Lab 4
Kernel Patching and Cross-Compilation for RPi Embedded Linux

Julia Desmazes
Michael Nissen

October 25, 2017

1 Download / Set the good version of Linux sources

To execute the tasks for this report, the first step is to get the Raspberry Pi cross-compilation tools, and specific version of the linux kernal that one wants to patch. This is going to be done using `git clone`, which ofcourse assumes a working installation of `git`.
To get the latest version of the kernel source code and the cross-compilation tools, the following commands with be copied to the terminal:

```
host$ git clone https://github.com/raspberrypi/tools.git
host$ git cone -b rpi-4.4.y https://github.com/raspberrypi/linux.git
```

One now has to make sure to get the same configuration as is used by ones Raspberry Pi during the kernel compilation, which can be done by checking out the `git branch` that corresponds to the hash of the Linux kernel running on the Raspberry Pi.
This will be done by by running the shells script seen in Code Snippet 1. This shell script will grep the firmware hash found on the Raspberry pi, and then get the `git hash` from the github link relative to the hardware hash.

```
#!/bin/bash

PLATFORM=`uname -s`

if [ ${PLATFORM} = "Darwin" ]
then
    CMD="gunzip -c"
elif [ ${PLATFORM} = "Linux" ]
then
    CMD="zcat"
else
    echo "Sorry, the platform ${PLATFORM} is not supported !!!"
    exit -1
fi

FWHASH=`${CMD} /usr/share/doc/raspberrypi-bootloader/changelog.Debian.gz | grep -m 1 'as
    of' | awk '{print $NF}'`
#echo "Firmware Hashcode: fwhash = $FWHASH"

LINUXHASH=`wget -qO- https://raw.github.com/raspberrypi/firmware/$FWHASH/extra/git_hash`

echo "Linux Hashcode: linuxhash = $LINUXHASH"
```

Code Snippet 1: Shell script to get Linux kernel git hash

When one has figured out the `git hash`, one can change directory to the linux git repository and checkout the respective brach by the following command:

```
host$ cd linux
host$ git checkout <git_hash>
host$ make clean -mrproper
```

At line 3, we are cleaning kernel tree from all unneeded files of the first version which is a recommended practice prior to each kernel compilation!

2 Patch the Kernel

Before starting to patch the kernel, one should make sure to identify the Linux sources that one is trying to install. This can be done using the `head` command, which will look like this:

```
host$ head Makefile

  # Ouput generated by running head on the Makefile #
  VERSION = 4
  PATCHLEVEL = 4
  SUBLEVEL = 21
  EXTRAVERSION =
  NAME = Blurry Fish Butt
```

Now that one know the specific version of the Linux kernel, it is time to download the latest `PREMEPT-RT` patch that corresponds to ones kernel version - if follows the format: `version.patch-level.sub-level`. To download the patch from the patch from `kernel.org` using the `wget` command like so:

```
wget https://cd.kernel.org/pub/linux/kernel/projects/rt/4.9/patch-4.9.47-rt47.patch.gz
```

Now one can actually start patching the kernel sources by executing the following commands in succession:

```
host$ gunzip patch-<version.patch-level.sub-level>-rt<last>.patch.gz
host$ cat patch-<version.patch-level.sub-level>-rt<last>.patch | patch -p1
```

One can now create a folder named `rt-modules` under your project folder - at the same level as the `linux` and `tools` folders, and export an environment variable `INSTALL_MOD_PATH` that points to that folder:

```
host$ mkdir rt-modules
host$ cd rt-modules
host$ export INTALL_MOD_PATH=$PWD
```

Here `$PWD` is the path to the current directory.

