Dedicated To

My Family

Who supported me thick and thinPreface

First of all, Congratulations on investing your hard earned money on this book. Who should read this book past this page?? A Techie by heart, A Hobbyist by nature & A Hacker by chance. Someone looking forward to have a rocking time with this magical box, labeled Raspberry Pi.

The Author was definitely as excited as you are right now to begin his hacking spree, but it was when I faced numerous challenges (both on Hardware and Software level), that I decided to capture my daily learning’s in the form of simple notes on notepad. And then, It all started, with my collection of simple notes to capture new learning’s, which has now taken the shape of a book.

The basic idea behind publishing this book, is to cut short the plethora of information available on various websites on how-to build System Image for Raspberry-Pi, and to help readers quickly jump to the practical stuff that matters, without wasting time and energy on.

All said and done, I would personally love to hear from you, your suggestions to make this book even better.

Regards,

Sudhanshu Gupta

Founder , CEO

[www.softwaresunleashed.com](http://www.softwaresunleashed.com)

E-mail : softwares.unleashed@gmail.com

#### Macintosh HD:Users:sudhanshu:Documents:R_Drive:01_Programming_Stuff:09_SoftwaresUnleashed:QR_Codes:Website_SoftwaresUnleashed:code_small.png

# About the Author

Sudhanshu Gupta (Founder – Softwares Unleashed), is a B.Tech in Electronics & Tele-Communications & M.S. in Softwares Systems, and has 12+years of experience in Telecom domain and Embedded Software development.

He had worked with Major Industry gaints, LG, Infineon, Intel to name a few. Sudhanshu during his stint with the corporate world, has contributed to numerous success stories of Big OEMs (LG, Samsung, Nokia)…cutting short the list.

He is now on a fast track to take his passion forward, ie. Application Development and Sharing his Technical Knowledge for the benefit of others.

# Table Of Contents

About the Author 3

Table Of Contents 4

About Raspberry Pi 6

Preparing Raspberry Pi for First Boot 7

Hardware Requirements 7

Install Raspbian on Your Pi and Connect to It Over SSH 7

Step By Step Guide (Building Custom Linux Kernel) 8

1) Get the kernel source 8

2) Get the cross-compiler 8

3) Install necessary packages 9

4) Code Compilation 10

5) Preparing the SYSTEM IMAGE 12

6) Transfer the Kernel Image 13

7) Building the Device Drivers (Modules) 14

8) Updating the GPU(Graphics Processing Unit) firmware 14

9) Transfer the firmware 15

Installing Latest Ubuntu Kernel 16

Step 1. Get the latest Linux kernel source code 16

Step 2. Configure the Linux kernel features and modules 16

Step 3. Install the required compilers and other tools 17

Step 4. Configuring the kernel 17

Step 5. How to compile a Linux Kernel 18

Step 6. Install the Linux kernel modules 18

Step 7. Install the Linux kernel & Update grub config 18

Step 8. Reboot 19

Welcome to the World of Linux 20

01. Introduction 20

02. Application Programming - User Space 20

03. Linux Device Driver Programming - Kernel Space 20

Application Programming - User Space 21

Linux Device Drivers 22

Introduction to LDD 22

First Device Driver 23

Module Parameter 24

Major Minor Number 25

Device File Creation 26

CDEV File Operations 27

Real Device Driver 28

IOCTL 29

ProcFs 30

WaitQueue 31

SysFs 32

Interrupt Tutorial 33

Interrupt Programming 34

WorkQueues – I 35

WorkQueues - II 36

IoT & RaspberryPi 37

1) Cloud Temperature Monitor 37

Troubleshoot Guide 38

Installing Kernel Headers 38

Error : do\_IRQ: x.yy No IRQ handler for vector 39

Further Reference(s) 43

Legends 44

# About Raspberry Pi

The Raspberry Pi is a series of credit card-sized single-board computers developed in the United Kingdom by the Raspberry Pi Foundation to promote the teaching of basic computer science in schools and in developing countries. Now over four years old, the Raspberry Pi, a cheap credit card sized computer, has taken the computing and DIY world by storm.

