Assignment 1: Build Environment

Read the entire assignment before beginning!

- Submit deliverables to CourSys: https://courses.cs.sfu.ca/
- Late penalty is 10% per calendar day (each 0 to 24 hour period past due, max 2 days).
- This assignment is to be done **individually**. Do not give your work to another student, do not copy code found online, and do not post questions about the assignment online.
- For any question not directly related to items for marks, please post to the class discussion forum (see website). For questions related to your code, make your post private.

1. Establish Communication

Setup:

- Complete the Quick Start Guide for the BeagleBone Green (on course website).
  - See the website for a list of help-documents/guides for this (and later) sections.
  - The assignment describes what to do; the help-docs describe how to do it.
- Setup a Linux machine (or virtual machine) for doing your work this semester.
- Have the target communicating with the host via the serial port using `screen`.
- Have the target and host on the same Ethernet network (physical or Ethernet over USB).
- Note: If you have a typo in a command when capturing screen output (i.e., a screen capture to a text file), don't worry about "correcting" it. Just repeat the command.
- Commands to be executed in a Linux console on host PC are shown starting with `$`, on the target are shown with a `#`. These are not part of the command; they just represent the terminal prompt.

1.1 Serial (TTL to USB) Connection

Capture the following output from the target's Linux terminal over the serial port.
Hint: In `screen`, you can turn on logging by pressing CTRL+a, and then press H (capital). This will create a log file on the host in the folder you started `screen` from.

- Display target's internet configuration (**`ifconfig`**).
- Ping the host and let in ping at least 3 times; press **`Control C`** to stop it:
  # **`ping 192.168.7.1`**
  (Note: The IP address must be the host's IP; run **`ifconfig`** on the host to find address).
- Rename the log file to `as1-targetViaSerial.txt`

1.2 Ethernet Connection

Perform the following commands on the host in a terminal:

- Display the host's internet configuration.
- Ping the target and let in ping at least three times.
- SSH to the target (change the IP to match the target):
  `$ ssh root@192.168.7.2`
  - This will log you in as `root`, no password is required.
  - Perform a directory listing of the `/proc` directory.
  - Display the kernel's version, up-time, and CPU info:
    `# cat version`
    `# cat uptime`
    `# cat cpuinfo`

- Exit the SSH session using the **`exit`** command.

- Copy the text of your session to a new file named `as1-hostViaIP.txt`
  You should be able to select the text in the terminal and copy-and-paste it into a text file (try running `gedit` on the host to create the .txt file)

2. NFS and Custom Login Messages

2.1 NFS Mount

- Mount your host's `~\cmpt433\public\` folder on the target using NFS. See the website for NFS guide.
- Create a script in your target's `/root/` home directory **named `mountNFS`** which mounts your NFS.
- Hints:
  - Use echo to create the file, such as:
    ```
    # echo Da Command To Mount Folder Goes Here > daFileToSaveInto
    ```
  - Change the permissions on script to be executable:
    ```
    # chmod +x daFileToRun
    ```

2.2 Root File System Customization

Make the following changes to the target's root file system stored in eMMC (on board):

- Overwrite the `/etc/hostname` file with: `yourSFUEmailAddress-beagle`
  Example: `bfraser-beagle`
  - Tip: View what the `/etc/hostname` file currently contains first, and make a backup copy (just in case!)
    ```
    # cd /etc
    # ls hostname
    # cat hostname
    # cp hostname hostname.bak
    ```
  - Tip: Use `echo` (as used in previous step) to change the contents of the `hostname` file.
- Change the hosts file:
  ```
  # nano /etc/hosts
  ```
  - Change all mention of `beaglebone` to `yourSFUEmailAddress-beagle`
  - This will change the board's Linux terminal prompt once you reboot the board. Reboot using the command:
    ```
    # reboot
    ```

2.3 ASCII Greeting

Change the root file system stored on the target to display an ASCII welcome message on boot.

- There are three files that display messages at start-up:
  - `/etc/issue`: Shown only on the serial port before the user logs in.
  - `/etc/motd`: Shown only on the serial port after the user logs in.
  - `/etc/issue.net`: Shown only on SSH connection before the user logs in. Note: Does not support any escape sequences (as described below).
- First, create a short ASCII message:
  - Use a website such as: `http://patorjk.com/software/taag/`
  - Make the message include at least your first name. You may add anything else you like.
  - Pick a "font" which is readable and would fit on the terminal screen.

