Assignment #1 - Getting Started:
Environment Configuration & First Kernel Program
Grade Weight: 50pts

Learning Objectives:
- Gain proficiency leveraging programming environment on the Beagle Board Black (BBB). This is the programming environment that we will use throughout the semester.
- Understand how to compiling and execute Linux Kernel Modules (LKM) and gain a practical understanding of some of the core difference between kernel-space and user-space.

Assignment Description:
In this assignment, you will build your first program that runs inside of the Linux Kernel. This code will be an extension of the operating system and will run directly on the microcontroller board (Beagle Board Black). Kernel-level code is privileged, in that the software has permissions to do things that regular user program can not do directly. For example, kernel code i handles allocations of hardware resources (processes running on the CPU, accessing memory, etc..), and provides an interface to the user-programs that are requesting accessing to hardware resources. Due to the critical takes performed in the OS kernel there are special considerations including reliability, performance and security concerns when writing kernel-level programs. In fact, kernel-programming can be dangerous in that a mistake can jeopardize your entire system (however, in this class we will always be able to reset it back to the original state). To get started, this assignment will walk you through the steps of configuring your board and executing your first program in the kernel.

Class Software Image:
All students will use the same software image for their Beagle Board Black (BBB). We will be writing Linux Kernel code on these boards. Writing kernel-level code requires certain software dependencies (such as the kernel header files). Also kernel development varies based on the Linux version. To prevent headaches, we will be consistent and have the entire class use the exact same Linux image. It will contain the required dependencies for writing kernel software. Specifically, we are using Anstrom distribution of Linux that using kernel version 3.8.   Anstrom Linux is specially designed for resource constrained hardware, such as embedded devices.

To image your board, you will need to acquire the SD card from the instructor or TA.

Put the SD card in the SD slot of your BBB and turn it on. As shown in the image below.
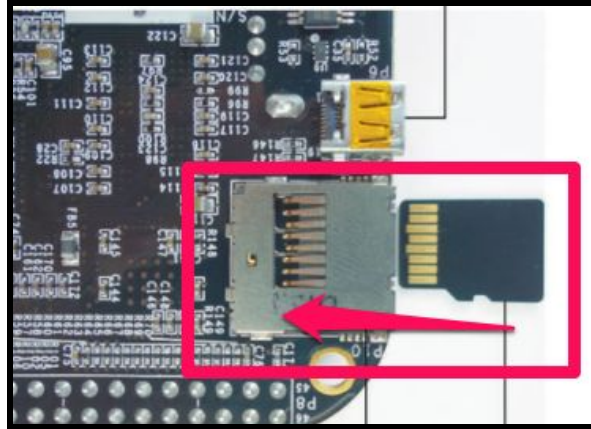
The LEDs on the board will flash in sequence for about 20 minutes, which your board is being imaged. After that completes, remove the SD card and restart your board. Make sure to return the SD card to the instructor or TA. Make sure your remove the SD card before you restart your machine. Otherwise, it will proceed to reimage your board again and you'll have to wait another 20 minutes for the imaging to complete.

Connecting to the Board (via SSH)

Connect your BBB to your computer using the USB cable and allow 5 to 10 seconds for the board to boot. Then follow the instructions below based on your computer's operating system. If it doesn't connect successfully, you might need to download the USB driver shown in the next section.

For Mac Systems:

In Macs, there is no need to download anything. You can simply use a built-in application called **Terminal**.

To start Terminal, go to your Mac's Applications folder => click on the Utilities folder => then click on Terminal.

