Assignment 1: Build Environment

Read the entire assignment before beginning!

- 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 other than the course forum.
- If you have previously taken the course, you may not re-use your previous solution.
- Post questions to the course 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 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 `(host)$`, on the target are shown with a `(bbg)$`. These are not part of the command; they just represent the terminal prompt.

1.1 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
  - Perform a directory listing of the `/proc` directory.
  - Display the kernel's version, up-time, and CPU info:

```
(bbg)$ cat version
(bbg)$ cat uptime
(bbg)$ 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

This section has no deliverables on its own; the next section guides you through capturing the required output.

2.1 NFS Mount

- Mount your host's `~\cmpt433\public\` folder on the target using NFS[1]. See the website for NFS guide.

1 Yes, please use the “CMPT433” paths because all the guides mention this. We will expect this directory structure when marking.

- Create a script in your target's home directory **named `mountNFS`** which mounts your NFS.
- Hints:
  - Use echo to create the script file, such as:
    ```
    (bbg)$ echo Da Command To Mount Goes Here > daFileToSaveInto
    ```
  - Change the permissions on script to be executable (change file name!):
    ```
    (bbg)$ 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: mbadali`-beagle`
  - Tip: View what the `/etc/hostname` file currently contains first, and make a backup copy (just in case!)
    ```
    (bbg)$ cd /etc
    (bbg)$ ls hostname
    (bbg)$ cat hostname
    (bbg)$ cp hostname hostname.bak
    ```
  - Hint: `hostname` is owned by the root user, so you need to use `sudo` to edit it:
    ```
    (bbg)$ sudo nano hostname
    ```
    or you could use the following:
    ```
    (bbg)$ echo yourMessage | sudo tee hostname
    ```
- Change the hosts file:
  ```
  (bbg)$ 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:
    ```
    (bbg)$ sudo reboot
    ```

2.3 ASCII Greeting

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

- These files display messages at start-up:
  - `/etc/motd`: Shown on after the user logs in (via serial or ssh).
- 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 `motd` in your host's shared folder:
  `~/cmpt433/public`
- Using NFS, copy `motd` from the host to your target's `/etc/` folder.
- Log out, using the `exit` command, to see your new message.

Hint: You'll be seeing this message a lot because it displays anytime you successfully log into your board! Make it something you want to see, like "Welcome!" or "Have Fun!".

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

("Pressing buttons; turning on lights")

3.1 Hello World

- On the host, create a new directory for assignment 1's source code:
  `~/cmpt433/work/as1`
  - Note: ~ expands to the user's home directory, `/home/morteza` in my case.
  - You may actually put your code anywhere, such as in a Git repo.
- On the host, create a new folder in public for sharing your compiled applications to share them with the target via NFS:
  `~/cmpt433/public/myApps/`
- 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 <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:
  `(host)$ 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 Buttons

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

Display a hello-world welcome message

Continuously loop through the following steps to play the game:

1. Wait while user holds down USER button
2. Light up only LED 0
3. Wait a random time (between 0.5s and 3.0s)
4. If user is pressing the USER button already (too soon):
   - Record response time as 5.0s
   - Skip to “Light up all LEDs”
5. Light up LED 3 & start timer
6. When user presses USER button, stop timer
   - If timer > 5s, exit with a message without waiting for button press
7. Light up all LEDs
8. Display summary:
   - How many ms was the current response time?
   - How many ms is the best response time so far this game?

Example Console Output

```
debian@BeagleBone:/mnt/remote/myApps$ ./hello
Hello embedded world, from Morteza!

When LED3 lights up, press the USER button!
New best time!
Your recation time was  123ms; best so far in game is  123ms.
Your recation time was  944ms; best so far in game is  123ms.
Your recation time was  196ms; best so far in game is  123ms.
Your recation time was  392ms; best so far in game is  123ms.
New best time!
Your recation time was    3ms; best so far in game is    3ms.
Your recation time was 5000ms; best so far in game is    3ms.
Your recation time was 1181ms; best so far in game is    3ms.
No input within 5000ms; quitting!
debian@BeagleBone:/mnt/remote/myApps$
```

Requirements

- **LEDs**
  - The LEDs you are controlling are on the BeagleBone Green LEDs (not external LEDs). You may assume that these LEDs are present (the `/sys/class/leds/beaglebone:green:usr#` folders exist).
  - At startup, your C program must set each LED’s triggers
  - When your program finishes, leave the LED triggers as “none” and turn off all LEDs.

- Follow the LED guide before writing the program.

