Lab 4

Kernel patching and cross-compilation for RPi

Prerequest 1: This lab is to be done on a Debian-based OS installed on your host machine and a Raspbian installed properly on your RPi. ■

Prerequest 2: Your RPi should be accessible from the host machine via SSH. ■

Prerequest 3: Commands prefixed by `$host` are to be done in the host machine and those prefixed by `$rpi` are to be done on your RPi. ■

1 Download/Set the good version of Linux sources

✍ **Note 1:** Many devices have two essential software pieces that make them function in Linux: 1) the first is a working driver, which is the software that lets your system talk to the hardware; 2) the second is firmware, which is usually a small piece of hardware code that is uploaded directly to the device for its proper operation. You can think of the firmware as a way of programming the hardware inside the device. In almost all cases, firmware is a black box setup with no freely distributed sources.

Once you install your custom kernel, there is a chance of getting it overwritten when you update firmware through `rpi-update` command which can cause some incompatibilities after cross-compilation. To avoid this, you can disable kernel update by `sudo SKIP_KERNEL=1 rpi-update` in your RPi. ■

The following notes describes how to patch/cross-compile Linux kernel for Raspbian image on Debian-based host machines.

1. Install `git`:
   `host$ sudo apt-get install git-core libncurses5-dev`;
2. Get the last version of kernel source code and the tools needed to cross-compile the image:
   a) Create a working directory `kernel_labs` in your home and enter it;
   b) `host$ git clone https://github.com/raspberrypi/tools.git`;
   c) `host$ git clone -b rpi-4.4.y https://github.com/raspberrypi/linux.git` (this may take some time).

   ✍ **Note 2:** If you want to switch back to an older kernel version like 4.1, you use the `git` command `host$ git checkout rpi-4.1.y`. ■
3. In order to point on the same configuration as that used in your RPi during the kernel compilation, you have to get `git` branch hash value that corresponds to the Linux kernel running on your RPi. For that, you have to do the following commands:
   a) Download the joined `get_hash.sh` form `campus.ece.fr` to your home at the host machine;
   b) Copy using `scp` the file in your RPi;
   c) Under the RPi, give it the rights 755;
   d) Running it under RPi outputs a git branch hash value `<versionhash>`, corresponding to the kernel version running on your RPi.
   e) Under sources `linux/`, use `git checkout` to point on that branch;
   d) Clean the sources;
   ✍ **Note 3:** The option `mrproper` cleans the kernel tree from all unneeded files of the first version (the downloaded one). It is recommended prior to each kernel compilation. ■

2 Patch the kernel

You are supposed to be under the folder linux/ in your host machine.

1. Identify the Linux sources version.patch-level.sub-level (using the command head);
2. Download the last PREMEPT-RT patch corresponding to that version.patch-level.sub-level (using the command wget from kernel.org);
3. Patch the kernel sources:
   a) host$ gunzip patch-<version.patch-level.sub-level>-rt<last>.patch.gz
   b) host$ cat patch-<version.patch-level.sub-level>-rt<last>.patch | patch -p1
4. Create a folder rt-modules under kernel_labs and export to environment a variable INSTALL_MOD_PATH pointing on that newly created folder;

3 Configure cross-compilation

1. Under your host machine, export to environment ARCH and CROSS_COMPILE in order to specify resp. the ARM architecture and cross-compilation tools (previously downloaded) prefix (under 64bit machines, point on the good sub-directory of tools). Export an additional variable KERNEL=kernel7 (kernel if you are using an RPi 1/0/0W);
2. In order to configure the kernel, you may follow two options:

   2.1) Generate the .config file from the pre-packaged RPi template by running
   host$ make bcm2709_defconfig (is not appropriate for the lab context).
   ✍ **Note 4:** In case of RPi 1/0/0W, use bcmrpi_defconfig in the above command. ■