3 Configure Cross-compilation

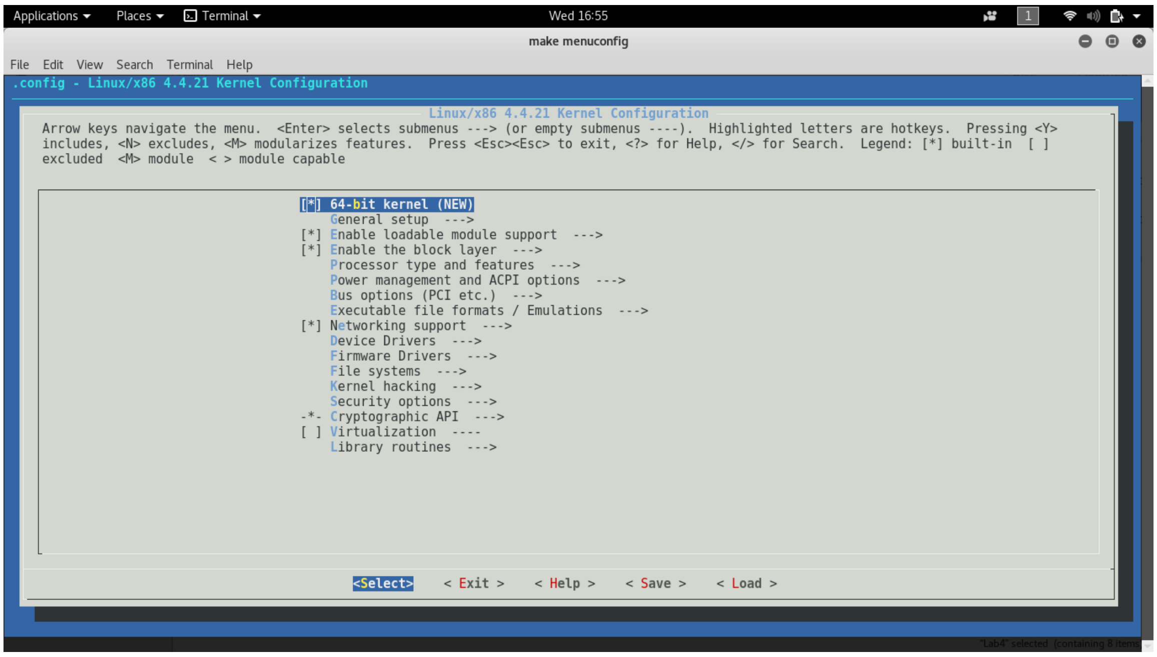
Before starting the cross-compilation, one is going to specify the architecture for which to compile, the cross-compiler that ones wishes to use, and the kernel that one is trying to compile for. This is done by exporting some specific environment variables that is used by the makefile:

```
host$ export ARCH=arm KERNEL=kernel7
host$ export CROSS_COMPILE=$PWD/tools/arm-bcm2708/gcc-linaro-arm-linux-gnueabihf-raspbian
    -x64/arm-linux-gnueaihf-
```

Now, to configure the kernel in an appropriate way, one can use the existing kernel configuration from the Raspberry Pi. This configuration file can be generatied using the `modprobe` command, and then copy that generated file from the Raspberry Py over to the `linux/` folder on the host machine using the `scp` command. Finaly, extract the content of the `config.gz` into a `.config` file:

```
host$ sudo modprobe configs
host$ scp pi@193.169.0.26:/proc/config.gz /home/user/kernel_labs/linux
host$ zcat -c config.gz > .config
```

Now that one has created a default `.config` file, it is time to costumise the kernel using the `sudo make menuconfig` which opens the GUI seen in Figure 1. For the purpose of this lab assignment, we are going to enable the `Fully Preemptible Kernel` setting, and the `High Resolution Timer Support`.



```
.config - Linux/x86 4.4.21 Kernel Configuration
                          Linux/x86 4.4.21 Kernel Configuration
  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).  Highlighted letters are hotkeys.  Pressing <Y>
  includes, <N> excludes, <M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </> for Search.  Legend: [*] built-in  [ ]
  excluded  <M> module  < > module capable

                                  [*] 64-bit kernel (NEW)
                                      General setup  --->
                                  [*] Enable loadable module support  --->
                                  [*] Enable the block layer  --->
                                      Processor type and features  --->
                                      Power management and ACPI options  --->
                                      Bus options (PCI etc.)  --->
                                      Executable file formats / Emulations  --->
                                  [*] Networking support  --->
                                      Device Drivers  --->
                                      Firmware Drivers  --->
                                      File systems  --->
                                      Kernel hacking  --->
                                      Security options  --->
                                  -*- Cryptographic API  --->
                                  [ ] Virtualization  ----
                                      Library routines  --->

                    <Select>    < Exit >    < Help >    < Save >    < Load >
```

Figure 1: The `menuconfig` GUI

4 Build the new Kernel and Modules

To build the kernel,