






#Raspberry Pi 4B: Real-Time System using Preempt-RT (kernel 4.19.y)

 [Raspberry Pi](#), [Real Time Systems](#) |  7 min |  29845






 [preempt-rt kernel](#)  [performance](#)  [real time system](#)  [raspberry_pi 4b](#)  [raspbian buster](#)

Table of content

Hide

Hardware & Software

Getting the Sources

Requirements

Configuration

Toolchain

Building the Kernel Configuration

Compiling the Kernel

Transfer the Kernel

Installing the Kernel Image, Modules & Device Tree Overlays

Performance Test

RT-Tests suite

Conclusions

Tips and Solutions



I also uploaded the compiled and patched kernel to [lemariva/RT-Tools-RPi](#). You can follow all these steps and compile the kernel by yourself or you can download the files from the repository and deploy the kernel on your Raspberry Pi 4B.

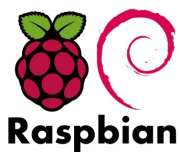
This tutorial is the second of a series of three that evaluate the performance of the Raspberry Pi 4B. This time the performance of the standard and Preempt-RT kernels is evaluated and compared. The planned articles are the following:

- -The first article was about comparing the Raspberry Pi 4B and 3B+: [#Raspberry Pi 4B: How much faster is the new CPU?](#).
- -This article compares the Raspberry Pi 4B running standard and Preempt-RT kernels.
- -The performance of the Raspberry Pi 4B and 3B+ with standard kernel ~~will be~~ is compared with and without active cooling: [#Raspberry Pi 4B: Sometimes it's cool to be hot -or warm, to be exact ;\)](#)

The tutorial is written in a compact way. A full version with more explanations can be found in the previous article: [#Raspberry Pi: Real Time System - Preempt-RT Patching Tutorial for Kernel 4.14.y](#). If you have any problems, check the `Tips` and `Solutions` section of this and the previous article.

Hardware & Software

In these articles, the following software and hardware are used:



Raspbian Pi OS Lite



Raspbian Tests & Tutorial Data





Raspberry Pi 3 B+



Raspberry PI 4B



N-Queens Problem Benchmark



ICE-Tower CPU Cooling Fan



Getting the Sources

For this tutorial, you need a host computer running Linux. In my case, I am using Ubuntu 18.04 LTS, but the tutorial should work with any version of Linux.

Very important: I repeat it again, but this time, I use some colors! Most of this tutorial (configuring and compiling the Kernel) is performed on a **host computer (x86/x64) running Linux, not on the Raspberry Pi!**. Only the deployment is realized on the Raspberry Pi.



Shell

```
~$ mkdir ~/rpi-kernel
~$ cd ~/rpi-kernel
~rpi-kernel$ mkdir rt-kernel
```

Then, clone the following repositories:

Shell

```
~/rpi-kernel$ git clone https://github.com/raspberrypi/linux.git -b rpi-4.19.y-rt
~/rpi-kernel$ git clone https://github.com/raspberrypi/tools.git
```

The Raspberry PI kernel source will be downloaded to the `linux` subdirectory (1.5-2 GB) and the Raspberry PI cross-compilers to the `tools` subdirectory (1 GB).

Configuration

Toolchain

You need to set the following variable before starting to configure and/or compile the kernel source:

Shell

```
~/rpi-kernel$ export ARCH=arm
~/rpi-kernel$ export CROSS_COMPILE=~/.rpi-kernel/tools/arm-bcm2708/gcc-linaro-arm-linux-gnueabihf/bin/arm-linux-gnueabihf-
~/rpi-kernel$ export INSTALL_MOD_PATH=~/.rpi-kernel/rt-kernel
~/rpi-kernel$ export INSTALL_DTBS_PATH=~/.rpi-kernel/rt-kernel
```

Building the Kernel Configuration

Depending on your current Raspberry Pi hardware, you need to set the `KERNEL` variable and make the right configuration following this:

- Raspberry Pi 1/1.2 B(+), A(+), Zero (W):

Shell

```
~/rpi-kernel$ export KERNEL=kernel
~/rpi-kernel$ cd ~/rpi-kernel/linux/
~/rpi-kernel/linux/$ make bcmrpi_defconfig
```



```
~/rpi-kernel$ export KERNEL=kernel/  
~/rpi-kernel$ cd ~/rpi-kernel/linux/  
~/rpi-kernel/linux/$ make bcm2709_defconfig
```

- Raspberry Pi 4B:

Shell

```
~/rpi-kernel$ export KERNEL=kernel7l  
~/rpi-kernel$ cd ~/rpi-kernel/linux/  
~/rpi-kernel/linux/$ make bcm2711_defconfig
```

This is very important, otherwise you are going to waste your time compiling a kernel that is not going to boot on your system.