   2.2) In case you want to use your existing kernel configuration, get the .config file from your RPi by using below command [**more appropriate**]:

   a) Generate your configuration using modprobe;
   b) Copy the generated /proc/config.gz in your host machine under linux/ (using scp);
   c) Use the command zcat to extract its content in .config file;
   ✍ **Note 5:** The command zcat uncompresses either a list of files on the command line or its standard input and writes the uncompressed data on standard output. ■
3. You have created the default .config file through one of the two options above. You can now customize your kernel using below optional command by textual menuconfig:
   a) host$ make menuconfig
   b) Set the key CONFIG_PREEMPT_RT_FULL by enabling
   Kernel Features → Preemption Model (Desktop) → Fully Preemptible Kernel (RT);
   c) Be sure that the key CONFIG_HIGH_RES_TIMERS is enabled by checking that
   General setup → Timers subsystem → High Resolution Timer Support
   is enabled.
   ✍ **Note 6: High resolution timer (hrtimer):** ticks are dynamic, (architecture-free and independent from the ISR [clock event source] tick interrupts issued from hardware) and based on nanoseconds (ns). hrtimers are kept in a time sorted, per-CPU list, implementation as a red-black tree ($O(log(n)$ insertion and removal needed to maintain expiration times when the timer wheel complete an entire turn; the data structure is efficient than already used structures and used in other performance critical parts of the kernel e.g. memory management). More information can be found here. ■

4 Build the new kernel and modules

1. Build kernel using following commands (you can boost the compilation time by using the option -j<nb_procs> of `make` where <nb_procs> is twice the number of cores of your computer):

   a) Option `zImage` to compile the kernel;

   b) Option `modules` to compile modules and firmware;

   c) Option `dtbs` to get Device Tree;

   d) Option `modules_install` to install the kernel modules in `$INSTALL_MOD_PATH`;

2. Now create the kernel image:

   a) Create a folder `boot/` under `$INSTALL_MOD_PATH`;

   b) `host$ ./scripts/mkknlimg ./arch/arm/boot/zImage $INSTALL_MOD_PATH/boot/$KERNEL.img`;

   c) `host$ cp ./arch/arm/boot/dts/*.dtb $INSTALL_MOD_PATH/boot/`

   d) `host$ cp -r ./arch/arm/boot/dts/overlays $INSTALL_MOD_PATH/boot`

   e) Under `$INSTALL_MOD_PATH`, create a compressed file `kernel.tgz` from the content of the folder `$INSTALL_MOD_PATH` and copy it under `/tmp` in your RPi.

3. It remains to copy kernel image, the Device Tree, the firmware and modules in the proper locations under your RPi:

   a) Under `/tmp`, decompress the exported file `kernel.tgz`;

   b) Save a backup of `/boot/*.dtb`, `/boot/overlay/`, `/boot/kernel*`, `/lib/modules`, `/lib/firmware` in your host machine (using `scp`);

   c) Remove `/boot/overlays` (`rm -rf`);

   d) Merge the content of `/tmp/boot` with that of `/boot` (using `cp -rd`);

   e) Merge the content of `/tmp/lib` with that of `/lib` (using `cp -rd`)

4. Most people also disable the Low Latency Mode (`llm`) for the SD card. Edit the file `/boot/cmdline.txt` and add at end of the line `sdhci_bcm2708.enable_llm=0`;

5. Now your SD card is updated with the kernel you have built. Reboot your RPi to boot.

5 Preemptible kernel check

The patched kernel may be checked by running the command `uname -a` on your RPi. You can check that full preemption is enabled by compiling the joined file `ifpreempt.c` and running `./ifpreempt`. The program returns `this is a PREEMPT RT kernel`, if it is the case.

6 Select the scheduling policy

1) Which command is used to run a process using a specific scheduling policy? Use this command to display the scheduling policies provided by the kernel.

2) How to run a program using the `SCHED_FIFO` policy under a given priority?

7 Boot time optimization

1) Use the tool `systemd-analyse` to get the whole boot duration and that of each service.

2) Reduce the time boot of your RPi (to be tested at home) by setting a static internet connection.

8 Kernel modules

8.1 Create and compiling modules

Within the working directory `kernel_labs`, create a sub-folder `extra` and move into it:

1) Start by writing a simple module `hello.c` that
   - displays a hello message when it is loaded,
   - displays a goodbye message when it is unloaded, and
   - declares metadata information using the directives `MODULE_LICENSE()`, `MODULE_DESCRIPTION()` and `MODULE_AUTHOR()`;
2) Define a Makefile in order to cross-compile the created module:
   - The Makefile shall define the object output file;
   - It shall point on the path to kernel sources downloaded previously;
   - It should define the `all` option to build the module, and the `clean` one for cleanup.
3) Cross-compile the module and check outputs in case of successful termination.

8.2 Loading/Unloading modules

1) Copy the file `hello.ko` in your RPi under the home directory of the `pi` user;
2) Under your RPi, load the module and check its initialization message;
3) Unload it and check its exit message.

Next lab: Kernel modules ...