The original model became far more popular than anticipated, selling outside of its target market for uses such as robotics.

According to the Raspberry Pi Foundation, over 5 million Raspberry-Pi(s) have been sold before February 2015, making it the best-selling British computer.

#### Overview

Several generations of Raspberry-Pi(s) have been released. The first generation (Raspberry Pi 1 Model B) was released in February 2012. It was followed by a simpler and inexpensive model Model-A. In 2014, the foundation released a board with an improved design in Raspberry Pi 1 Model B+. The model laid the current "mainline" form-factor. Improved A+ and B+ models were released a year later. A cut down "compute module" was released in April 2014, and a Raspberry Pi Zero with smaller size and limited input/output (I/O) and general-purpose input/output (GPIO) abilities was released in November 2015 for US$5. The Raspberry Pi 2, which added more RAM, was released in February 2015. Raspberry Pi 3 Model B released in February 2016 is bundled with on-board WiFi and Bluetooth. As of December 2016, Raspberry Pi 3 Model B is the newest mainline Raspberry Pi. These boards are priced between US$5–35.

All models feature a Broadcom system on a chip (SoC), which includes an ARM compatible central processing unit (CPU) and an on chip graphics-processing unit (GPU, a VideoCore IV). CPU speed ranges from 700 MHz to 1.2 GHz for the Pi 3 and on board memory range from 256 MB to 1 GB RAM. Secure Digital (SD) cards are used to store the operating system and program memory in either the SDHC or MicroSDHC sizes. Most boards have between one and four USB slots, HDMI and composite video output, and a 3.5 mm phone jack for audio. Lower level output is provided by a number of GPIO pins which support common protocols like I²C. The B-models have an 8P8C Ethernet port and the Pi 3 has on board Wi-Fi 802.11n and Bluetooth.

The Foundation provides Raspbian, a Debian-based Linux distribution for download, as well as third party Ubuntu, Windows 10 IOT Core, RISC OS, and specialized media center distributions.[8] It promotes Python and Scratch as the main programming language, with support for many other languages. The default firmware is closed source, while an unofficial open source is available.

# Preparing Raspberry Pi for First Boot

## Hardware Requirements

1. Raspberry Pi (ofcourse absolute yes!!)
2. Power adapter (your micro usb phone charger will do).
3. Memory Card (MMCSD Class 10 preferred)
4. Wi-Fi dongle (USB) (good to have)
5. Ethernet Cable to connect to network. (If you don’t have a Wi-Fi dongle)
6. USB Keyboard (optional)
7. HDMI Monitor (optional)

You could hook your Raspberry Pi up to a keyboard and monitor and set things up that way, or you can connect to your Pi over SSH and run every step from the comfort of your laptop. Although my personal favourite is SSH method, which is much easier than begging your friend for a random monitor.

## Install Raspbian on Your Pi and Connect to It Over SSH

Next, you’ll need to flash your Raspberry Pi with Raspbian image.

On a newly installed Rasbian image, try connecting to RPI board via the following default credentials :

Username : pi (Default)

Password : raspberry (Default)

# Step By Step Guide (Building Custom Linux Kernel)

Building the system image for raspberry pie is easier than as depicted on various websites. This step-by-step guide will get your brand new Raspberry Pi box up and running in no time (minus the complex jargon involved). RPi development is best done on a Ubuntu (Debian distro) with access to root privileges on shell. The guide assumes you have access to a shell with root privileges on a Ubuntu laptop / desktop.

NOTE: MAC OSX users may install Virtual Box with the latest Ubuntu distro installed. Then follow the instructions in the subsequent pages. Link to Virtual Box for MAC (<https://www.virtualbox.org/wiki/Downloads>). Also, install the extension pack provided on the website. Extension packs provide a bridge between your host environment (MAC OSX) and the guest environment (Linux Ubuntu)

## 1) Get the kernel source

First things first, we need to pull Linux Kernel source from WWW and store it locally on our hard drive.