Compiling the Kernel

To compile the kernel you need to type the following:

Shell

```
~/rpi-kernel/linux$ make -j4 zImage  
~/rpi-kernel/linux$ make -j4 modules  
~/rpi-kernel/linux$ make -j4 dtbs  
~/rpi-kernel/linux$ make -j4 modules_install  
~/rpi-kernel/linux$ make -j4 dtbs_install
```

Choose the right `-jX` parameter according to the number of processors that your host computer has. In my case 4. Take a coffee or may be 2! ;)

The last line returned after installing `modules_install` reports the kernel version that you compiled, e.g.:

Shell

```
DEPMOD 4.19.59-rt23-v7l+
```



You'll need this information for the kernel deployment. Then, make just a blob of data at the end of the kernel image typing:



```
~/rpi-kernel/linux$ cd $INSTALL_MOD_PATH/boot  
~/rpi-kernel/rt-kernel/boot$ mv $KERNEL.img kernel7_rt.img
```

Transfer the Kernel

After the compilation is completed, compress all files to transfer them to the Raspberry Pi:

```
Shell  
~/rpi-kernel/linux$ cd $INSTALL_MOD_PATH  
~/rpi-kernel/rt-kernel$ tar czf ../rt-kernel.tgz *
```

Then, transfer the resulting '.tgz' file to the Raspberry Pi using `scp` and your ssh credentials:

```
Shell  
~/rpi-kernel/rt-kernel$ cd ..  
~/rpi-kernel$ scp rt-kernel.tgz pi@<ipaddress>:/tmp
```

Change `<ipaddress>` to the corresponding IP of your Raspberry Pi.

Installing the Kernel Image, Modules & Device Tree Overlays

Before you start doing this, be sure that you've already saved the important data from your Raspberry Pi (maybe you should do a MicroSD card backup). This tutorial helps you to install the kernel version 4.19.y. Discussion for kernel compatibilities are [here](#).

If you are sure to continue, type the following on the Raspberry Pi:

```
Shell  
~$ cd /tmp  
/tmp$ tar xzf rt-kernel.tgz  
/tmp$ cd boot  
/tmp/boot$ sudo cp -rd * /boot/  
/tmp/boot$ cd ../lib  
/tmp/lib$ sudo cp -dr * /lib/  
/tmp/lib$ cd ../overlays  
/tmp/overlays$ sudo cp -d * /boot/overlays  
/tmp/overlays$ cd ..  
/tmp$ sudo cp -d bcm* /boot/
```



```
# Add the following option:  
kernel=kernel7_rt.img
```

Reboot the Raspberry Pi and if all the stars are aligned, you get the Preempt-RT kernel working! I am just kidding, It should work without any problems! ;P. You can test if the kernel is working, typing:

```
Shell  
~$ uname -r  
4.19.59-rt23-v7l+
```

Performance Test

Again, I used Visual Studio Code (VSCode) and the Remote-SSH extension (read the VSCode part of [this article](#)) to connect to the Raspberry Pi and install Python3-pip, git and the Python3 needed extension as:

```
Shell  
pi@raspberrypi:~$ sudo apt-get install python3-pip git  
pi@raspberrypi:~$ pip3 install tqdm
```

Then, the performance of the Raspberry Pi 4B resolving the N-queens problem (multi/single-thread) can be measured using the following code:

```
Shell  
pi@raspberrypi:~$ git clone https://github.com/lemariva/N-Queens-Problem.git  
pi@raspberrypi:~$ cd N-Queens-Problem  
# multi-thread (N=12, Threads=4, Repetition=100)  
pi@raspberrypi:~/N-Queens-Problem$ python3 queenpool_multithread.py multithread_output.csv  
[...]  
  
# single-thread (N=12, Threads=1, Repetition=100)  
pi@raspberrypi:~/N-Queens-Problem$ python3 queenpool_multithread.py singlethread_output.csv  
[...]
```



A comparison of the performance between standard and Preempt-RT patched kernel in multi- and single-thread configuration can be found in Fig. 1 and 2, respectively. The sub-figure on the left describe the results for the standard kernel while the sub-figure on the right the corresponding results for the Preempt-RT.



Fig. 1a: Standard Kernel - Raspberry Pi 4B**Multi-thread Configuration****Fig. 1b: Preempt-RT - Raspberry Pi 4B****Multi-thread Configuration****Fig. 2a: Standard Kernel - Raspberry Pi 4B****Single-thread Configuration****Fig. 2b: Preempt-RT - Raspberry Pi 4B****Single-thread Configuration**

The Preempt-RT patched kernel was 1.11x/1.01x (multi/singlethread) slower to resolve the N-queens-problems. The loss of performance is not considerable large. The temperature values increased only about 0.7°C. In this test, unlike the one carried out with the Raspberry 3B+, an SSH section was connected to the Raspberry Pi, the whole time. The problem with IRQ-39 did not appear on the new Raspberry using the network chip. This problem was observed in the previous model because the network uses the same chip and therefore BUS as the USB ports.