Use the following command: ssh debian@192.168.7.2

For Linux Systems:

For Linux, you should not need to download anything. You most likely already have OpenSSH installed on your machine.
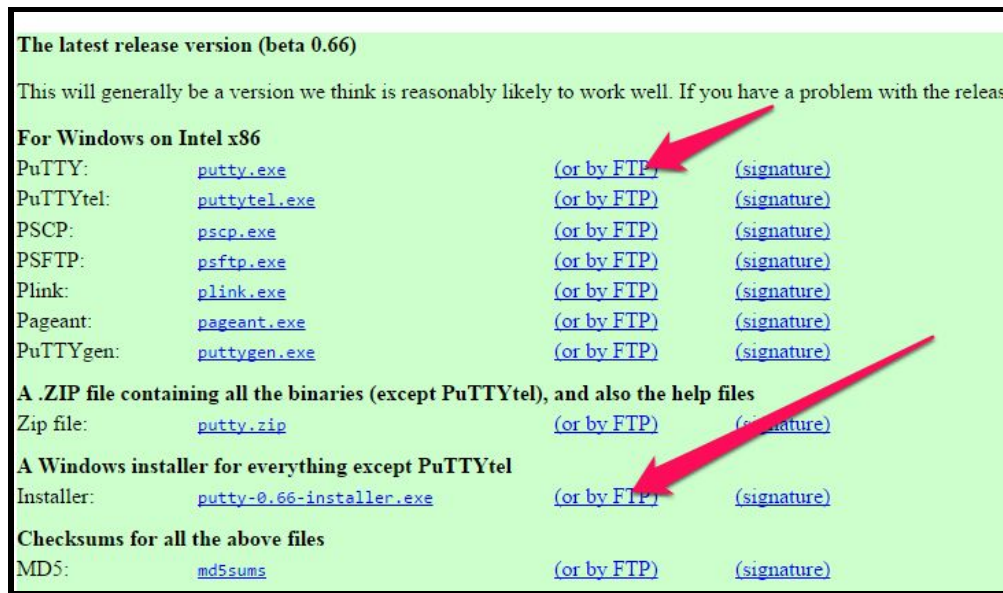
In an open terminal, type in the following command:

Use the following command: ssh debian@192.168.7.2

For Windows Systems:

Download Putty: http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html
(Note: This software is already available on UAlbany library machines)

The latest release version (beta 0.66)

This will generally be a version we think is reasonably likely to work well. If you have a problem with the releas

For Windows on Intel x86

PuTTY:	putty.exe	(or by FTP)	(signature)
PuTTYtel:	puttytel.exe	(or by FTP)	(signature)
PSCP:	pscp.exe	(or by FTP)	(signature)
PSFTP:	psftp.exe	(or by FTP)	(signature)
Plink:	plink.exe	(or by FTP)	(signature)
Pageant:	pageant.exe	(or by FTP)	(signature)
PuTTYgen:	puttygen.exe	(or by FTP)	(signature)

A .ZIP file containing all the binaries (except PuTTYtel), and also the help files

Zip file:	putty.zip	(or by FTP)	(signature)

A Windows installer for everything except PuTTYtel

Installer:	putty-0.66-installer.exe	(or by FTP)	(signature)

Checksums for all the above files

MD5:	md5sums	(or by FTP)	(signature)

Connect to your BBB using IP address 192.168.7.2 and the SSH protocol (port #22).

NOTE - It is recommended on your personal machine to save this configuration session. Just type a name into the Saved Sessions Box and click the Save button. This will save you time, since you won't have to retype the IP address every time you connect to your board.
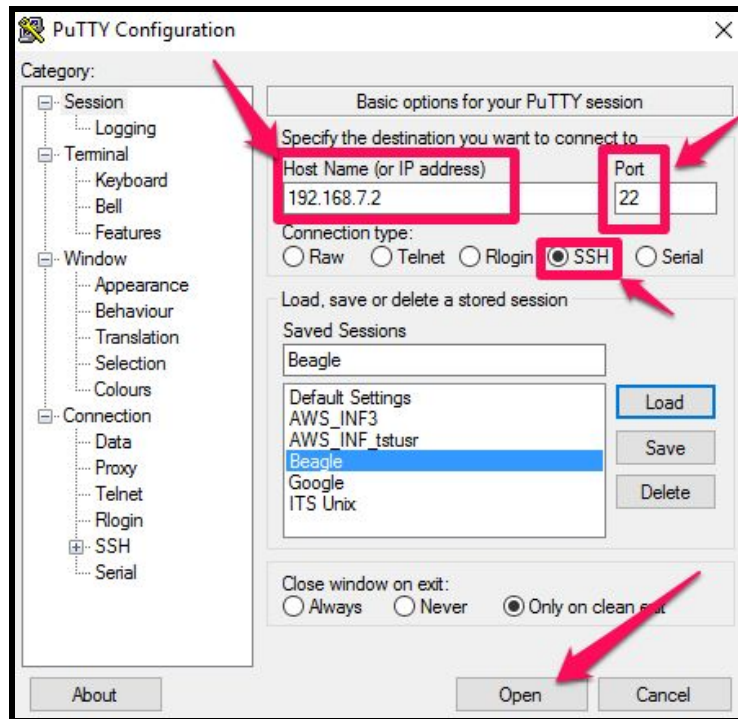
USB Driver

