

Installing LITMUS-RT on Raspbian OS

Table of Contents

Prerequisites	2
Installing Raspbian OS	2
Obtaining default kernel configuration	2
Kernel cross-compilation.....	3
Installing liblitmus and feather-trace-tools	7
Creating a user space linked with liblitmus.....	8
Important links and further reading.....	9

IMPORTANT NOTE: If you find that over the time some of the instructions cannot be performed, let me know at filipmarkovic.fm@gmail.com so that we can constantly update this guide.

Filip Marković
(Mälardalen University, Sweden)

Prerequisites

- Preferably **Ubuntu host** (since this tutorial used it) but you may try some other Linux derivation.
- (On your Ubuntu host): Downloaded **Raspbian image**: 2017-07-05-raspbian-jessie or 2017-06-21-raspbian-jessie. Both of them worked with my Raspbian Pi 3 without problems, light and full versions. You can obtain an image on the following URL: <https://downloads.raspberrypi.org/raspbian/images/>
- **SD card reader**, and **SD card** above 8GB.
- **Raspberry Pi 3**, but you may try the tutorial with any other. It should work as well (only few steps differ, and they are covered here).

Installing Raspbian OS

First, install Raspbian image on your Raspberry Pi 3. We will describe the instructions using balenaEtcher tool.

1. Install and Open balenaEtcher,
2. Plug in your SD card into SD card reader
3. Select image (point to your Raspbian image that you previously downloaded)
4. Select drive (point to your SD card)
5. Flash.

When the process is finished, your SD card will contain two partitions, one boot FAT32, and one root EXT4. If you use any other install method, just be sure that the same partition types are created as well.

Obtaining default kernel configuration

In these steps, we will obtain a kernel configuration file from the installed Raspbian OS.

Put the SD card to the Raspberry card slot and turn on your Raspberry Pi. Wait for it to boot, and then generate config.gz file. We do this step by running the following command on the Raspbian system:

```
sudo modprobe configs
```

and config.gz file will be located in the *proc/* directory. Copy this file and paste it to some user-space directory, because *proc/* directory is emptied once the device is turned off. Of course, remember the location of the file. Now, turn off raspberry and insert SD card into the SD card reader.

Next, on the Ubuntu host, plug in the SD card reader and perform the following steps.

In these steps, we will obtain a raspberry pi Linux kernel repository on our Ubuntu host, then we will patch Litmus-RT to the Linux kernel repository.

First, clone the Raspberry pi Linux kernel repository:

```
git clone https://github.com/raspberrypi/linux.git
cd linux
git checkout rpi-4.9.y-stable
```

Now, add the Litmus-RT repository and fetch it.

```
git remote add litmus https://github.com/LITMUS-RT/litmus-rt.git
git fetch litmus
git remote -v
```

At the end, we cherry-pick the commits from litmus branch and apply them onto the raspberry pi Linux kernel with the following command.

```
git cherry-pick rpi-4.9.y-stable..litmus/linux-4.9-litmus
```

if there is an error “Apr 6 19:38... error: commit ae76... is a merge but no -m option was given”, try to cherry-pick the exact commit and then perform the following:

```
git cherry-pick -m 1 ae76...
git commit --allow-empty
git cherry-pick --continue
```

Kernel cross-compilation

First, we install all necessary libraries for Raspbian kernel cross-compilation

```
$ sudo apt-get install libncurses5-dev libncursesw5-dev
```

Install toolchain on your Linux host (in my case Ubuntu). You probably can reduce the following steps by just performing:

```
apt install gcc-arm-linux-gnueabi
```

But if that does not work, then:

Use the following command to download the toolchain to the home folder:

```
git clone https://github.com/raspberrypi/tools ~/tools
// or just
```

Updating the \$PATH environment variable makes the system aware of file locations needed for cross-compilation. On a 32-bit host system you can update and reload it using:

```
echo PATH=\$PATH:~/tools/arm-bcm2708/gcc-linaro-arm-linux-  
gnueabihf-raspbian/bin >> ~/.bashrc  
source ~/.bashrc
```

If you are on a 64-bit host system, you should use:

```
echo PATH=\$PATH:~/tools/arm-bcm2708/gcc-linaro-arm-linux-  
gnueabihf-raspbian-x64/bin >> ~/.bashrc  
source ~/.bashrc
```

Prior cross-compiling the kernel, we first need to configure the current configuration file from our Raspbian OS, because we use this config file to create new kernel files which include LITMUS-RT.

