Building and Running Xinu for BBB

To compile Xinu, you need a *cross compiler*; a secondary compiler targeting an architecture different from the one the computer is running on. While the compiler on your computer likely targets i386 or x64, for BBB, we need a compiler that targets the ARM ISA. Specifically, we can use the *arm-none-eabi* target of GCC. It is also important that the compiler not use any built-in Linux libraries (like usually happens by default), since Xinu is running by itself on bare hardware. This is what the "none-eabi" signifies.

The Hard Way

Build GCC from source. You will need to build binutils first, along with some other dependencies for your platform. Some OSes may have a pre-built *arm-none-eabi-gcc* by default – for example, on Mac, Homebrew has a tap for it, and on ArchLinux there is a package for it.

The Easy Way

Fortunately, the Xinu .tar you download already has a directory called *cross_compiler* containing all the *arm-none-eabi* binaries needed. However, these binaries were compiled for a 32-bit Linux target, meaning you probably have to install a Virtual Machine.

Setting up the Virtual Machine

For compiling Xinu, I will use the newest 32-bit version of Ubuntu. If you prefer a specific Linux distro, it will likely work, as long as it is the 32-bit version. Note though, that I have only tested on the Ubuntu described below.

1. Download the newest appropriate version of VirtualBox from: https://www.virtualbox.org/wiki/Downloads

   Note for Windows 10: If you have Windows Hyper-V installed, VirtualBox may not work properly (it will crash your computer). One possible solution is to set up the VM through Hyper-V Manager, or disable Hyper-V in the Programs & Features Control Panel.

2. In VirtualBox, click the **New** button in the top left. Give the VM a **Name**, and select **Linux** for type and **Ubuntu (32-bit)** for Version.

3. Run through the setup instructions. The default amount of memory should be fine, and be sure to create a virtual hard disk with the default format and size.

4. Download the newest 32-bit desktop image of Ubuntu from http://mirror.pnl.gov/releases/16.04.1/ - the download is in the first section at the top of the page, and is called **32-bit PC (i386) desktop image**.

5. Make sure your new VM is selected in the side bar, and click the **Settings** button.

6. Select the **Storage** tab, and find the **Controller: IDE** choice. Click on the **Empty** label below it.

7. To the right, next to **Optical Drive:**, click the icon that looks like a CD, and select the Ubuntu .iso you just downloaded.

8. Follow the install instructions for Ubuntu and reboot.

9. From the VirtualBox dropdown menu, select **Device > Insert Guest Additions CD Image**. A popup should appear in the VM; select run and allow the VirtualBox tools to install.

Getting Ready to Compile

1. Download the ARM version of Xinu (http://www.xinu.cs.purdue.edu/files/Xinu-code-BeagleBoneBlack.tar.gz) and untar it somewhere on the VM.

2. Open a new terminal window and navigate to the Xinu **/compile** directory.

3. Run the command **make clean** to remove the pre-built version of Xinu.

4. The default Ubuntu install does not have all the tools needed to build; run **sudo apt-get install flex bison gawk** to install the needed programs.

5. Run **make**. Xinu should successfully compile and create its set of "xinu" executables. Every time you make a change to the Xinu source, run **make** again and your changes should be reflected in the executable. If something goes wrong, try running **make clean** to start building from scratch.

Work Faster with a Shared Folder

Consider creating a shared folder through VirtualBox to put the Xinu directories in; this way, you can edit and upload on your host OS, and just use the Ubuntu instance for compiling. The shared folder will end up in **media/sf_*foldername*.** Alternatively, if you were having issues uploading to the BBB, it may be easier to do it all in the VM (just be sure to enable the USB-TTY cable as a device in VirtualBox).

Cannot initialize the Ethernet PHY

If you have gotten Xinu running on your BBB, you likely ran into this problem; Xinu would boot, but stop after this error message, and then panic on any further action. This is the default behavior of Xinu; if it is not connected to a physical Ethernet network, it will not boot to the shell. Thankfully, we can edit the source and remove the requirement (for now, this won't affect anything in terms of the labs and assignments). I have written a patch file to do this. This patch will also set up the structure for your *main.c* file for Lab 01.

Download the file **lab01.patch** from Canvas.
In the virtual machine, **make sure you are in the main Xinu directory**.
Run the following command: **patch -p1 < *path/to/patch*/lab01.patch**

The Result

Investigate the patch file; what does this change in the boot process? This disables the entire Xinu network subsystem; calling functions like *ping* in the shell will cause a panic. Is it necessary to disable the entire network subsystem, just to prevent Ethernet from initializing?

Note that the "failed to initialize Ethernet PHY" message still appears, followed by a DHCP and IP error (this will appear about 15 seconds later). **This is fine;** your main will start running right after. What other parts of Xinu's boot process could be changed to prevent Ethernet init? Try looking through the different functions Xinu goes through during boot.

HINT: Xinu starts executing the assembly file **system/start.S**, and eventually ends with starting the shell in **system/main.c**. If you can't find the actual definition for a call, try searching for it with **grep -rnw . -e "*functionname*"**