If your board didn't not connect successfully, you probably need the USB driver available on this website. http://beagleboard.org/getting-started

User Credentials:

You will need to log in using the username "debian" and the password "temppwd".

Setting the Clock

Important - Complete this set every time your board has lost power!

You have now established a terminal connection to your BeagleBone Black (BBB). This is an ultra-low cost computer. This means that some standard computer features had to be cut out in order to reduce costs. For example, your laptop and computer have a little coin-battery-powered 'Real Time Clock' (RTC) module, which keeps time even when the power is off, or the battery removed. To keep costs low and the size small, a battery-backed RTC is not included with the BBB. Instead, the BBB is intended to be connected to the Internet via Ethernet or WiFi, updating the time automatically from the global **ntp** (network time protocol) servers.

In our case, we will not always have the luxury to easily connect to the internet, so we are going to manually set the clock. Complete the following commands to set the date and time. The time below is setting for July 7th at 9:08PM. Update your clock to the current date / time.

It's important to complete the clock set correctly. When we compile our Linux Kernel Modules (LKM), the build takes the time into account the to determine which files are out-of-date that need to be compiled. If the system clock is out-of-date, the build will likely fail.

You will need to perform this step every time your BBB has lost power.

Moving Files To/From Your Board

Download the hello world kernel program: hello.tar

You will need some software to move files back and forth from your computer to your BBB. It is recommended that you use FileZilla. This software is free and available on the university library computers. This software allows you to transfer files using SFTP (Secure File Transfer Protocol).

Download: https://filezilla-project.org/download.php

After you install FileZilla, follow the steps to configure and connect to your board
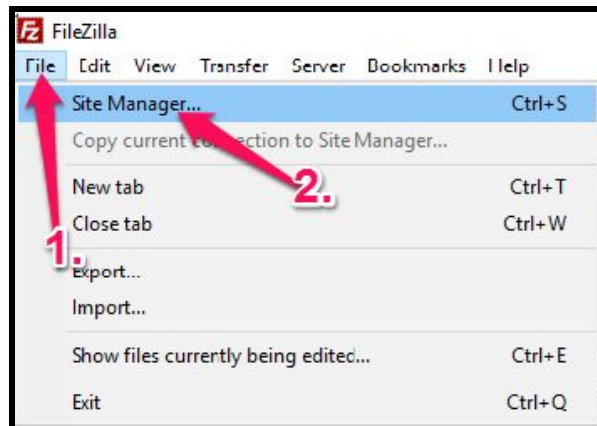
Remember, the credentials are username: debian and password: temppwd
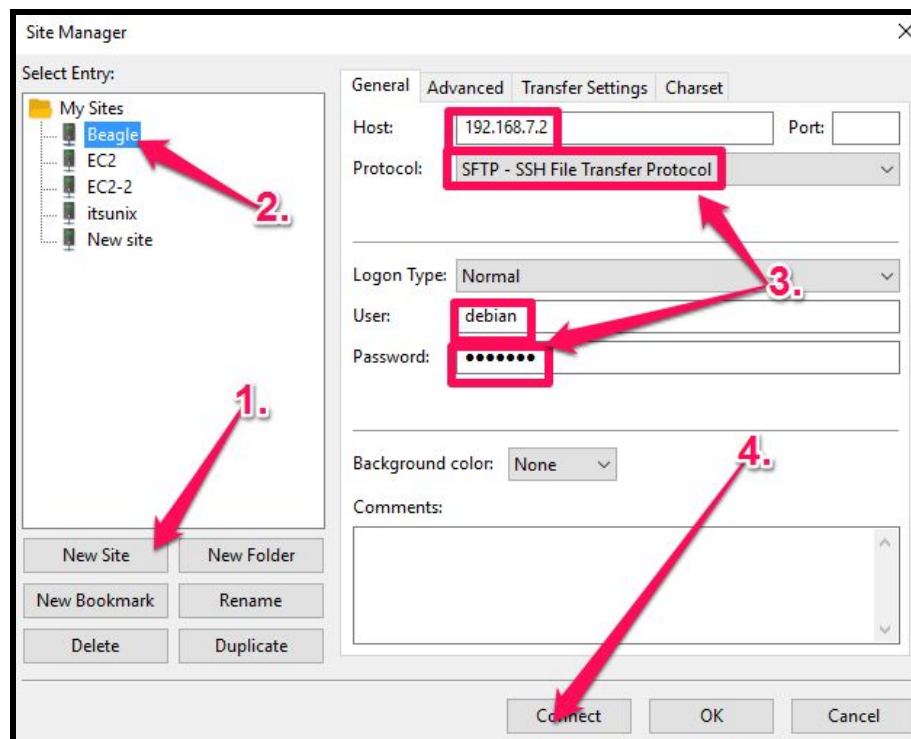
Transfer the hello world program .tar file to your BBB.