Copy the previously remembered config.gz file into your Ubuntu host system and extract it inside the linux (git source) directory. After this step, you obtain the .config file which has all the information about the kernel configuration.

```
zcat config.gz > <path-to-git-linux-directory>/linux/.config
```

Then, run the following cross compilation command to configure the current kernel configuration file (.config), using menuconfig parameter:

```
cd linux  
KERNEL=kernel7  
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- menuconfig
```

To configure the kernel, the following steps are necessary:

1. Add a recognizable local version, such as `-litmus` (or whatever you want). It is used to show that you are running the compiled kernel. This will be necessary after we have the running kernel on the board in order to insure what kernel is currently running. This can be entered under **General setup->Local version** - append to kernel release.
2. Enable in-kernel preemptions. This can be set under **Kernel features->Preemption model**. Choose **Preemptible Kernel (Low-Latency Desktop)**.
3. Disable group scheduling. First, **disable the Automatic process group scheduling** option under **General setup**. Second, under **General setup->Control group support->CPU controller**, **disable Group scheduling for SCHED_OTHER**.
4. Disable frequency scaling and power management options that affect timer frequency. Under **General setup->Timers subsystem->Timer tick handling**, set the option to **constant rate, no dynticks**. Under **Power management options**, make sure that **Suspend to RAM and standby, Hibernation and Opportunistic sleep** are **disabled**. Under **CPU Power Management->CPU Frequency scaling**, **disable CPU Frequency scaling**.

5. When planning to do development, enable tracing in LITMUS^RT. Under LITMUS^RT- >Tracing, enable TRACE() debugging. Note that this is a high-overhead debug tracing interface that must not be enabled for any benchmarks or production use of the system

Now, cross-compile the kernel with the following command, this might take some time, so add more processors with the -j parameter

```
KERNEL=kernel7  
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- -j 2
```

Note that KERNEL=kernel7 is a command for Pi 2, Pi 3, Pi 3+, or Compute Module 3. For Pi 1, Pi Zero, Pi Zero W, or Compute Module try to execute KERNEL=kernel first, and for Pi 4 KERNEL=kernel7l

then for any of them, execute the following:

```
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- zImage modules  
dtbs -j 2
```

Having built the kernel, you need to copy it onto your Raspberry Pi and install the modules; this is best done directly using an SD card reader.

First, use lsblk before and after plugging in your SD card to identify it. You should end up with something like this:

```
sdb  
  sdb1  
  sdb2
```

with sdb1 being the FAT (boot) partition, and sdb2 being the ext4 filesystem (root) partition.

If it's a NOOBS card, you should see something like this:

```
sdb  
  sdb1  
  sdb2  
  sdb5  
  sdb6  
  sdb7
```

with sdb6 being the FAT (boot) partition, and sdb7 being the ext4 filesystem (root) partition.

Mount these first, adjusting the partition numbers for NOOBS cards (if necessary):

```
mkdir mnt  
mkdir mnt/fat32  
mkdir mnt/ext4
```

```
sudo mount /dev/sdb1 ~/mnt/fat32
sudo mount /dev/sdb2 ~/mnt/ext4
```

```
cd linux
sudo make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi-
INSTALL_MOD_PATH=~/.mnt/ext4 modules_install
```

Check the files in the directories with `ls -a`. Finally...

```
cd // get in the parent directory of mnt
sudo cp mnt/fat32/$KERNEL.img mnt/fat32/$KERNEL-backup.img
cd linux
sudo cp arch/arm/boot/zImage ~/mnt/fat32/$KERNEL.img
sudo cp arch/arm/boot/dts/*.dtb ~/mnt/fat32/
sudo cp arch/arm/boot/dts/overlays/*.dtb* ~/mnt/fat32/overlays/
sudo cp arch/arm/boot/dts/overlays/README ~/mnt/fat32/overlays/
sudo umount ~/mnt/fat32
sudo umount ~/mnt/ext4
```

Insert the SD card in the RaspberryPi and boot.