~ # mkdir rpi\_4\_x\_xx <enter>

~ # cd rpi\_4\_x\_xx <enter>

~/rpi\_4\_x\_xx # git clone git://github.com/raspberrypi/linux.git <enter>

where x & xx are the major and minor release numbers for Raspberry-Pi linux kernel. Latest RPI Linux Kernel version as of this writing was 4.9.y.

Quick Repo Sync Tip : If you wish to save some time and download just the latest and greatest Linux Kernel available as of date, append “--depth=1” to the git clone command above. Believe me, it saves a lot of download time (and your broadband data cost) , since most of the branching information is redundant to a hobbyist. However, if you are one of those geeky minds who would like to dwell into each and every commit that has ever been done on the repo, feel free to omit “—depth=1”.

## 2) Get the cross-compiler

Since, we shall be building Raspberry Pi system image on a machine, which would be running a CPU with different architecture, we would require to download a Cross-Compiler. A Cross-Compiler is a program that generates code for a target device (Raspberry Pi in this case), although it is executing on a completely different machine with completely different architecture.

Cross compiling from Linux (pre-built bmc2708\_armv6kz compiler)

~ # mkdir rpi\_tools

~ # cd rpi\_tools

~/rpi\_tools # git clone git://github.com/raspberrypi/tools.git <enter>

NOTE: Install git command if not already installed via the following command

# apt-get install git

## 3) Install necessary packages

* On Ubuntu shell execute the following command (requires root privileges)

# apt-get install gcc-arm-linux-gnueabi make ncurses-dev

* Some Editors are handful , if not already installed

# apt-get install vim-gnome

## 4) Code Compilation

#### 1#. Go to the build directory on PC

# cd <path\_to\_kernel\_source\_directory>/linux/

#### 2#. Firstly, ensure your build directory is clean:

<path\_to\_kernel\_source\_directory>/linux/ # make mrproper

#### 3#. Define Cross-Compiler & Processor Architecture

From this point on, if you are cross-compiling, please substitute <your\_compiler> with your compiler binary prefix (e.g. <your\_compiler>=arm-bcm2708hardfp-linux-gnueabi- or arm-linux-gnueabihf-) as each compiler will be named slightly differently. Check your toolchain folder for the compiler you wish to use to compile Linux Kernel for RPi.

Debug Note : Do *not* forget the hyphen (-) at the end of the compiler name. This has caused lots of time wasted and errors resulting in “xxxxxxxxx command not found”.

If you are building on the RPi (although we wont recommend that as build process would be quite slow), remove ARCH=arm CROSS\_COMPILE=<your\_compiler> from each command.

#### 4#. Kernel Config File

You will want to get a working kernel configuration (.config) to start from.

If you are one of those lucky few, who got RPi with a pre-installed image loaded on the memory card, you can get the config file by executing following command on the shell (on the RPi):

/raspberryPi\_root\_folder # zcat /proc/config.gz > .config

& then copy .config file to your build directory (on PC).

OR

**Alternatively**, the default configuration is available in the downloaded kernel source in <path\_to\_kernel\_src\_dir>/linux/arch/arm/configs/bcmrpi\_defconfig.

Copy (& rename) bcmrpi\_defconfig to .config in the build directory

OR

**Alternatively**, execute the following command on shell

<kernel\_src\_dir> $ export ARCH=arm

<kernel\_src\_dir> $ export CROSS \_COMPILE=arm-bcm2708hardfp-linux-gnueabi-

<kernel\_src\_dir> $ make bcmrpi\_defconfig

#### 5#. Setting Build Environment

Ensure that your configuration file is up-to-date by executing the following command...Textual, Sequential access to configuration parameters. (**Quite tedious)**

# make ARCH=arm CROSS\_COMPILE=<your\_compiler> oldconfig

OR

Optionally, if you want to tweak the configuration Graphically & more Organized -- **Better**), run this command on shell:

# make ARCH=arm CROSS\_COMPILE=<your\_compiler> <config\_targets>

Use any one of the following <config\_targets> :

config :- Update config using a Line-oriented program

nconfig :- Update config using a ncurses menu based program

menuconfig :- Update config using a menu based program

**xconfig :- Update config using a QT based front-end**

gconfig :- Update config using a GTK based front-end

NOTE : The configuration info is stored in ".config" file on exit from the configuration menu.