Build Your First Kernel Module

We are now going to run our first "hello world program" in the Linux Kernel.

First extract the tar file using the following command:

```
debian@beaglebone:~$ tar -xvf hello.tar
hello/
hello/hello.c
hello/Makefile
debian@beaglebone:~$
```

Inside the hello directory, you will see two files (hello.c and a Makefile).

Make the hello program by type the "make" command. You can see from the display that the makefile is using some Linux kernel dependencies to compile the program.

```
debian@beaglebone:~/hello$ make
make -C /lib/modules/3.8.13-bone70/build M=/home/debian/hello modules
make[1]: Entering directory `/usr/src/linux-headers-3.8.13-bone70'
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/debian/hello/hello.mod.o
  LD [M]  /home/debian/hello/hello.ko
make[1]: Leaving directory `/usr/src/linux-headers-3.8.13-bone70'
debian@beaglebone:~/hello$
```

After you successfully build the hello world program type "ls". You will see a lot of files that were generated, including a .ko file which is a kernel-level object file.

Load the kernel module you just built using the following command. This causes the hello world program to run into the kernel, however, you will not see anything displayed. The **insmod** command stands for insert module.

```
debian@beaglebone:~/hello$
debian@beaglebone:~/hello$ sudo insmod hello.ko
debian@beaglebone:~/hello$
```

Use the **lsmod** command to see which kernel modules are currently running. You should see "hello" listed.

```
debian@beaglebone:~/hello$ sudo lsmod
Module                  Size  Used by
hello                    755  0
g_multi                50407  2
libcomposite           15028  1 g_multi
omap_rng                4062  0
mt7601Usta            639170  0
debian@beaglebone:~/hello$
```

We can see the print out from the hello world program by looking at the kernel logs. Type the following command to see the last 10 lines from the kernel log. As you can see, the Hello World was displayed in the log.

```
debian@beaglebone:~/hello$ tail /var/log/kern.log
Mar  1 20:46:02 beaglebone kernel: [   11.364329]  gadget: high-speed config #1: Multifunction with RNDIS
Mar  1 20:46:09 beaglebone systemd[1]: Startup finished in 3s 45ms 330us (kernel) + 14s 965ms 524us (users
Mar  1 20:46:17 beaglebone kernel: [   26.202679] net eth0: initializing cpsw version 1.12 (0)
Mar  1 20:46:17 beaglebone kernel: [   26.210634] net eth0: phy found : id is : 0x7c0f1
Mar  1 20:46:17 beaglebone kernel: [   26.210677] libphy: PHY 4a101000.mdio:01 not found
Mar  1 20:46:17 beaglebone kernel: [   26.215751] net eth0: phy 4a101000.mdio:01 not found on slave 1
Mar  1 20:46:17 beaglebone kernel: [   26.251880] IPv6: ADDRCONF(NETDEV_UP): eth0: link is not ready
Jul  7 18:32:34 beaglebone kernel: [ 6171.983465] hello: module license 'unspecified' taints kernel.
Jul  7 18:32:34 beaglebone kernel: [ 6171.983510] Disabling lock debugging due to kernel taint
Jul  7 18:32:34 beaglebone kernel: [ 6171.989755] Hello world 1. ←
debian@beaglebone:~/hello$
```