- For the above website, click the "Test All" link on the left and pick a good one.

- Copy the art into a new text file named `issue.net` in your host's shared folder: `~/cmpt433/public`
  - Note: changing `/etc/issue.net` is optional for this assignment, but easy to do!
- Copy your new `issue.net` file to a new file name just `issue`
- Edit `issue` and add at least one escape code to the file to display some interesting information.
  - List of escape codes can be found here: http://www.cyberciti.biz/faq/howto-change-login-message/
  - In `/etc/issue` you'll need to replace all the '\' characters in your ASCII art with "\\" because '\' is the start of an escape sequence. Note that SSH uses `issue.net` and does not support escape characters, so don't change '\' to '\\' in that file, only in `issue`.
- Using NFS, copy `issue` and `issue.net` from the host to your target's `/etc/` folder.
  - You may want to first make a backup copy of the contents of `/etc/issue` and `/etc/issue.net` on the target.
- Log out, using the `exit` command, to see your new message.
  - If logging in via the serial port, you'll see `/etc/issue` along with your extra escape characters.
  - If logging in via SSH, you'll see `/etc/issue.net` which does not support escape characters. You'll need to re-SSH into the board to see this message.
  - You may delete or rename the `/etc/motd` file to cleanup the boot messages on the serial port if you like.

2.4 This Section's Deliverables

The next section guides you through capturing the boot-up text, which will show these changes.

3. Hello world = Der-Finger-Poken & Der-Lighten-Blinken

("Moving the joystick, turning on lights")

3.1 Hello World

- On the host, create a new directory for assignment 1:
  ```
  $ cd ~
  $ mkdir cmpt433/work/as1
  ```
  - Note: ~ expands to the user's home directory, `/home/brian` in my case.
- On the host, inside ~/`cmpt433/`**`public`**`/`, create a folder `myApps/`
  - This will hold your compiled programs where they can easily be run on the target.
- Create a `hello` program in `cmpt433/`**`work`**`/as1/` (in C not C++!) using `printf()` to display a message of the form:
  "`Hello embedded world, from Brian Fraser!`" (Substitute in your name!)
- Create a makefile that compiles your application and places the compiled file to the `cmpt433/`**`public`**`/myApps/` folder. Your makefile should be such that you can compile and deploy (place executable to the `myApps/` folder) using the following single command:
  ```
  $ make
  ```

Simple Makefile (you may modify as desired):

```
all:
	arm-linux-gnueabihf-gcc -Wall -g -std=c99 -D _POSIX_C_SOURCE=200809L -Werror hello.c -o hello
	cp hello $(HOME)/cmpt433/public/myApps/
```

3.2 Use LEDs & Read Joystick

Modify your `hello` program from above to also play the following LED and Joystick game.

Display hello-world welcome message

Continuously loop through the following:

1. Display user's current score
2. Program selects a 'target' answer of either `up` or `down`.
   If `up`, turn on BBG's LED 0 (top one);
   If `down`, turn on BBG's LED 3 (bottom one)
3. Wait until user moves joystick:
   If left or right, then turn off BBG's LEDs, display goodbye message, and exit app.
   If up or down, then:
   - display correct/incorrect message, update score, and
   - if correct flash all LEDs on once for 0.1s
   - if incorrect flash all LEDs on five times for 0.1s on 0.1s off
4. Wait until joystick is released (so you don't double trigger on one push).

- The LEDs you are turning on are on the BeagleBone, not on the Zen cape.
- Your program must run without error even if some or all of the GPIO pins for the LEDs or the joystick are already exported for GPIO use.
- You need *not* un-export any GPIO pins your program exports for the assignment.