The numerical values can be found in the following table:

	rPi 4 Model B	rPi 4 Model B	rPi 3 Model B+	rPi 3 Model B+	rPi 3 Model B+
	Preempt-RT Buster	Std. Buster	Std. Buster	Preempt-RT Stretch	Std. Stretch
Avg. Multi-Thread Solving Time	24.27 s	21.79 s	49.83 s	70.38 s	62.66 s
Multi-Thread Max. Temperature	82.16 °C	81.40 °C	77.63 °C	69.55 °C	68.78 °C
Avg. Single-Thread Solving Time	67.55 s	66.72 s	169.96 s	235.75 s	213.34 s
Single-Thread Max. Temperature	70.56 °C	67.09 °C	60.39 °C	56.91 °C	54.22 °C



RT-Tests suite



Shell

```
sudo cyclicttest -l500000000 -m -S -p90 -i200 -h400 -q > output.txt  
grep -v -e "^#" -e "^$" output.txt | tr " " "," | tr "\t" "," > histogram.csv  
sed -i '1s/^/time,core1,core2,core3,core4\n /' histogram.csv
```

Additionally, I measured the CPU and GPU temperatures to see if they remained constant during the test, and between the tests with the standard and Preempt-RT kernels. To do that I used the following code:

```
#!/bin/bash  
  
while true  
do  
    cpuTemp0=$(cat /sys/class/thermal/thermal_zone0/temp)  
    cpuTemp1=$(( $cpuTemp0/1000 ))  
    cpuTemp2=$(( $cpuTemp0/100 ))  
    cpuTempM=$(( $cpuTemp2 % $cpuTemp1 ))  
    clear  
    echo "CPU temp=$cpuTemp1.$cpuTempM'C"  
    echo "GPU $(/opt/vc/bin/vcgenclmd measure_temp)"  
    sleep 5  
done
```

Figure 3 presents the temperature and latency, that result from using Standard (3a) and Preempt-RT Kernels (3b).

**Fig. 3a: Latencies and Temperature using
Std. Raspbian Kernel (4.19.57-v7l+)**

**Fig. 3b: Latencies and Temperature using
Preempt-RT Raspbian Kernel (4.19.59-rt23-v7l+)**

The CPU and GPU temperatures were almost the same for both kernel on the tests. No significant difference was registered (improvement w.r.t. [to the version 3B+](#)). The maximal latency response for the real-time kernel was 93 us while the test with the standard kernel reached 301 us. Thus, the latency was 3.23x optimized using the Preempt-RT patched kernel.



I did not expect such good results. If latency is important for your application, apply a Preempt-RT patch to the Kernel but use only a Raspberry Pi 4B. The Raspberry Pi 3B+ loses too much performance with the RT-kernel.

However, remember that the Raspberry Pi 4B is a general-purpose single-board computer and not a dedicated real-time system. I write this because I was contacted this year by some people who wanted to replace FPGA systems with a Raspberry Pi and wanted to acquire data, perform processing (FFT) and make decisions in less than 1 ms. Something impossible for this type of systems. Low latencies bring Raspberry Pi closer to a real-time type of system, but it is still far from dedicated and programmed systems for a specific activity (FPGA, microcontrollers, etc.)

Tips and Solutions

1. If you get this error on your host PC:

```
Shell
scripts/extract-cert.c:21:10: fatal error: openssl/bio.h: No such file or directory
#include <openssl/bio.h>
```

install the following: `sudo apt-get install libssl-dev`

Related posts

**#Raspberry Pi: Preempt-RT vs. Standard Kernel
4.14.y**

**#Raspberry PI Zero W: Preempt-RT Kernel
Performance**

**#Raspberry Pi: The N-queens Problem
(benchmark) Preempt-RT vs. Standard Kernel**

**#Raspberry Pi: Real Time System - Preempt-RT
Patching Tutorial for Kernel 4.14.y**



**#Raspberry Pi: Real Time System - Xenomai
Patching Tutorial for Kernel 4.14.y**

**#Raspberry Pi 4B: Where were the quality
controls?**

**#Orange Pi Zero LTS: An 'upgraded' alternative
to the old Raspberry Pis**

**#Raspberry Pi 4B: How much faster is the new
CPU?**

Comments



John Talbot 10.23.2019

Great article. I did not know about Isolcpus so thanks. After booting the rpi-4.19.y-rt kernel that I built on my Pi 3B+, I got the following error messages on the screen

```
*** 4 Raspberries ***
raspberrypi-firmware soc:firmware: Request 0x00040013 returned 0x01
bcm2708_fb soc:fb: Unable to determine number of FB's. Assuming 1
**** end of messages ****
```

I was first confused by `bcm2708_fb` , but it was a read hearing. I googled the error and determined fb stood for frame buffer. I diffed a working `.config` file with the one created by `bcm2709_defconfig` . After a process of elimination, I found that changing `CONFIG_FB_SIMPLE` too undefined solved the problem.

I am letting you know, just in case someone else ever has this issue.



Thanks for the great article, John

[permalink](#) [reply](#) [show all replies](#)



Michael Ruder 10.24.2019

Hi,

thanks for the article! It seems that in 4.19-rt the issue that already appeared in 4.14-rt is back:

<https://github.com/raspberrypi/linux/issues/2943>

This results in considerable CPU load from the dwc_otg USB driver. For 4.14-rt there was a patch, that does not work for 4.19-rt.

When I tried the 4.19-rt on a RPI3B+, I also had the CPU load issue, but I did not try the cmdline.txt options suggested yet.

I am also not sure if the RPI4B is affected by this at all. Did you notice CPU load from dwc_otg? Did you set any special cmdline.txt options?

Thanks and best regards!

[permalink](#) [reply](#) [show all replies](#)



Detlef Metje 10.27.2019

Thanks for this article. I wanted to use the rt kernel on my raspberry pi 3B. As I wanted to build the kernel configuration on my ubuntu 19.04 I got the following message I didn't understand.

```
YACC    scripts/kconfig/zconf.tab.c
/bin/sh: 1: bison: not found
make[1]: *** [scripts/Makefile.lib:196: scripts/kconfig/zconf.tab.c] Fehler 127
make: *** [Makefile:534: bcm2709_defconfig] Fehler 2
detlef@Studio:~/rpi-kernel/linux$ ^C
detlef@Studio:~/rpi-kernel/linux$
```

Could you please give me a hint what went wrong? Thanks Detlef

[permalink](#) [reply](#) [show all replies](#)



Detlef 10.27.2019

Hi! I solved the problem by installing bison and flex. Sometimes you are next to yourself! Thanks Detlef





Jimmi 01.19.2020

Using the option `--depth=1` with 'git clone' will considerably reduce the size of download. Is there any reason for not to use it?

[permalink](#) [reply](#) [show all replies](#)



John 04.13.2020

There appears to be a typo where you define your `CROSS_COMPILE` variable.

`export CROSS_COMPILE=~/rpi-kernel/tools/arm-bcm2708/gcc-linaro-arm-linux-gnueabi-hf-raspbian-x64/bin/arm-linux-gnueabi-hf-` It looks like the text was clipped.

[permalink](#) [reply](#) [show all replies](#)



benjaminh 06.15.2020

Hi, I did the steps as described above. Then, after install I checked the kernel config. `CONFIG_HZ_100=y` and `CONFIG_HZ=100` Is it possible to change the kernel settings after installation on `rpi`? Thanks Kind regards, Benjamin

[permalink](#) [reply](#) [show all replies](#)



tk 06.18.2020

Is Raspberry Pi Model B Rev 2 (BCM2835) supported?

[permalink](#) [reply](#) [show all replies](#)



wooden 07.02.2020





alex 07.04.2020

i got the rt kernel to boot but the USB devices dont work, i get this message during boot

xhci_hcd 0000:01:00.0: cant setup: -110 xhci_hcd 0000:01:00.0: init 0000:01:00.0 fail, -110

i can ssh into the pi, any idea how to fix the usb problem? thanks!

[permalink](#) [reply](#) [show all replies](#)

1 2 [Next →](#)

 Name

 E-Mail

type your message here!



I'm not a robot

reCAPTCHA
[Privacy](#) - [Terms](#)

SEND



Trending Posts
Last 7 Days

[#Raspberry Pi 4: Hardware accelerated video decoding_\(GPU\) in Chromium](#)



[Privacy](#) - [Terms](#)

[#Raspberry Pi 4: booting from an SSD with enabled TRIM](#)

🕒 9 min | 🔍 3889

[#Raspberry Pi: Amazon Prime, Netflix, etc. and a DRM solution!](#)

🕒 3 min | 🔍 8009

[#Raspberry Pi 4B: Real-Time System using Preempt-RT \(kernel 4.19.y\).](#)

🕒 7 min | 🔍 29845

Blog Topics

[Analytics](#) (10)

[Android Things](#) (15)

[Cloud Platforms](#) (6)

[General](#) (18)

[Hacking](#) (5)

[Home Automation](#) (10)

[M5Stack](#) (13)

[MicroPython](#) (42)

[LoRa/LoRaWAN](#) (5)

[PCB/Board designs](#) (6)

[Product Reviews](#) (15)

[Raspberry Pi](#) (21)

[Real Time Systems](#) (9)

[Travel Photoset](#) (2)

[Single-board Computers](#) (3)



