Lab #1 -- EC535, Spring 2019

Assigned: January 31, Thursday --- **Due:** February 7, 10:00pm Thursday

Lab Preparation:
*Go to: Blackboard Learn / EC535 -> Content -> Lab Preparation and Reading Questions -> **Lab1Prep.***

The labs are based on the Gumstix development boards. Technical specifications of the boards can be found at http://gumstix.com. The board can run either Windows CE or Linux operating systems. In this course, we will use Linux exclusively.

During the first lab you will be using a processor emulator to learn how to build a boot loader, configure and build a Linux kernel and a root file system. You will also configure and build a *Busybox* and learn the basics of Linux boot sequence and how to use *cron*, the Unix scheduler.

Before you start the lab, you need to understand the following concepts:
Boot loader — http://en.wikipedia.org/wiki/Boot_loader
Kernel image — http://en.wikipedia.org/wiki/Kernel_image
JFFS2 filesystem — http://en.wikipedia.org/wiki/JFFS2
QEMU processor emulator — http://en.wikipedia.org/wiki/QEMU

Step-by-step instructions

0. Setting the environment

All of the lab tools are located in the **/ad/eng/courses/ec/ec535/** directory. Run the following line on a terminal window.
Optional: To conveniently access the tools, you can edit your .bash_profile file to include the following line at the end.

```
source /ad/eng/courses/ec/ec535/bashrc_ec535
```

Check if the environment variable $EC535 is set up correctly by typing:
`echo $EC535`
This command should show the path to the ec535 directory.

Create a folder. Suppose your directory is foo **(use a recognizable name for the folder, not foo)**. Then:

```
mkdir foo
```

Important note:
You may also work under the /tmp folder (i.e., `mkdir /tmp/foo`), but note that /tmp folder is not backed up and it may not be accessible remotely via eng-grid.
Copy any files or folders you would like to save to your home directory: /home/yourBUlogin. The instructor, the TAs, or Eng-IT are *NOT* responsible for lost files.

1. Instructions to build jffs2 rootfs

This step converts a regular directory tree into a jffs2 image. (You can also do this under /tmp.)

" > " denotes a single command line. Be careful about putting in the entire line before hitting enter. Make sure to omit the character > while copying the lines below.

```
> cd /home/yourBUlogin/foo
> rm -rf root
> tar xjf $EC535/gumstix/root.tar.bz2
> cp  $EC535/gumstix/device_table.txt .
> mkfs.jffs2 -l -U -e 128KiB -d ./root -D device_table.txt sumtool -e 128KiB -o
rootfs_gumstix.jffs2
```

2. Instructions to run qemu simulation.

Without a real board you can develop software using a simulator like **qemu** we are using in this lab. Run the simulation in your local directory under /tmp. In order to run the simulation:

1. First, prepare a flash image file

   ```
   > cd /home/yourBUlogin/foo
   > cp $EC535/gumstix/uImage .
   > cp $EC535/gumstix/u-boot.bin .
   > rm flash
   > dd of=flash bs=1k count=16k if=/dev/zero
   > dd of=flash bs=1k conv=notrunc if=u-boot.bin
   > dd of=flash bs=1k conv=notrunc seek=256 if=rootfs_gumstix.jffs2
   > dd of=flash bs=1k conv=notrunc seek=31744 if=uImage
   ```

   Note this flash file is created only for simulation. You won't need it when experimenting with the real gumstix board.
   Try to understand what uImage (kernel image or micro image) is; you can search online as needed.

2. Second, run qemu with the following arguments

   ```
   > qemu-system-arm -M verdex -pflash flash -monitor null -nographic -m 289
   ```

   Ignore the screenful of errors regarding unimplemented flash cmd. They are due to some limitations of the flash memory model in the qemu simulator. But they will disappear the next time you run qemu on the same flash image file.

   At the login prompt, log in as root. Password is gumstix.

3. Explore the directory structure of the simulated Linux environment. Play with the Linux commands in the simulated Linux environment (ls, cd, cd .., etc.).
   To exit qemu, first power off the processor under simulation by typing:

   ```
   > poweroff
   ```

   Then, after seeing "System halted", type CTRL-a and then x.