- **Button**
  - The button your C program is reading is the USER (also called “BOOT”) button on the BBG.
  - You may assume that its GPIO pin has been exported (`/sys/class/gpio/gpio##/` folder exists)
  - Your C program must execute `config-pin` to force the button’s pin to be treated as GPIO (see sample code below).
  - Your C program must configure the button’s pin to be input.
  - Follow the GPIO guide before writing the program.
- Use a Makefile to: cross-compile your program on the host, and copy the executable to your NFS shared directory (`~/cmpt433/public/myApps`). Run the program via the terminal on the target.
- Sample console output is shown below (your output may differ). 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.
- Your code must be clean and maintainable.
  - Use functions to organize your code
  - All functions and variables must have good intention revealing names
  - Use comments effectively (OK to have no comments if your code reads like a book)
  - For this assignment, you may put all your code in one file, or you can choose to use multiple `.h` and/or `.c` files.

Hints

- *Hint: For timers*
  - To start a timer, record the current time as the start-time. Then compare the current time to the start-time as your program runs.
  - Work with time in milliseconds.
  - Time often means elapsed time since ~1970, so with milliseconds we should use the `long long` data type to ensure we don’t overflow a 32-bit int.
  - To print a `long long`, use:
    ```
    long long x = 123456789LL;
    printf(“Big number is %lld right!”, x);
    ```
  - You can get the current time with:

```
static long long getTimeInMs(void)
{
    struct timespec spec;
    clock_gettime(CLOCK_REALTIME, &spec);
    long long seconds = spec.tv_sec;
    long long nanoSeconds = spec.tv_nsec;
    long long milliSeconds = seconds * 1000
                + nanoSeconds / 1000000;
    return milliSeconds;
}
```

- You can wait a number of milliseconds with:

```
static void sleepForMs(long long delayInMs)
{
    const long long NS_PER_MS = 1000 * 1000;
    const long long NS_PER_SECOND = 1000000000;

    long long delayNs = delayInMs * NS_PER_MS;
    int seconds = delayNs / NS_PER_SECOND;
    int nanoseconds = delayNs % NS_PER_SECOND;

    struct timespec reqDelay = {seconds, nanoseconds};
    nanosleep(&reqDelay, (struct timespec *) NULL);
}
```

- *Hint: To run a Linux command from in your C program, use:*

```
static void runCommand(char* command)
{
    // Execute the shell command (output into pipe)
    FILE *pipe = popen(command, "r");

    // Ignore output of the command; but consume it
    // so we don't get an error when closing the pipe.
    char buffer[1024];
    while (!feof(pipe) && !ferror(pipe)) {
        if (fgets(buffer, sizeof(buffer), pipe) == NULL)
            break;
        // printf("--> %s", buffer);  // Uncomment for
debugging
    }

    // Get the exit code from the pipe; non-zero is an error:
    int exitCode = WEXITSTATUS(pclose(pipe));
    if (exitCode != 0) {
        perror("Unable to execute command:");
        printf("  command:   %s\n", command);
        printf("  exit code: %d\n", exitCode);
    }
}
```

- *Hint: Good function design will save you a lot of time throughout this course!*
  - Create some initialization functions to initialize the button and LEDs.
  - Create some functions to give you high-level control of button and LEDs.

3.3 Run it via eMMC

- On the target, copy your blinking `hello` program to `debian` user's home directory (`/home/debian/`). This will make it available on the target even when you have not mounted the public folder via NFS.
- On the target, edit the `debian` user's `.profile` file. Add a call your `hello` program when the user logs in.
  - The `.profile` file is in the user's home directory (`/home/debian/`); it is a script that runs whenever the user logs in.
  - Edit `.profile` with:
    ```
    (bbg)$ nano .profile
    ```
  - On a new line, enter the full path of the program (such as `/home/name/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.
  - Use `cat` to check the file was edited correctly.
  - 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:
    ```
    (bbg)$ ls -a
    ```
- Test your script by logging out and logging back in. Log out with:
  ```
  (bbg)$ exit
  ```

3.4 This Section's Deliverable

- Create an archive of your code, as described in the Deliverables section of the assignment.
- Using SSH, capture the output of the following actions. Save the output into a file name `as1-bootTrace.txt`:
  - Capture output of the following actions:
    1. Login as `debian`, which will execute your `hello` light-blinking program.
    2. Play a couple rounds of the game and then exit the program by letting it timeout (no user input).
    3. Mount your NFS shared directory using your script (previous section).
    4. 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-hostViaIP.txt`
2. `as1-helloWorld.tar.gz`
   Compressed copy of source code and build script (`Makefile`).

   Archive must expand into the following (without additional nested folders to find the `Makefile`)

   ```
   <assignment directory name>
    |-- Makefile
    \-- <dependencies such as .c, .h files>
   ```

   `Makefile` must support both the `make` and `make all` commands to build your program to `$(HOME)/cmpt433/public/myApps/`
   (Do not use relative paths for getting to the `cmpt433/public/myApps/` directory because the TA may build from a different directory than you.)

   Hint: Compress the `as1/` directory with the command like:
   `(host)$ tar cvzf as1-helloWorld.tar.gz as1`
3. `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. You are likely to find other student's previous submissions online. Copying from another source is academic dishonesty. Everyone's submissions will be quite similar, given the nature of this assignment, but please make sure you do your own work; we will be checking.

Revision History:

- V1: initial post