Requirements:

- Use a Makefile to compile your program on the host, and copy the executable to your NFS shared directory (`~/cmpt433/public`). Run the program via the terminal on the target.
- Follow the LED and GPIO guides before writing the program.
- Sample console output is shown below. Note that it does not show which LED is on, nor what the user input was. First and last line in capture are the terminal.

```
root@beaglebone:/mnt/remote/myApps# ./hello
Hello embedded world, from Brian!

Press the Zen cape's Joystick in the direction of the LED.
  UP for LED 0 (top)
  DOWN for LED 3 (bottom)
  LEFT/RIGHT for exit app.
Press joystick; current score (0 / 0)
Correct!
Press joystick; current score (1 / 1)
Correct!
Press joystick; current score (2 / 2)
Correct!
Press joystick; current score (3 / 3)
Correct!
Press joystick; current score (4 / 4)
Correct!
Press joystick; current score (5 / 5)
Incorrect! :(
Press joystick; current score (5 / 6)
Correct!
Press joystick; current score (6 / 7)
Incorrect! :(
Press joystick; current score (6 / 8)
Your final score was (6 / 8)
Thank you for playing!
root@beaglebone:/mnt/remote/myApps#
```

- Hint: Good function design will save you a lot of time!
  - Create some initialization functions to initialize the joystick and LEDs.
  - Create some functions to give you high-level control of joystick and LEDs.
  - For joystick, I suggest you create an enum to represent the four directions, plus a 'no direction'. Then create a function which returns which direction the joystick is pressed. Note this won't handle multiple directions at once, but that's OK for this app.
- Keep your file organized!
  - You are welcome to use multiple files (.c and .h files) if you like, but you need not.
  - If you use one .c file, make sure you organize your functions inside the file well.
    - You may want to use function prototypes so you can organize your functions in the way you think makes the most sense.

3.3 Run it via eMMC

- On the target, copy your blinking `hello` program to the `root` user's home directory (`/root/`). This will make it available on the target even when you have not mounted the public folder via NFS.
- On the target, edit the root user's `.profile` file using the `nano` text editor. Add a call your `hello` program when the user logs in.
  - The `.profile` file is in the user's home directory (`/root/`), and is a script that runs whenever the user logs in.
  - Launch `nano` to edit a file (such as `.profile`) with:
    ```
    # nano .profile
    ```
  - On a new line, enter the full path of the program (such as `/root/myFileHere`).
    - Hint: Have the call to `hello` be the last line in `.profile`. This will let the user skip the blinking at log-in (`Ctrl-c`) and still have the rest of the script execute.
  - To exit `nano` (and save), press `Ctrl+x`, then type `y` (when asked to save the file), then press ENTER to select the existing file name.
  - You can check that the file was edited correctly by using `cat` to display the file.
  - Note that file names starting with a period, such as `.profile`, are considered hidden by Linux and will not show up in a file listing. Use the `-a` option for `ls` to show all files:
    ```
    # ls -a
    ```
- Test your script by logging out and logging back in. Log out with:
  ```
  # exit
  ```

3.4 This Section's Deliverable

- Create a `tar.gz` file named `as1-helloWorld.tar.gz` of your `as1/` directory. Include at least your `hello` C source code and its `Makefile`.
  - In general, to create a `tar.gz` archive, use:
    ```
    $ tar cvzf targetFileName.tar.gz sourceFolderName
    ```
- Use the serial port (`screen`) to capture the output of the following actions. Save the output into a file name `as1-bootTrace.txt`:
    - Hint: Toggle logging within `screen` on or off by pressing `Ctrl+a`, and then `H` (capital, without Ctrl). This will log output to your home directory in a file `screenlog.0`. After you capture all the output rename the log file to the required .txt file name.
    - Note that if you press the arrow keys during your session it may insert escape characters into your capture which you should edit out before submitting. It may be best to not use the arrow keys if you can avoid it.
  - Capture output of the following actions:
    1. Reboot the board, and capture all the text shown on the serial port (text will be from uBoot, the Linux kernel, and the programs loaded from the root file system).
    2. Login as `root`, which will execute your `hello` light-blinking program.
    3. Play a couple rounds of the game and then exit the program using the joystick.
    4. Mount your NFS shared directory using your script (previous section).
    5. Use `ls` to display the contents of the shared directory mounted on the target which contains your program (likely `/mnt/remote/myApps`).

4. Deliverables

Submit the items listed below to the CourSys: https://courses.cs.sfu.ca/

1. `as1-targetViaSerial.txt`
2. `as1-hostViaIP.txt`
3. `as1-helloWorld.tar.gz`
4. `as1-bootTrace.txt`

Please remember that all submissions will be compared for unexplainable similarities. If you have taken the course before, you may not use your work from previous semesters as any part of your solution. Everyone's submissions will be quite similar, given the nature of this assignment, but please make sure you do your own work; we will still be checking.

Revision History:

- None yet!