3. Instructions to build busybox

In this step, you will learn to build a busybox package, which provides most of the Linux commands that you will need in a root file system. Again do this under the same folder you've been using:
(If you have been using the /tmp folder, you need to type `cd /tmp/foo` below.)

```
> cd /home/yourBUlogin/foo
> rm -rf busybox-1.1.2
> tar xzf $EC535/gumstix/busybox-1.1.2.gumstix.tgz
> cd busybox-1.1.2
```

1. First, you need to configure the busybox package to specify some building options, and optionally, select the commands that you want to be included.

   ```
   > make menuconfig
   ```

   Enter Busybox Settings: Build Options
   Select by pressing space key: Do you want to build Busybox with a Crosscompiler?
   Press down and enter, replace the original prefix with only "arm-linux-"
   Exit build options.

   Browse through and select other options.
   Exit and save the configuration.

2. Then, build busybox.

   ```
   > make
   ```

   The created busybox will be under the current directory.

3. To install, you can type make

   ```
   > make install
   ```

   The busybox binary will be installed under the ./_install folder. It will be organized in a similar way to a rootfs. You can copy the content of _install to your own root directory when you build your jffs2 image (in step 5). When you copy, make sure that you use "**cp -a**" so that the symbolic links are preserved.

4. Instructions to build Linux kernel

Again, do this step under your working folder:

```
> cd /home/yourBUlogin/foo
> rm -rf linux-2.6.21gum
> tar xjf $EC535/gumstix/linux-2.6.21gum.tar.bz2
```

1. First, load the default configuration file

```
> cd linux-2.6.21gum
> make ARCH=arm CROSS_COMPILE=arm-linux- gumstix_defconfig
```

This will copy gumstix configuration settings into the .config file, the default configuration file for Linux.

2. Second, in case you want to do some customization, do the following

```
> make ARCH=arm CROSS_COMPILE=arm-linux- menuconfig
```

Browse through the menus and enable/disable modules. You will encounter three types of settings, *, M or empty. * means the feature is selected as part of the kernel. M means the feature is selected, but will be built into a module, not part of the kernel. Empty means you are skipping the feature.

You need to set the following:
a. System Type -> Gumstix Platform Version -> Gumstix Verdex
b. Kernel features -> Preemptible Kernel
c. Kernel features -> Use the ARM EABI to compile the kernel
Allow old ABI binaries to run with this kernel.

3. Third, build the kernel. This may take several minutes.

```
> make ARCH=arm CROSS_COMPILE=arm-linux- uImage modules
```

The uImage file under ./arch/arm/boot/uImage is the resulting kernel image. You can then use the kernel image to build your flash image and run qemu.

```
> make ARCH=arm CROSS_COMPILE=arm-linux- \
INSTALL_MOD_PATH=/home/yourBUlogin/foo/built_modules modules_install
```

Note the above two lines are one command. This will create folder under `foo/built_modules` which contains all modules that are selected. This folder will be useful for building rootfs.

5. Instructions to create rootfs

The provided rootfs is a good example to follow. But you need to learn to customize your own root file system. Again, do this under /tmp.

```
> cd /home/yourBUlogin/foo
> rm -rf rootfs
```

1. Prepare an empty rootfs folder

```
> mkdir rootfs
```

2. Build Busybox, copy content in its installed directory to your rootfs:

```
> cp -a /home/yourBUlogin/foo/busybox-1.1.2/_install/* rootfs/
```

If the busybox binary is dynamically linked, you need to install shared libraries to the lib folder. The minimum library files to install include the following:

ld-uClibc-0.9.29.so	libc.so.0	libdl.so	libgcc_s.so	libuClibc-0.9.29.so
ld-uClibc.so.0	libc.so	libdl-0.9.29.so	libdl.so.0	libgcc_s.so.1
libcrypt.so	libcrypt.so.0			

Some of those are real library files. Some are just symbolic links. Enter the library directory first:

```
> cd rootfs/lib
```

(If the directory name is lib64, please do:

```
mv lib64 lib
```

to rename the directory)

Copy the necessary files from $EC535/arm-linux/lib directory. For example,

```
> cp -a $EC535/arm-linux/lib/ld-uClibc-0.9.29.so .
```

Copy the kernel modules (if there are any) from your foo/built_modules/lib.

```
> cp -a /home/yourBUlogin/foo/built_modules/lib/modules .
```

3. You will also need to create a dev folder, an etc folder, and populate the etc folder with inittab and necessary startup scripts. You can use the content in the provided rootfs (./root) as an example. But you definitely won't need all the files and folders in the provided rootfs.