The file is located in the "build artifacts folder" (if mentioned explicitly by the macro KBUILD\_OUTPUT)

e.g. In the build script we can mention the build output folder as

export KBUILD\_OUTPUT=\_build\_output\_folder

#### 6#. Let the Build Begin

Once you have made necessary changes in the Linux Kernel of RPi, you can trigger the build with the following command and have a cup of coffee or your lunch. This shall take time for the first fresh build. Incremental builds there-after shall be much less time consuming.

# make –j<N> ARCH=arm CROSS\_COMPILE=<your\_compiler>

If you are on a multi-core machine, you can make the build faster by appending -j<N> to the build command above. Where ‘N’ is the number of cores on your system plus one.

Quick Tip : Don’t bother to clean object files, in case changes are made only in source files…A change in header file, deserves a cleaner build with object files and library files deleted manually.

## 5) Preparing the SYSTEM IMAGE

Once your linux kernel is successfully built, you need to pack the kernel such that Raspberry Pi likes to have it. Follow the steps below…

#### 1#. Get Build Tools

Because of the way the memory addresses are arranged in the Broadcom SoC (The CPU used on Raspberry Pi), you will need to prepare the compiled image, before uploading it to Memory Card.

If you haven't got the tools directory from the GIT repo, do so now :

# cd ~/rpi\_tools/compiler/tools

~/rpi\_tools/compiler/tools # git clone git://github.com/raspberrypi/tools.git

OR

DOWNLOAD\_FROM\_LINK\_TO\_TAR\_BALL ::

https://github.com/raspberrypi/tools/

#### 2#. Make Image

In the toolchain set, there is a folder called mkimage. Enter this directory, and then run the following:

# ~/rpi\_tools/compiler/tools/mkimage/imagetool-uncompressed.py <kernel\_build\_dir>/arch/arm/boot/zImage

Location of "kernel.img”::

Above command will output a file called "kernel.img" (in the same folder where the python script "imagetool-uncompressed.py" is located.)

Quick Tip [1] : Above python script expects boot-uncompressed.txt file to be present in the same folder as the imagetool-uncompressed.py script. Hence, to get rid of any errors, we need to be execute the python script from the “mkimage” folder , so that boot-uncompressed.txt is available to the python script.

Quick Tip [2] : If you get error regarding "python2" not available, try creating a soft link to python2 as follows:: (not sure why this is needed , but it worked for me)

# ln -s /usr/bin/python2.6 /usr/bin/python2

## 6) Transfer the Kernel Image

Copy your new kernel.img file into the RPi boot partition, though preferably as a new file (such as kernel\_new.img) just in case it doesn't work. If you're building on the RPi, just copy the file to /boot.

If you use a different filename, edit **config.txt** change the kernel line:

---

# Comment out the below line

#kernel=kernel.img

# Add this new line in /boot/config.txt

kernel=kernel\_new.img

---

## 7) Building the Device Drivers (Modules)

Now you need to transfer the Device Drivers (aka Modules).

In the build directory, run the following (substituting <modules\_path> for a folder somewhere (e.g. ~/modules):

<path\_to\_kernel\_source\_directory>/linux/ # make ARCH=arm CROSS\_COMPILE=<your\_compiler> modules\_install INSTALL\_MOD\_PATH=<modules\_path>

The contents of this directory should then be copied into the RPi root directory.

NOTE: If you have rebuilt the new kernel with exactly the same version as the one that's running, you'll need to remove the old modules first. Ideally this should be done offline by mounting the SD card on another system.

## 8) Updating the GPU(Graphics Processing Unit) firmware

Your RPi should now be ready to boot the new kernel. However, at this point it's recommended (not necessary) that you update your GPU firmware and libraries. This is required if you've just moved from 3.2 to 3.6 as the firmware interface has changed.

The "firmware" and "boot files" should be updated at the same time to ensure that your new kernel works properly

"master" - This is the version of firmware currently used in Raspbian (i.e. it works with the 3.2 kernel).

