

```
 Permission denied
(publickey,gssapi-keyex,gssapi-with-mic,password)
```

message from ssh and you're connecting from outside of campus, you need to need to either use AnyConnect VPN to tunnel in or setup an SSH keypair for passwordless login. Openlab machines do not accept outside password login attempts.

Moving forward

I suggest you create a new folder for your cs238p homeworks, like:

```
UCInetID@andromeda-XX$mkdir cs238p
```

Change into that directory:

```
UCInetID@andromeda-XX$cd cs238p
```

Fetch a version of the vagrant environment that explains to vagrant what kind of virtual machine you're planning to run:

```
UCInetID@andromeda-XX$ wget http://www.ics.uci.edu/~aburtsev/238P/hw/xv6-vagrant-master.tgz
UCInetID@andromeda-XX$ tar -xzvf xv6-vagrant-master.tgz
```

Change into the new folder

```
UCInetID@andromeda-XX$ cd xv6-vagrant-master
```

Change the name of the vagrant VM to something unique (otherwise we all end up with the same VM and vagrant is confused). In the `Vagrantfile` file change the following line

```
      vb.name = "xv6_box_anton" # <--- You should change this to make VM names unique
```

Start vagrant VM (this will take several minutes as it is building a patched version of QEMU inside)

```
UCInetID@andromeda-XX$ vagrant up
```

Common Troubleshooting

Clearing any previously set forwarded ports...

If vagrant fails with the following message:

```
==> default: Clearing any previously set forwarded ports...
Vagrant cannot forward the specified ports on this VM, since they
would collide with some other application that is already listening
on these ports. The forwarded port to 20000 is already in use
on the host machine.

To fix this, modify your current project's Vagrantfile to use another
port. Example, where '1234' would be replaced by a unique host port:

  config.vm.network :forwarded_port, guest: 26001, host: 1234
```

Go ahead with the suggested fix. Change the following line in the Vagrantfile setting the host port to something random below 64000:

```
  config.vm.network "forwarded_port", guest: 26001, host: 30000
```

Timing out while waiting for machine to boot

If you're getting something like the above message, maybe your Andromeda server doesn't have VT-x (virtualization extensions enabled, or doesn't have the Linux kernel that supports them). You can check by doing

```
ls /dev/kvm
```

The kvm device should be there. Pick another andromeda machine at random, i.e., 74 should be ok.

A VirtualBox machine with the name '/*my name*/' already exists

If you're getting something like:

```
A VirtualBox machine with the name '/*my name*/' already exists.  Please
use another name or delete the machine with the existing name, and try
again.
```

First check if your VM is still running with two commands ps and grep and combine them with a pipe like this:

```
ps -aux | grep VBoxHeadless
```

ps lists processes in the system, -a -- list all, -u show user who started it, -x -- list even the ones that don't have an associated terminal (i.e., VBoxHeadless doesn't have it). grep filters only strings that ps outputs to the specific filter "VBoxHeadless". On andromeda-19 it shows you're still running something.

Then you can try halting vagrant with

```
vagrant halt
```

If it doesn't work, try maybe killing the process like

```
kill <process_id>
```

You can get `process_id` from ps. If simple kill doesn't work try

```
kill -9 <process_id>
```

If a process can ignore kill signal (i.e., normally the OS gives a process a chance to save it's state before exiting with this signal). kill -9 will kill it without allowing to ignore. Finally, if you really need to destroy all traces of an exising vagrant VM you can do

```
vagrant destroy
```

Logging into Vagrant VM

If vagrant VM is up, you're ready to log in inside and start working on your xv6 Linux environment. Log in inside the vagrant VM. From the same folder where Vagrantfile is (i.e., from cs238p/xv6-vagrant-master) type

```
UCInetID@andromeda-XX$ vagrant ssh
```

Now you're inside the Linux Ubuntu 12.04.5 LTS. Your new vagrant machine should have everything you need to compile and run your xv6 code. Vagrant automatically shares the directory of your host machine where you put the `Vagrantfile` file (i.e., the `cs238p/xv6-vagrant-master` folder) as the `/vagrant` directory of the vagrant VM.

Note: power down your VM when you're done!

When you're done playing with your xv6 setup, or more specifically with the xv6 Vagrant VM don't forget to power it down (we have hundreds of people working on these assignments, plus other classes use openlab machines too.

```
vagrant halt
```

Boot xv6

From inside your Vagrant VM fetch the xv6 source:

```
vagrant@odin$ cd /vagrant
vagrant@odin$ mkdir cs238p
vagrant@odin$ cd cs238p
vagrant@odin$ git clone git://github.com/mit-pdos/xv6-public.git
Cloning into xv6...
...
```

Build xv6 (you only will see a bunch of output from the make command that invokes the gcc compiler on all kernel files, links the kernel with ld, creates the kernel file system with all user-level programs that will be available inside xv6, and so on):

```
vagrant@odin$ cd xv6-public
vagrant@odin$ make
```

You're now ready to run xv6.

```
vagrant@odin$ make qemu-nox
...
```

You can find more information about QEMU monitor and GDB debugger here, feel free to explore them.

Xv6 on your own Linux system

While we provide instructions for how to use Andromeda machines, you are more than welcome to configure and run xv6 on your own laptop, desktop, or VM. If you decide to use your own environment, see the instructions on the xv6 tools page for how to set up xv6. I've successfully built xv6 on my Ubuntu 14.04 LTS and later on Ubuntu 16.04 LTS. I had to install the following packages in order to build QEMU: libz-dev, libtool-bin, libtool, libglib2.0-dev, libpixman-1-dev, libfdt-dev.

XV6 in Docker

In case you want to use run xv6 on your own machine using docker containers, you can try it out as well. I have successfully built XV6 using the **grantbot/xv6** image hosted in the docker hub

In my localmachine, I downloaded the XV6 source code as follows :

```
localhost$ mkdir XV6_Dev
localhost$ cd XV6_Dev
localhost$ git clone git://github.com/mit-pdos/xv6-public.git
Cloning into xv6...
...
```

Next, you will need to setup Docker, if you don't have it already on your machine. I followed the instructions from here. You will find similar instructions for other OS as well in the docker website. Once you have the setup ready, download the **grantbot/xv6** image using

```
docker pull grantbot/xv6
```

Then you can start the container using

```
docker run -v '/{Path to local XV6 folder}/XV6_Dev':/home/a/XV6_Dev/ -i -t grantbot/xv6
```

Once you have the bash prompt you can type the following to start XV6,

```
cd ~/XV6_Dev/
make qemu-nox
```

Updated: October, 2018