   To understand how some Busybox commands work, including the format of the inittab file, check out http://www.busybox.net/downloads/BusyBox.html

4. Build a jffs2 image file with your rootfs

```
> cd /home/yourBUlogin/foo
> mkfs.jffs2 -l -U -e 128KiB -d ./rootfs -D $EC535/gumstix/device_table.txt
sumtool -e 128KiB -o rootfs_gumstix.jffs2
```

5. Build a flash image file, and simulate using qemu.

   If you encounter an init failure problem because a shared library is missing, go back to step 2 and copy the library to the lib folder. Do 4 and 5 again.

   If you encounter a **kernel panic problem**, it is likely that kernel cannot run your init program. This can happen when init is not present, or when some key shared library is missing, such as libdl.so. Other libraries that you may need: libcrypt.so and libcrypt.so.0.

****

Tasks to Complete

1. Your Linux should print the banner found at $EC535/gumstix/banner.ascii when booting up. You should see the penguin picture while booting when you complete this task successfully. Note that the banner should appear before the login prompt.

2. When booting up, automatically start a cron job, which prints the following message to the screen once every 5 seconds:
   `Welcome to EC535 Spring 2019! Current date and time: <Date and Time>`

   Hint: Minimum cron resolution may be larger than 5 seconds. Cron could, however, call a script that performs tasks (such as printing) more often.
   You can obtain the date and time by the Linux command "date". You can use a simple shell script to print the above message. For an introduction to cron, check out:
   http://www.unixgeeks.org/security/newbie/unix/cron-1.html and/or
   http://en.wikipedia.org/wiki/Crontab.
   Note that the busybox version of cron is slightly different from the standard Unix cron in specifying tasks.

   Cron may print some extra log outputs on the screen. If you clean these up, then you get 5 extra credit points.

3. Write a C program that prints all the letters in the English alphabet in order (ABC…YZ), cross-compile it, and place the binary (name it **alphabet**) under the /home folder under your root file system. You should be able to run the program during emulation. Optimize and strip your binary to make it small
   (Hints: try "man strip" and try checking out compiler optimizations in man page of gcc).

   Make sure you are using the arm-linux-gcc cross compiler under `$EC535/arm-linux/bin`. We will use another version of the cross compiler when we use the actual hardware.

4. Build a minimum root file system and a minimum kernel image. You receive 80% of the grade if you accomplish task 1, 2, and 3. The rest 20% will depend on the total size of the root file system and kernel image (the smaller the better). You should maintain the basic kernel functionalities while minimizing the file system and kernel image (traversing folders, listing directory contents, creating files, etc.). You can test your system with the following linux commands:
   `ps, pwd, ls, cd, cat, more, less, echo, cp, rm, mv, mkdir, rmdir`

   In other words, it is fine to remove other modules or utilities, as long as you can verify functionality of your system with the above commands. You should be able to run your simple C code, boot up displaying the penguin, and run the cron job as well.

   You can work on task 1-3 first using the provided kernel images/file systems. After you complete 4, make sure that you still include items 1, 2, 3 in your minimum file system.

Example competitive grading rubric (this will be adjusted based on submissions from this year's class):

uImage size:

10 points: size <= 500K; 8 pts: 500K < size <= 550K; 6 pts: 550K < size <= 700K; 4 pts: 700K < size <= 900K; 0 pts: 900K < size.

rootfs.tar.gz size:

10 points: size <= 450K; 8 pts: 450K < size <= 600K; 6 pts: 600K < size <= 750K; 4 pts: 750K < size <= 2000K; 0 pts: 2000K < size.

Submission

Items to submit:
(1) Your root file system (as a single tar.gz – learn how to use tar and gzip).
(2) Your Kernel image (uImage).
(3) Your kernel configuration file (.config).

Please put the above in a folder named yourBUlogin_lab1. Do not include any other files.

Submit a single archive (yourBUlogin_lab1.tar) via email to ec535spring2019@gmail.com. Your subject line should be: yourBUlogin_lab1.

*Not following submission guidelines **will** result in deduction of points from your lab grade.