ECGR 4101/5101 - Lab 3

Objective: Creating an embedded Linux distribution

Outcomes:
After this lab, you will be able to

- Build Linux kernel image for ARM Versatile board
- Build a root file system with Busybox
- Use Buildroot to build an embedded Linux file system, and test application

Build Kernel Image
Download the latest stable version of the Linux kernel (linux-5.8.14) from www.kernel.org. In linux-x.x.x folder, do the following

$ make ARCH=arm versatile_defconfig
If you get errors, "*flex: not found*" and "*bison: not found*", do the following installations,
$ sudo apt-get install flex
$ sudo apt-get install bison
Then redo the step, make ARCH=arm versatile_defconfig
$ make ARCH=arm menuconfig (Enable EABI support. Have a look at other kernel build features)
If you get error, "*Unable to find the ncurses package*", do the following installation,
$ sudo apt-get install libncurses-dev
Then redo the step, make ARCH=arm menuconfig and enable EABI support.
$ make ARCH=arm CROSS_COMPILE=arm-none-eabi- all -j 2 (Parallel compile)

This will start the building of the kernel using the ARM cross compiler; the build will create, among other binaries, a compressed kernel in a file called *zImage* located in *"arch/arm/boot"*

Install Linux cross compiler toolchain (Ubuntu/Linaro)
$ sudo apt-get install gcc-arm-linux-gnueabi

Build root file system with Busybox
Download Busybox source from http://www.busybox.net. Use version 1.25.1. I had issues with the latest stable version 1.30.0

BusyBox combines tiny versions of many common UNIX utilities into a single small executable. BusyBox has been written with size-optimization and limited resources in mind and is easily customizable for embedded systems.

```
$ tar xjf busybox-1.25.1-tar.bz2
$ cd busybox-1.25.1
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- defconfig
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- menuconfig
```

Check the option to build Busybox as a static executable.
The setting can be found in \Busybox Settings --> Build Options

```
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- install
$ cd _install
$ mkdir proc sys dev etc etc/init.d
```

To mount */proc* and */sys* directories every time on startup, we can use */sbin/init* functionality: this program is usually the first run by the Linux kernel, and its default behavior is to execute the initialization file with path */etc/init.d/rcS*. Create a file *_install/etc/init.d/rcS* with the following content,

```
#!/bin/sh
mount -t proc none /proc
mount -t sysfs none /sys
/sbin/mdev -s (Creates /dev files)
```

Change the rCS file to executable,

```
$ chmod +x _install/etc/init.d/rcS
```

From the _install directory do the following,

```
$ find . | cpio -o --format=newc > ../rootfs.img
$ cd ..
$ gzip -c rootfs.img > rootfs.img.gz
```

The *cpio* tool makes a list of files and outputs an archive; the *newc* format is the format of the *initramfs* file system that the Linux kernel recognizes.
Note: When the Linux kernel boots the system, it must find and run the first user program, generally called *“init”*. User programs live in filesystems, so the Linux kernel must find and mount the first (or *“root”*) filesystem in order to boot successfully. Ordinarily, available filesystems are listed in the file */etc/fstab* so the mount program can find them. But */etc/fstab* is itself a file, stored in a filesystem. Finding the very first

filesystem is a chicken and egg problem, and to solve it the kernel developers bundle a small ram-based initial root filesystem into the kernel, and if this filesystem contains a program called *"/init"* the kernel runs that as its first program. After this step a new root filesystem can be mounted from a different device. The previous root (from *initrd*) is then either moved to the directory */initrd* or it is unmounted.
QEMU passes the filesystem binary image to the kernel using the initrd parameter. Launch QEMU. Here's what I did -

From Lab3 directory,

```
$ ../Lab1/qemu-5.1.0/arm-softmmu/qemu-system-arm -M versatilepb -m 128M -kernel linux-5.8.14/arch/arm/boot/zImage -initrd busybox-1.25.1/rootfs.img -append "root=/dev/ram rdinit=/bin/sh" -dtb linux-5.8.14/arch/arm/boot/dts/versatile-pb.dtb -nographic
```

You should see *"#"* prompt. The shell can be used normally, for example you can run *ls* command to find the same directory structure as the Busybox install directory.

Do the following:

1. Follow the directions provided in this link to build a kernel, filesystem, and application with buildroot. The author uses ARM Vexpress board, with the root file system mounted via NFS.
   https://devarea.com/building-embedded-linux-system-with-qemu/