"next" - This is a development branch which provides a newer GPU firmware to work with the updated drivers in the 3.6 kernel.

For the "master" branch:

<path\_to\_folder\_where\_firmware\_is\_to\_be\_stored> # git clone git://github.com/raspberrypi/firmware.git

For the "next" branch:

<path\_to\_folder\_where\_firmware\_is\_to\_be\_stored> # git fetch git://github.com/raspberrypi/firmware.git next:refs/remotes/origin/next

## 9) Transfer the firmware

a) Firstly, update the required boot files in the RPi boot directory with those you've downloaded. These are:

-1- bootcode.bin

-2- fixup.dat

-3- start.elf

Next, you need to copy the VC libraries over.

There are two copies of this: one for **hard float** and one for **soft float**.

To find the correct one that you should be using, run the following command (substituting the program name for your compiler binary as required):

# arm-none-linux-gnueabi-gcc -v 2>&1 | grep hard

If something prints out, and you can see **--with-float=hard**, you need the hard float ones.

NOTE: The current version of Raspbian uses hard float.

b) Remove the /opt/vc directory from the RPi root, then:

For hard float, copy vc from the hardfp/opt directory into /opt in the RPi root directory

Otherwise copy vc from the top-level opt directory into /opt in the RPi root directory.

# Installing Latest Ubuntu Kernel

Let’s assume you don’t have access to any development board and want to learn Linux development on Ubuntu kernel-based host / virtual machine. Follow the guide below to compile and install Ubuntu’s latest kernel image on your development environment :

## Step 1. Get the latest Linux kernel source code

Visit Ubuntu’s WiKi page to get a list of latest repositories.

<https://wiki.ubuntu.com/Kernel/Dev/KernelGitGuide>

Fetch the latest code by using the command git clone <repo\_url> on terminal.

$ git clone git://git.launchpad.net/~ubuntu-kernel/ubuntu/+source/linux/+git/eoan

This will take a long time, so sit back and relax.

There is a tree for each of the currently supported releases as well as any open development and upcoming releases:

|  |  |
| --- | --- |
| eoan | git://git.launchpad.net/~ubuntu-kernel/ubuntu/+source/linux/+git/eoan |
| disco | git://git.launchpad.net/~ubuntu-kernel/ubuntu/+source/linux/+git/disco |
| bionic | git://git.launchpad.net/~ubuntu-kernel/ubuntu/+source/linux/+git/bionic |
| xenial | git://git.launchpad.net/~ubuntu-kernel/ubuntu/+source/linux/+git/xenial |
| trusty | git://git.launchpad.net/~ubuntu-kernel/ubuntu/+source/linux/+git/trusty |
| precise | git://git.launchpad.net/~ubuntu-kernel/ubuntu/+source/linux/+git/precise |

The distro kernel is always on the master branch in these repositories. Each release also has a master-next branch containing the commits that will go onto the master branch and become the next release for that release.

To use git you must have the [git package](http://packages.ubuntu.com/search?keywords=git) installed on your system, which you can install like this (if not already done):

$ sudo apt install git

## Step 2. Configure the Linux kernel features and modules

Before start building the kernel, one must configure Linux kernel features. You must also specify which kernel modules (drivers) needed for your system. The task can be overwhelming for a new user. I suggest that you copy existing config file using the [cp command](https://www.cyberciti.biz/faq/cp-copy-command-in-unix-examples/):

$ cd <root\_folder\_of\_linux>  
$ cp -v /boot/config-$(uname -r) .config

Sample outputs:

'/boot/config-4.15.0-30-generic' -> '.config'

## Step 3. Install the required compilers and other tools

You must have development tools such as GCC compilers and related tools installed to compile the Linux kernel.