You can remove your linux kernel module using the **rmmod** command.

```
debian@beaglebone:~/hello$ sudo rmmod hello
debian@beaglebone:~/hello$
```

Take a look at the kernel log again, you will see the "Goodbye World" message now displayed.

```
debian@beaglebone:~/hello$ tail /var/log/kern.log
Mar  1 20:46:09 beaglebone systemd[1]: Startup finished in 3s 45ms 330us (kernel) + 14s 965ms 524us (u
Mar  1 20:46:17 beaglebone kernel: [   26.202679] net eth0: initializing cpsw version 1.12 (0)
Mar  1 20:46:17 beaglebone kernel: [   26.210634] net eth0: phy found : id is : 0x7c0f1
Mar  1 20:46:17 beaglebone kernel: [   26.210677] libphy: PHY 4a101000.mdio:01 not found
Mar  1 20:46:17 beaglebone kernel: [   26.215751] net eth0: phy 4a101000.mdio:01 not found on slave 1
Mar  1 20:46:17 beaglebone kernel: [   26.251880] IPv6: ADDRCONF(NETDEV_UP): eth0: link is not ready
Jul  7 18:32:34 beaglebone kernel: [ 6171.983465] hello: module license 'unspecified' taints kernel.
Jul  7 18:32:34 beaglebone kernel: [ 6171.983510] Disabling lock debugging due to kernel taint
Jul  7 18:32:34 beaglebone kernel: [ 6171.989755] Hello world 1.
Jul  7 23:14:53 beaglebone kernel: [23111.212780] Goodbye world 1. ←
debian@beaglebone:~/hello$
```

Open and Edit the Files

Open the hello.c to make changes and add the following two lines. The lines should _not_ be inside of the init or cleanup functions. To receive credit for this assignment, make sure to use your name (not mine) for the author name.

> **MODULE_AUTHOR("Jonathan Muckell");**
> **MODULE_DESCRIPTION("Homework 1");**
> **MODULE_LICENSE("Proprietary");**

You have two options for editing the file.

> **Option #1**: Edit using the terminal using vi or emacs or nano
> _(** Note: nano is probably the easiest to use)_

> **Option #2**: Edit using a text editor (such as Notepad++). You will have to send the files back and forth from the BBB and your computer using SFTP (Filezilla).

Submission Instructions

Once you have edited the files, recompile the program and load the Linux Kernel Module (LKM) using the insmod command. If everything worked correctly, you should see the following display when you use the modinfo command shown below.

Take a screenshot, similar to the one above showing the results of modinfo. Obviously, it should be displaying your name.

Submit the following items:
- The screenshot above that includes your name (result of modinfo).
- A tar file that includes your hello.c file and the makefile.
  - Use the following command to create the tar file (where hello/ is the directory that contains the code and the Makefile and hw1_*yourNetID*.tar is the name of the tar file. Command: **tar -cvf hw1_ab123123.tar hello/**

Grading Rubric
You will be accessed based on the following criteria:

Deliverable	Points
Submitted .tar file that contains source code and makefile. Tar file is named correctly (includes your netID)	25 Points
Submitted proper screenshot showing your name as the output from the **modinfo** command. Must show complete command and output.	25 Points
TOTAL	50 Points

We must be able to compile your code or it will not receive any credit
***Double check your submission before submitting, as there are no resubmissions*