- For image 2017-07-05-raspbian-jessie, you need to wait for approximately 5 minutes in order for boot to finish. After that, power-off -p, and reboot.
- Note: If boot hangs at “random: crng init done”, just unplug and plug Rpi again. This happened to me on lite Raspbian image. However, after a new reboot, everything worked.

Finally, check if `uname -r` gives you the linux version (in my case 4.9.80-litmus+) and if the following command yields LITMUS schedulers:

```
$ cat /proc/litmus/plugins/loaded
```

Installing liblitmus and feather-trace-tools

In your raspberry, update the following packages in order to be able to compile liblitmus and feather-trace tools:

```
sudo apt install libncurses-dev git libssl-dev
```

navigate to some directory where you want to add litmus, liblitmus and feather-trace file directories. It is recommended that they have the same parent directory. In my case, I created a directory named “litmus-related” in “/opt” directory.

To install liblitmus, we first need to *git clone litmus-rt*

```
git clone https://github.com/LITMUS-RT/litmus-rt.git
```

Then, we clone litmus repository in the same parent directory

```
git clone https://github.com/LITMUS-RT/liblitmus.git
```

and after that we just execute:

```
cd liblitmus  
make
```

Note: if you do not have the same parent directory for litmus-rt and liblitmus directories, then you need to download .config file, using the main guide page (available in links at the end)

IMPORTANT: Now you need to add the PATH to the liblitmus directory to your .bashrc file, in order to be able to execute liblitmus functions:

```
nano .bashrc
```

add the following line at the end of the file and afterwards run “source .bashrc”

```
export PATH=/opt/litmus-related/liblitmus:$PATH
```

Also, in order to run setsched command, you need to install dialog, as follows:

```
apt-get install dialog
```

To install feather-trace-tools, perform git clone in the parent directory, as in official LITMUS-RT guide:

```
git clone https://github.com/LITMUS-RT/feather-trace-tools.git
```

upon successful cloning, change directory to feather-trace-tools, and then compile:

```
make
```

Note, you should also add feather-trace-tools directory to the \$PATH,

```
export PATH=/opt/litmus-related/liblitmus:/opt/litmus-  
related/feather-trace-tools:$PATH
```

Finally, for the sake of the remaining steps of the tutorial, set the scheduler to P-FP which is partitioned fixed-priority scheduling, i.e. run in the terminal

```
set-sched P-FP
```

In the next section of this tutorial, we create a user-space which is linked with LIBLITMUS, where we can create our own real-time tasks.

Creating a user space linked with liblitmus

First, in the parent directory of liblitmus, litmus-rt and feather-trace-tools, create a new directory, e.g. called mytools/.

```
mkdir mytools
```

and then the src/ directory:

```
cd mytools  
mkdir src
```

Finally, add the mytools directory to the PATH variable in your .bashrc file.

```
export PATH=/opt/litmus-related/liblitmus:/opt/litmus-  
related/feather-trace-tools:/opt/litmus-related/mytools:$PATH
```

Important links and further reading

In order to create your own tasks within user-space, trace their execution, and perform other Liblitmus operations, refer to:

<https://wiki.litmus-rt.org/litmus/LinkAgainstLiblitmusTutorial>

<http://www.litmus-rt.org/tutorial/manual.html#writingreal-timetasksfromscratch>

And other documentation available at <http://www.litmus-rt.org/>

Other links that were helpful for this tutorial are:

<http://www.litmus-rt.org/installation.html>

<https://www.raspberrypi.org/documentation/linux/kernel/building.md>

<https://lists.litmus-rt.org/pipermail/litmus-dev/attachments/20180518/abc5a917/attachment-0001.pdf>

Many thanks to people who worked and are still working on Litmus-RT, and Mercea Otniel Bogdan who created a tutorial for LITMUS-RT installation on ArchLinux!