Type the following [apt command](https://www.cyberciti.biz/faq/ubuntu-lts-debian-linux-apt-command-examples/) or [apt-get command](https://www.cyberciti.biz/tips/linux-debian-package-management-cheat-sheet.html) to install the same:

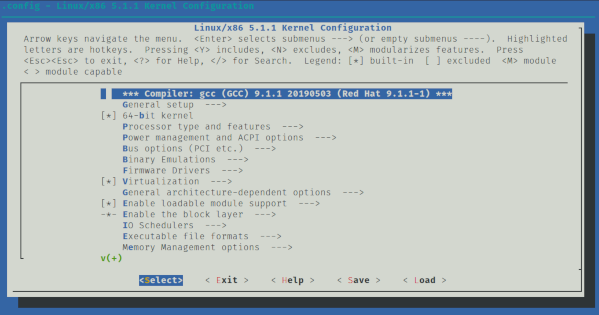
$ sudo apt-get install build-essential libncurses-dev bison flex libssl-dev libelf-dev

## Step 4. Configuring the kernel

Now you can start the kernel configuration by typing any one of the following command in source code directory:

* **$ make menuconfig** – Text based color menus, radiolists & dialogs. This option also useful on remote server if you wanna compile kernel remotely.
* **$ make xconfig** – X windows (Qt) based configuration tool, works best under KDE desktop
* **$ make gconfig** – X windows (Gtk) based configuration tool, works best under Gnome Dekstop.

For example, run make menuconfig command launches following screen:  
$ make menuconfig



You have to select different options as per your need. Each configuration option has HELP button associated with it so select help button to get help. Please note that ‘make menuconfig’ is optional. I used it here to demonstration purpose only. You can enable or disable certain features or kernel driver with this option. It is easy to remove support for a device driver or option and end up with a broken kernel. For example, if the ext4 driver is removed from the kernel configuration file, a system may not boot. When in doubt, just leave support in the kernel.

## Step 5. How to compile a Linux Kernel

Start compiling and to create a compressed kernel image, enter:  
$ make

To speed up compile time, pass the -j as follows:  
## use 4 core/thread ##  
$ make -j 4

## get thread or cpu core count using nproc command ##  
$ make -j $(nproc)

Compiling and building the Linux kernel going take a significant amount of time. The build time depends upon your system’s resources such as available CPU core and the current system load. So have some patience.

## Step 6. Install the Linux kernel modules

$ sudo make modules\_install

## Step 7. Install the Linux kernel & Update grub config

So far we have compiled the Linux kernel and installed kernel modules. It is time to install the kernel itself:  
$ sudo make install

You need to modify Grub 2 boot loader configurations. Type the following command at a shell prompt as per your Linux distro:

$ sudo update-initramfs -c -k 5.4.1  
$ sudo update-grub

It will install three files into /boot directory as well as modification to your kernel grub configuration file:

* initramfs-5.4.1.img
* System.map-5.4.1
* vmlinuz-5.4.1

## Step 8. Reboot

You have compiled a Linux kernel. The process takes some time, however now you have a custom Linux kernel for your system. Let us reboot the system.

Just issue the [reboot command](https://www.cyberciti.biz/faq/linux-reboot-command/) or shutdown command:  
# reboot

Verify [new Linux kernel version after reboo](https://www.cyberciti.biz/faq/find-linux-kernel-version/)t:  
$ uname -mrs

Sample outputs:

Linux 5.4.1 x86\_64

Configurations! You completed various steps to build the Linux kernel from source code and compiled kernel should be running on your system.

# Welcome to the World of Linux

## Introduction

## Application Programming - User Space

## Linux Device Driver Programming - Kernel Space

# Linux Programming - User Space

/\* TODO : Add tutorials to learn User Space programming \*/

# Linux Device Drivers

## Introduction to LDD

## First Device Driver

## Module Parameter

## Major Minor Number

## Device File Creation

## CDEV File Operations

## Real Device Driver

## IOCTL

## ProcFs

## WaitQueue

## SysFs

## Interrupt Tutorial

## Interrupt Programming

## WorkQueues – I

## WorkQueues - II

# IoT & RaspberryPi

## 1) Cloud Temperature Monitor

Have you ever dreamt of controlling your home’s cooling/heating equipment, just to make that perfect ambience by the time you reach your home? All this and more, without you clicking a single button.

Monitoring the temperature of your home remotely, and that too without your intervention could be a bliss, not to mention the optimized communication between various IoT enabled devices, that help minimize your electricity bills.

Following pages will take you through

# Troubleshoot Guide

## Installing Kernel Headers

For developing Linux device drivers, we need to install Linux kernel headers for which we are trying to develop the driver for. Most of the time the headers are included in Linux distro that you are using, however, at times when linux-headers don’t ship out-of-box, they can be easily installed using following commands.

First check your installed kernel version as well as kernel header package that matches your kernel version using following commands.

$ uname -r

$ apt search linux-headers-$(uname -r)

On Debian (including Raspbian for Raspberry Pi), Ubuntu and their derivatives, all kernel header files can be found under /usr/src directory. You can check if the matching kernel headers for your kernel version are already installed on your system using the following command.

$ ls -l /usr/src/linux-headers-$(uname -r)

If the output is empty, that shall confirm that header’s aren’t installed for the current running instance of kernel. And hence needs to be installed.

Before you can install the appropriate kernel headers, update your packages index, in order to grab information about the latest package releases, using the following command.

$ sudo apt update

Then run the following command that follows to install the Linux Kernel headers package for your kernel version.

$ sudo apt install linux-headers-$(uname -r)

Next, check if the matching kernel headers have been installed on your system using the following command

$ ls -l /usr/src/linux-headers-$(uname -r)

## Error : do\_IRQ: x.yy No IRQ handler for vector

Following snapshot shows the error thrown by kernel when we try to raise an interrupt for an already registered IRQ (registered in device driver code via request\_irq() )

A black sign with white text

Description automatically generated

To get rid of them, add a Kernel-Bootoption to your Grub-Bootloader, for Grub2 follow these steps:

1. sudo vi /etc/default/grub

2. Change GRUB\_CMDLINE\_LINUX\_DEFAULT="quiet splash pci=nomsi,noaer"

3. sudo update-grub

The above method used to work on older kernel versions but fails on later versions. The reason is that the generic IRQ handler do\_IRQ() has been changed for better IRQ handling performance. Instead of using the irq\_to\_desc() function to get the IRQ descriptor, it reads it from the per-CPU data. The descriptor is put there during the physical device initialization. Since this pseudo device driver don't have a physical device, do\_IRQ() don't find it there and returns with an error. If we want to simulate IRQ using software interrupt, we must first write the IRQ descriptor to the per-CPU data. Unfortunately, the symbol vector\_irq, the array of the IRQ descriptors in the per-CPU data, is not exported to kernel modules during kernel compilation. The only way to change it, is to recompile the whole kernel (checkout section that talks about Building Ubuntu image from source code).

If you think it worth the effort, you can add the line:

EXPORT\_SYMBOL (vector\_irq);

in the file:

arch/x86/kernel/irq.c

right after all the include lines.

After compiling and booting from the newly compiled kernel, change your linux driver as follows:

Add an include line:

#include <asm/hw\_irq.h>

And change the read function to:

/\* Driver function that needs to generate a software based interrupt on x86 based machines. This code base won’t work on ARM core based processors \*/

static ssize\_t driver\_read(struct file \*filp,

char \_\_user \*buf,

size\_t len,

loff\_t \*off)

{

struct irq\_desc \*desc;

printk(KERN\_INFO "Read function\n");

desc = irq\_to\_desc(11);

if (!desc) return -EINVAL;

\_\_this\_cpu\_write(vector\_irq[59], desc);

asm("int $0x3B"); // Corresponding to irq 11 (on Intel based cores)

return 0;

}

# Further Reference(s)

|  |
| --- |
|  |
| How to compile and install Linux Kernel from source code  <https://www.cyberciti.biz/tips/compiling-linux-kernel-26.html> |
|  |
| Ubuntu Mainline Builds - WiKi Page  <https://wiki.ubuntu.com/Kernel/MainlineBuilds> |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

# Legends

|  |  |
| --- | --- |
|  |  |
| Rpi | Raspberry Pi |
| SoC | System on Chip |
| VC | Video Core |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |