Experiment No. - 0

Aim: Making your own Linux Based Pendrive

Prerequisites

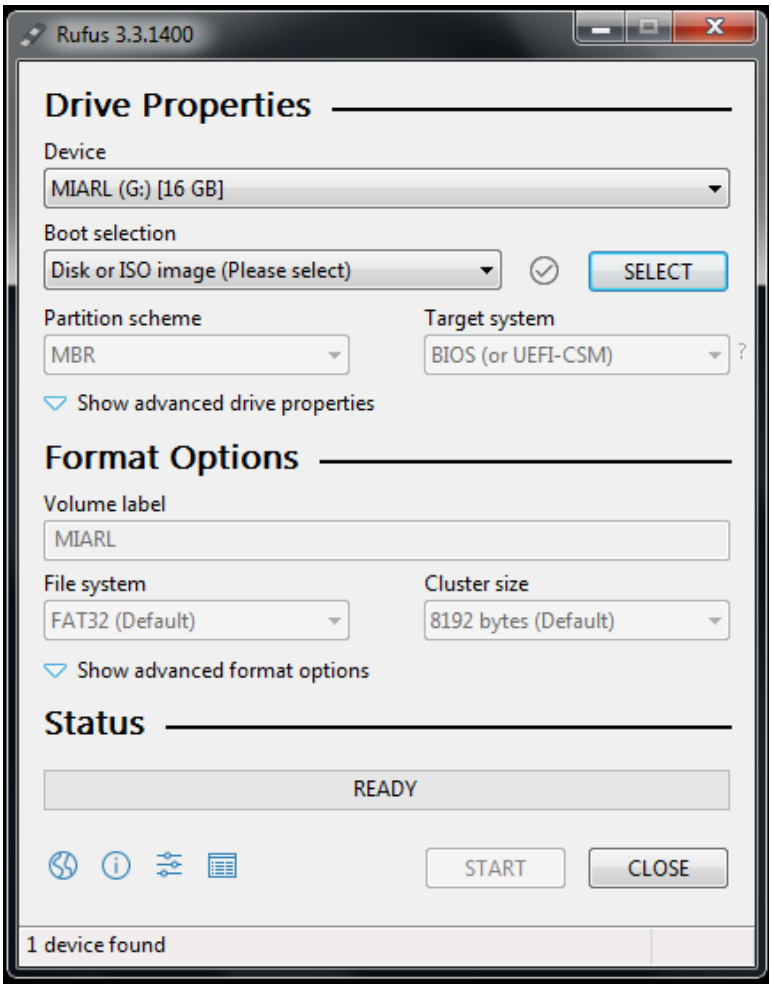
1) Fresh Pendrive 2 Nos. – One is 32 GB & another is 16 GB
   Note : we will call 32 GB Pendrive as Linux Ubuntu Pendrive and 16GB Pendrive as Linux bootable Pendrive.
2) Ubuntu-16.04.2.iso file (Image File)
3) Rufus 3.3 Software
4) SD Card Formatter Software

Step – 1

Insert 16 GB Pendrive in Windows based desktop system

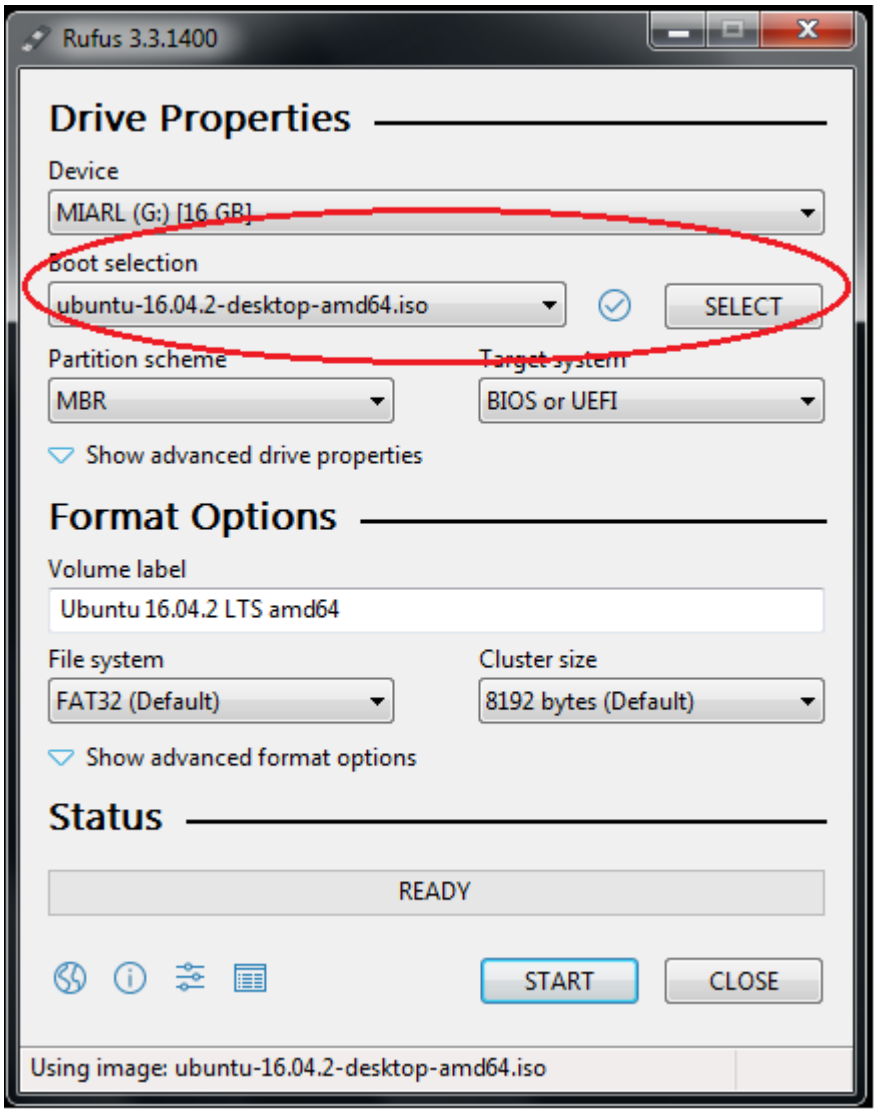
Step – 2

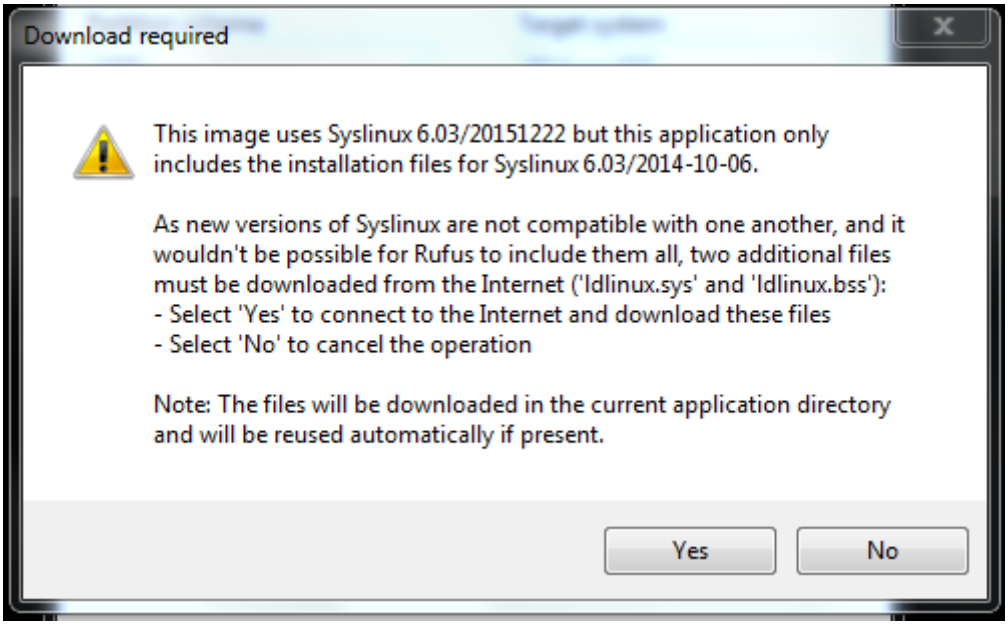
Open Rufus 3.3

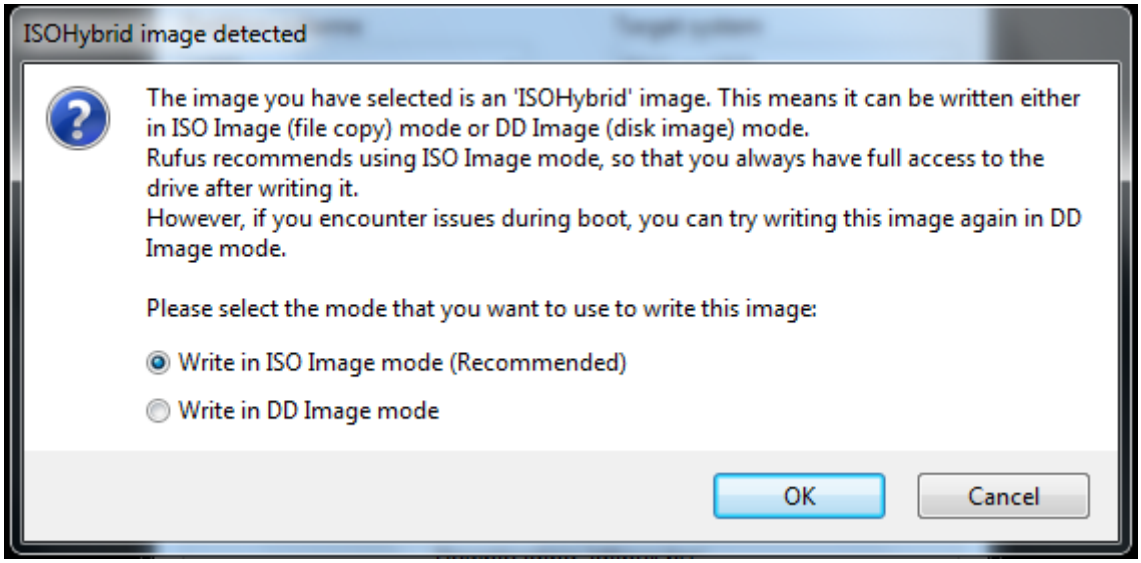


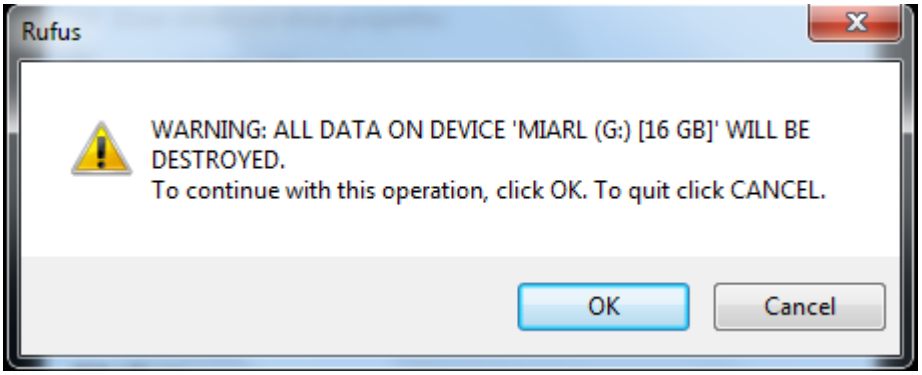
Step - 3

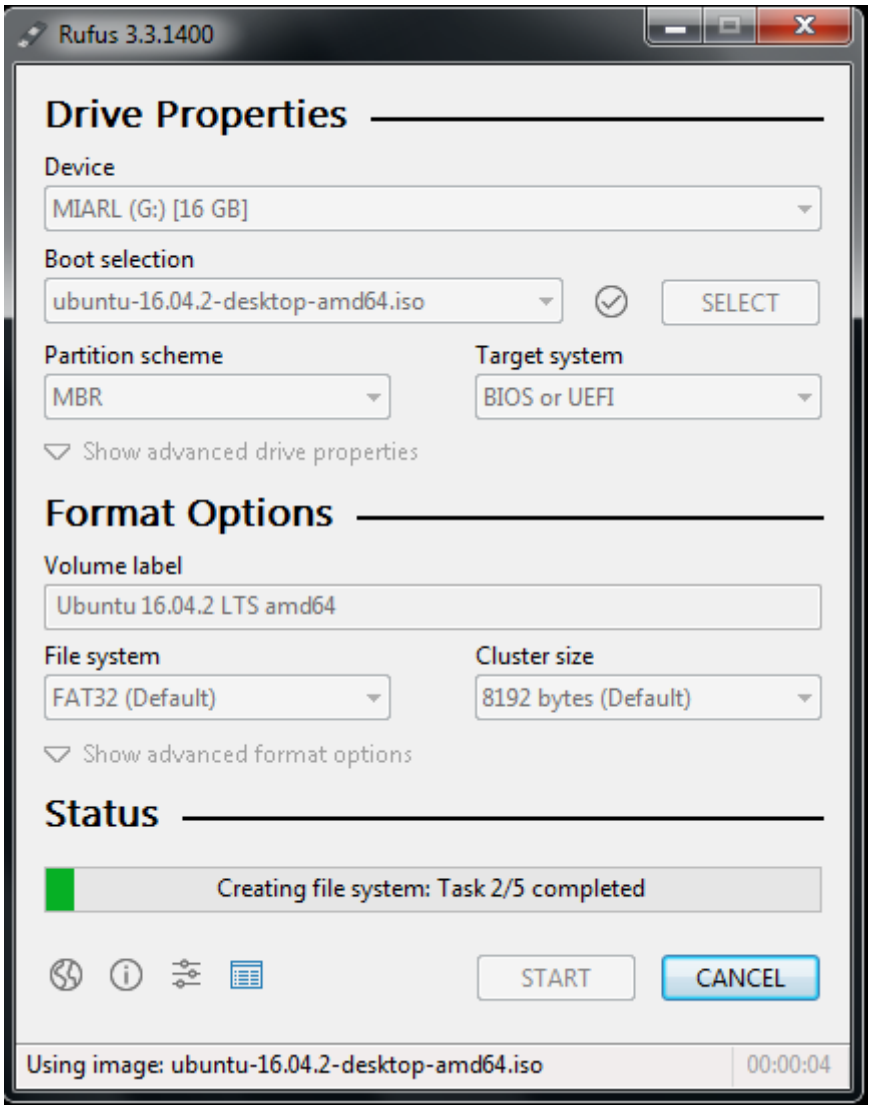
Click on the Select Button for Boot Selection and select the .iso file of Ubuntu 16.04 from the folder where you have kept it. Then Click on 'START'

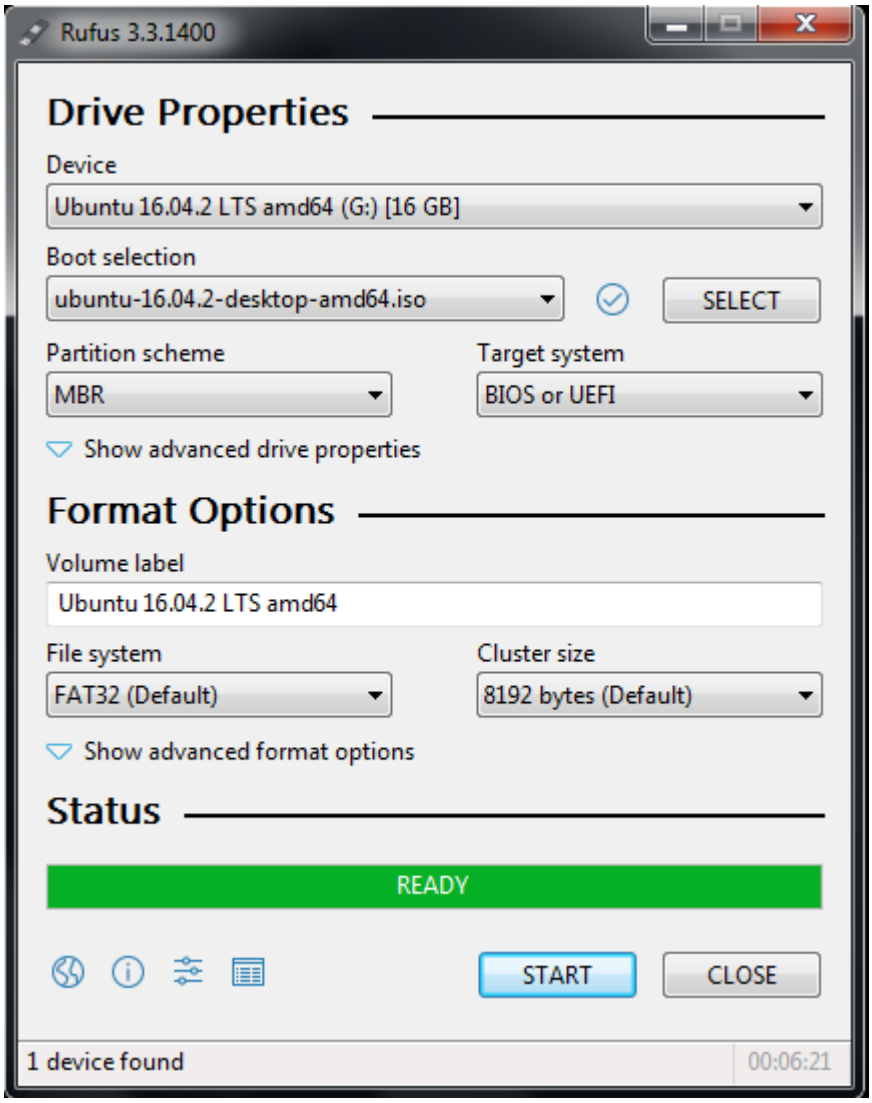

Rufus 3.3.1400
Drive Properties
Device
MIARL (G:) [16 GB]
Boot selection
ubuntu-16.04.2-desktop-amd64.iso
SELECT
Partition scheme
MBR
Target system
BIOS or UEFI
Show advanced drive properties
Format Options
Volume label
Ubuntu 16.04.2 LTS amd64
File system
FAT32 (Default)
Cluster size
8192 bytes (Default)
Show advanced format options
Status
READY
START
CLOSE
Using image: ubuntu-16.04.2-desktop-amd64.iso


Download required
This image uses Syslinux 6.03/20151222 but this application only includes the installation files for Syslinux 6.03/2014-10-06.
As new versions of Syslinux are not compatible with one another, and it wouldn't be possible for Rufus to include them all, two additional files must be downloaded from the Internet ('ldlinux.sys' and 'ldlinux.bss'):
- Select 'Yes' to connect to the Internet and download these files
- Select 'No' to cancel the operation
Note: The files will be downloaded in the current application directory and will be reused automatically if present.
Yes
No


ISOHybrid image detected
The image you have selected is an 'ISOHybrid' image. This means it can be written either in ISO Image (file copy) mode or DD Image (disk image) mode.
Rufus recommends using ISO Image mode, so that you always have full access to the drive after writing it.
However, if you encounter issues during boot, you can try writing this image again in DD Image mode.
Please select the mode that you want to use to write this image:
Write in ISO Image mode (Recommended)
Write in DD Image mode
OK
Cancel

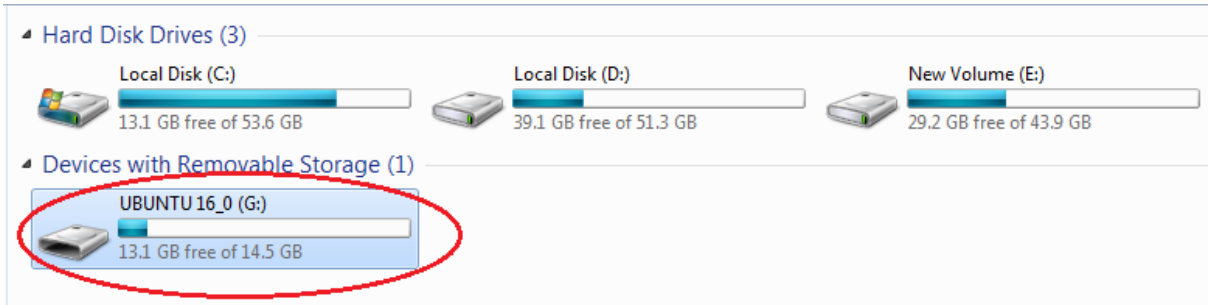

Rufus
WARNING: ALL DATA ON DEVICE 'MIARL (G:) [16 GB]' WILL BE DESTROYED.
To continue with this operation, click OK. To quit click CANCEL.
OK
Cancel


Rufus 3.3.1400
Drive Properties
Device
MIARL (G:) [16 GB]
Boot selection
ubuntu-16.04.2-desktop-amd64.iso
SELECT
Partition scheme
MBR
Target system
BIOS or UEFI
Show advanced drive properties
Format Options
Volume label
Ubuntu 16.04.2 LTS amd64
File system
FAT32 (Default)
Cluster size
8192 bytes (Default)
Show advanced format options
Status
Creating file system: Task 2/5 completed
START
CANCEL
Using image: ubuntu-16.04.2-desktop-amd64.iso
00:00:04



Step -4

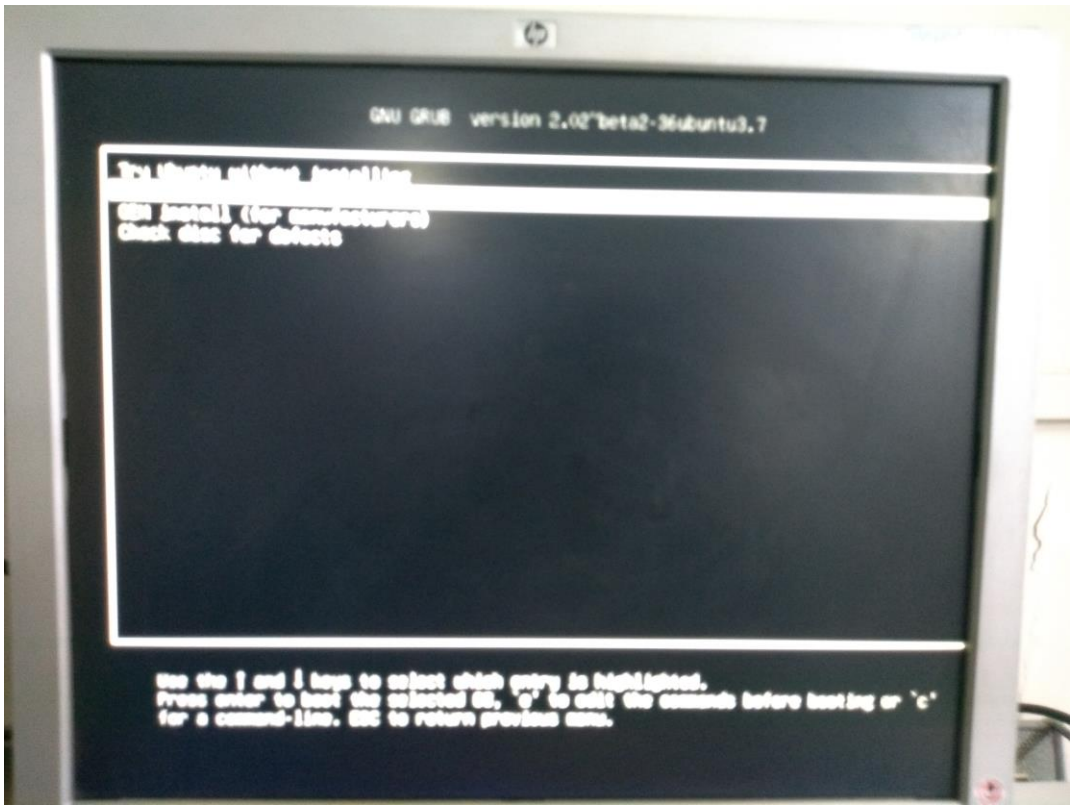
Check the Pendrive. You will identify boot folders in it. Your bootable pendrive for Ubuntu 16.04 is ready.



Name	Date modified	Type	Size
autorun	12/5/2018 10:32 AM	Icon	34 KB
autorun	12/5/2018 10:32 AM	Setup Information	1 KB
README.diskdefines	12/5/2018 10:32 AM	DISKDEFINES File	1 KB
syslinux.cfg	12/5/2018 10:32 AM	CFG File	1 KB
ubuntu	12/5/2018 10:32 AM	File	0 KB
md5sum	12/5/2018 10:32 AM	Text Document	21 KB
preseed	12/5/2018 10:32 AM	File folder	
pics	12/5/2018 10:32 AM	File folder	
pool	12/5/2018 10:32 AM	File folder	
EFI	12/5/2018 10:32 AM	File folder	
install	12/5/2018 10:32 AM	File folder	
isolinux	12/5/2018 10:32 AM	File folder	
dists	12/5/2018 10:32 AM	File folder	
casper	12/5/2018 10:26 AM	File folder	
.disk	12/5/2018 10:26 AM	File folder	
boot	12/5/2018 10:26 AM	File folder	

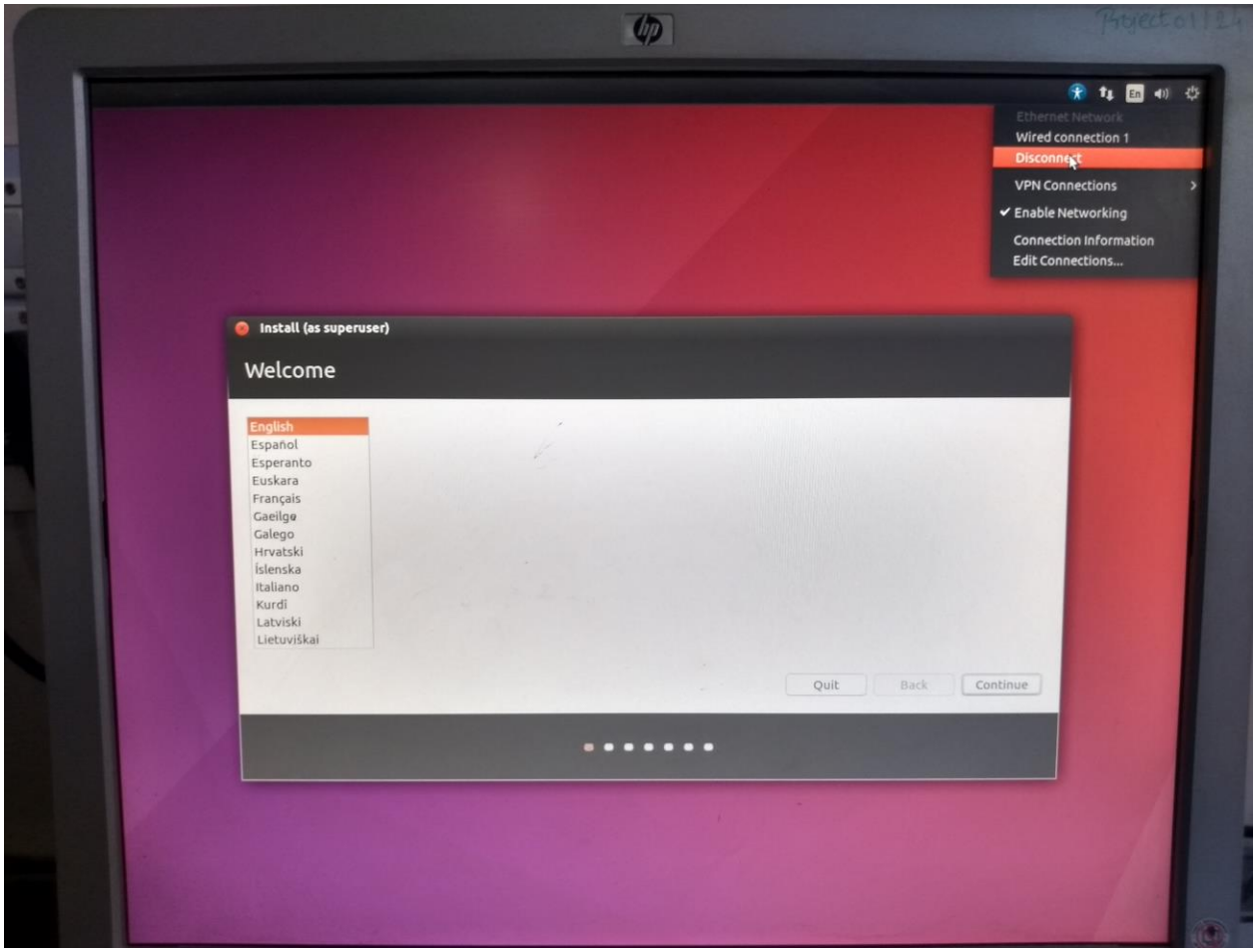
Step – 5

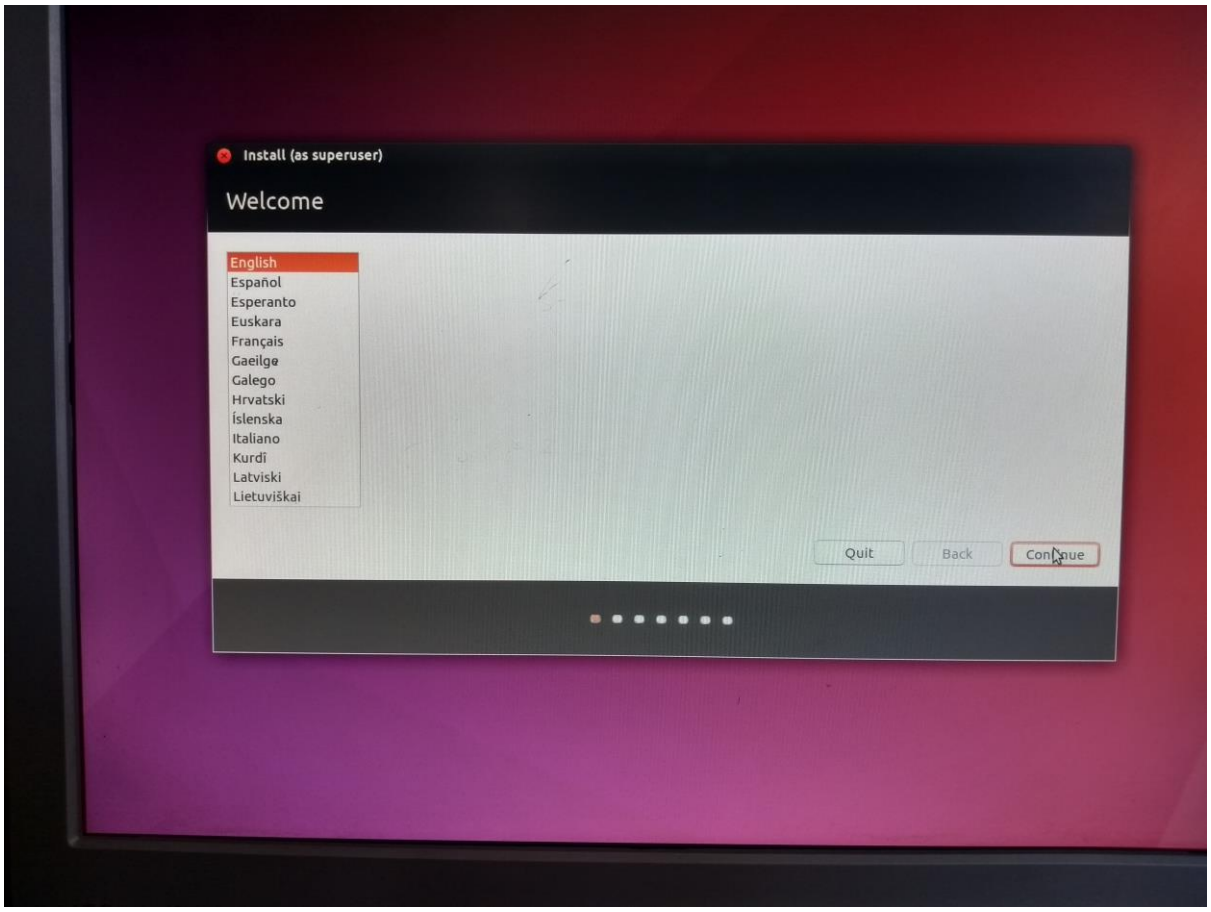
In some of the cases system will not show the message as “Boot UBuntu from USB”, in that case change the priority of device from Boot Menu (F9 or F10 Shortcut keys)



Step – 6

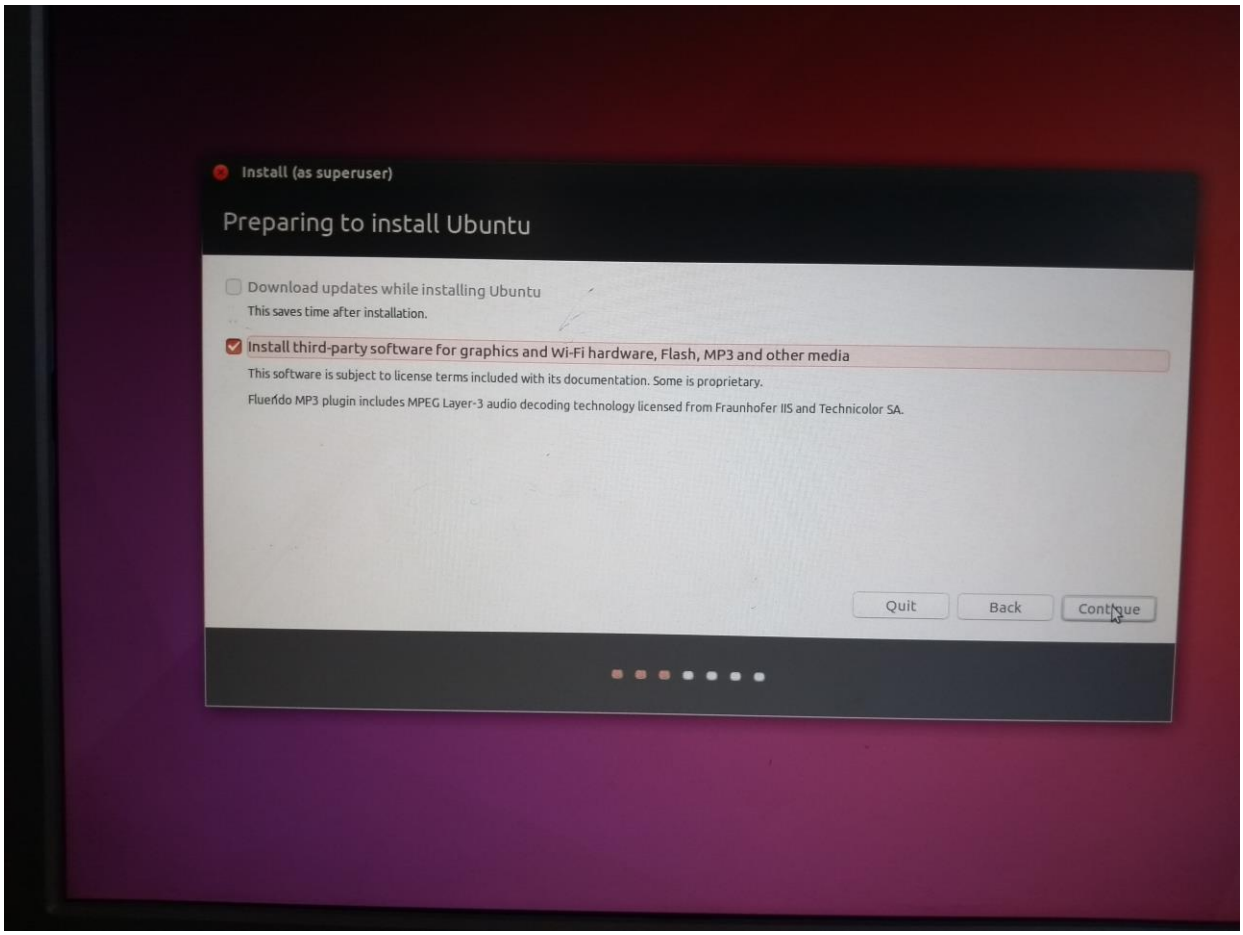
When Ubuntu 16.04 windows will open, disconnect your system from Internet first and click on 'Continue'.




Install (as superuser)
Welcome
English
Español
Esperanto
Euskara
Français
Gaeilge
Galego
Hrvatski
Íslenska
Italiano
Kurdî
Latviski
Lietuviškai
Quit
Back
Continue

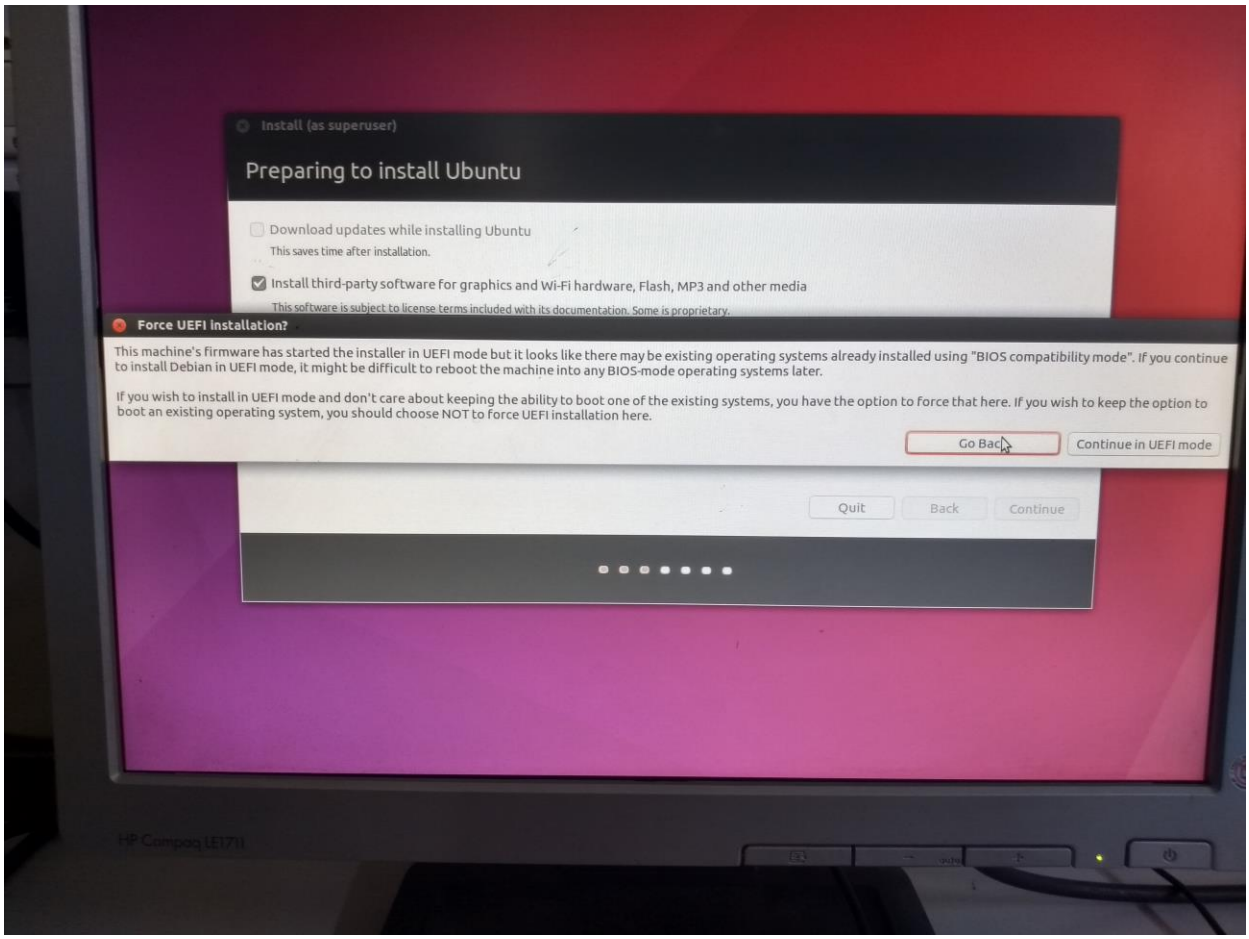
Step -7

Click on ‘install third party’ software



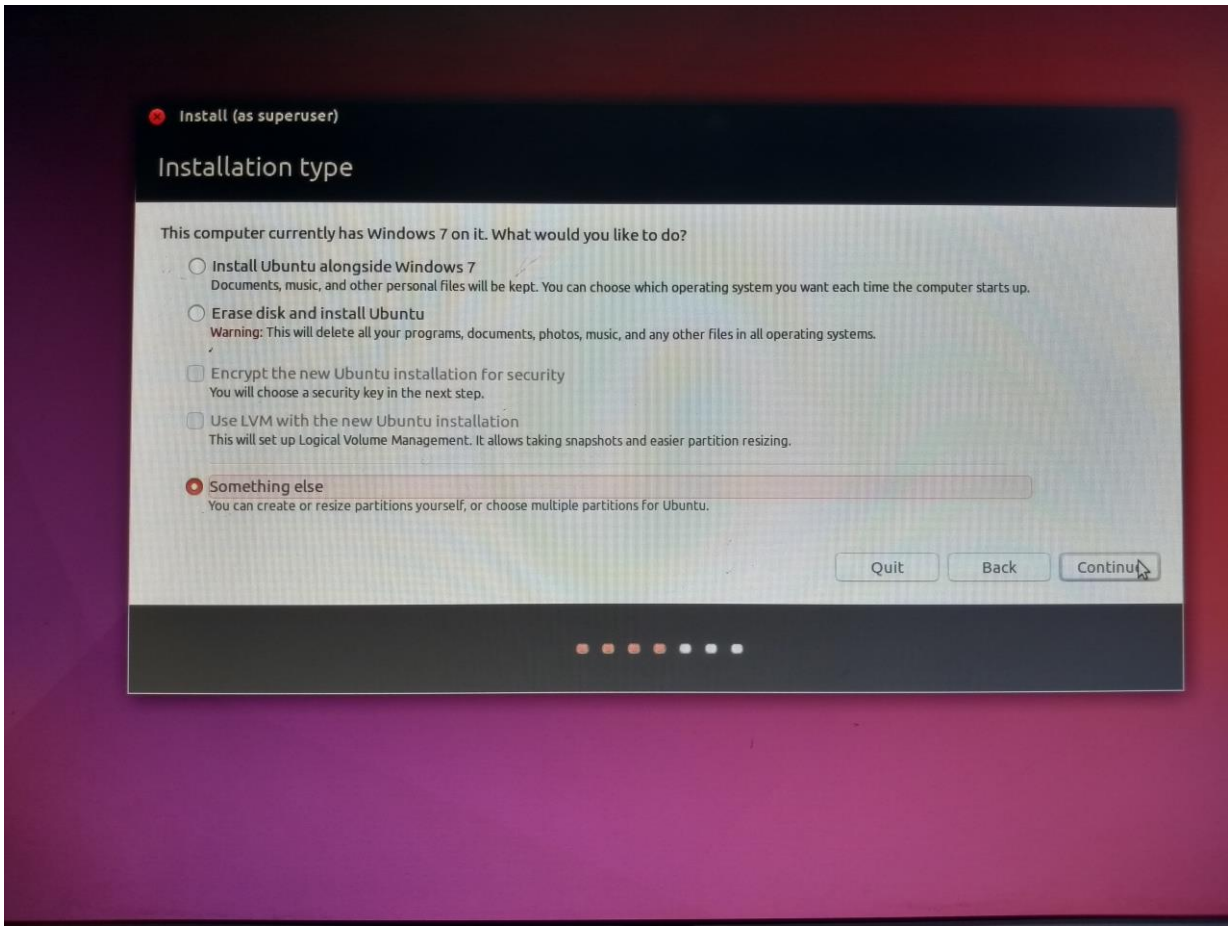
Step – 8

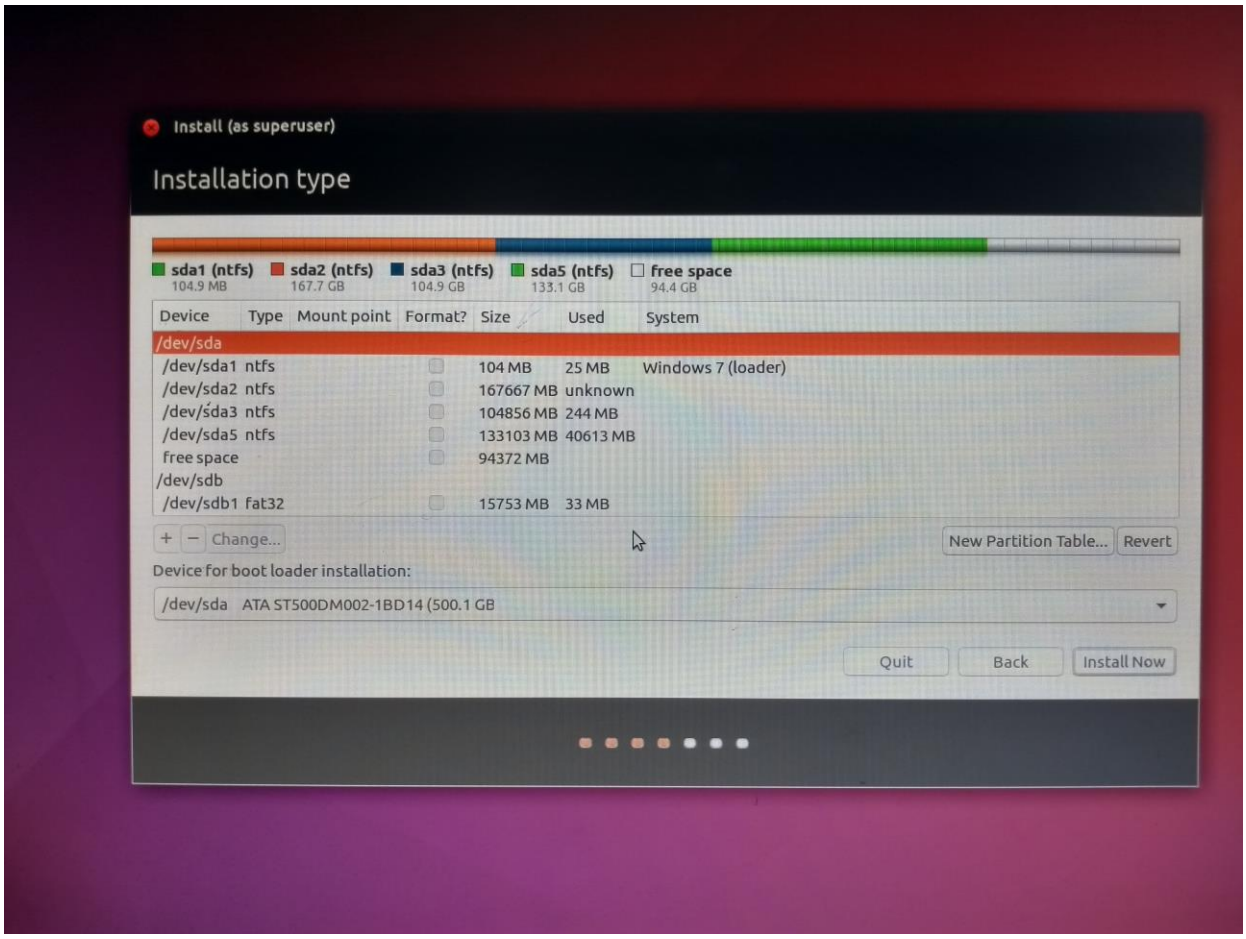
Click on ‘Go Back’



Step – 9

Click on “Something Else”




Install (as superuser)
Installation type
sda1 (ntfs) 104.9 MB
sda2 (ntfs) 167.7 GB
sda3 (ntfs) 104.9 GB
sda5 (ntfs) 133.1 GB
free space 94.4 GB
Device Type Mount point Format? Size Used System
/dev/sda
/dev/sda1 ntfs 104 MB 25 MB Windows 7 (loader)
/dev/sda2 ntfs 167667 MB unknown
/dev/sda3 ntfs 104856 MB 244 MB
/dev/sda5 ntfs 133103 MB 40613 MB
free space 94372 MB
/dev/sdb
/dev/sdb1 fat32 15753 MB 33 MB
+ − Change...
New Partition Table... Revert
Device for boot loader installation:
/dev/sda ATA ST500DM002-1BD14 (500.1 GB
Quit Back Install Now

Step -10

Select Pendrive on which you want to install Linux. Do not select the Hard Drive of the system.

Create the partition on your Linux pendrive by clicking on “+” as follows.

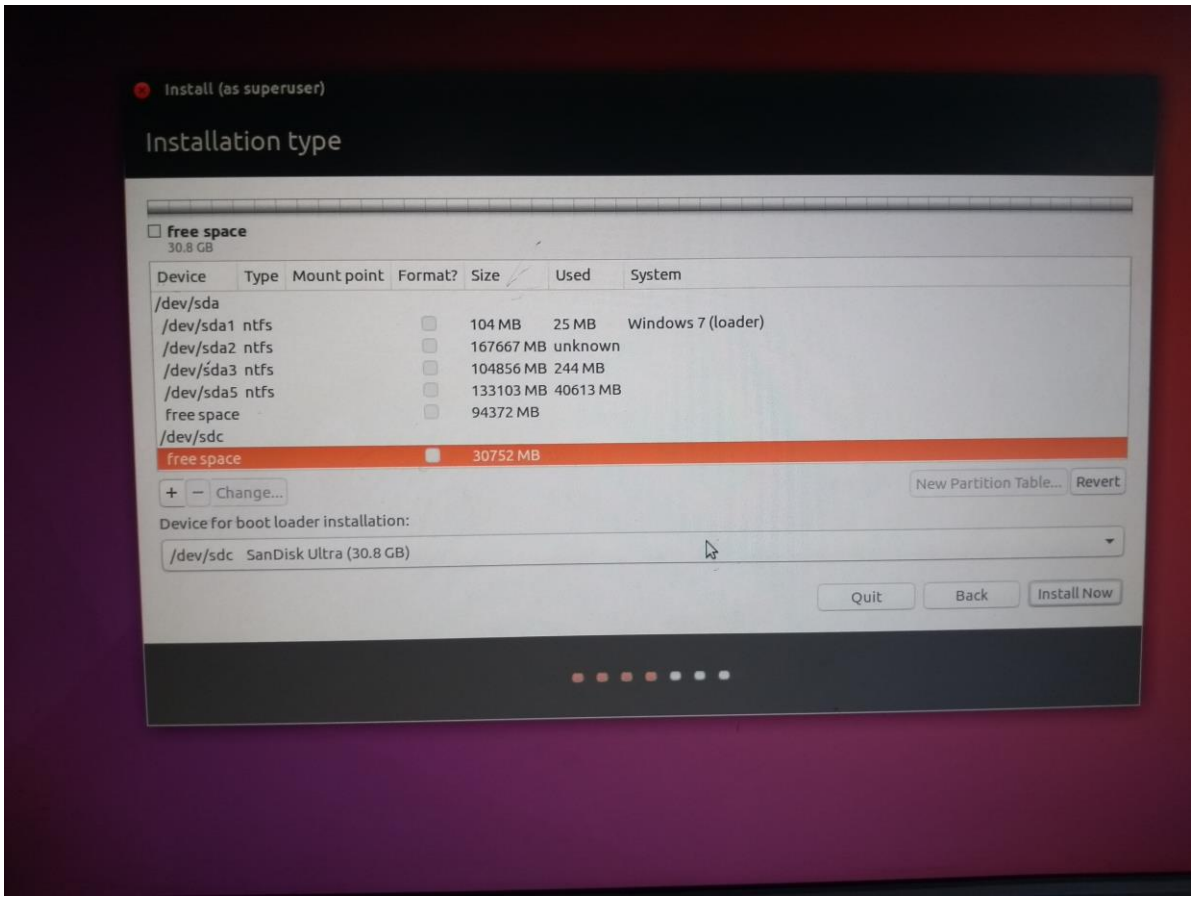
Root (/) - 15000 MB

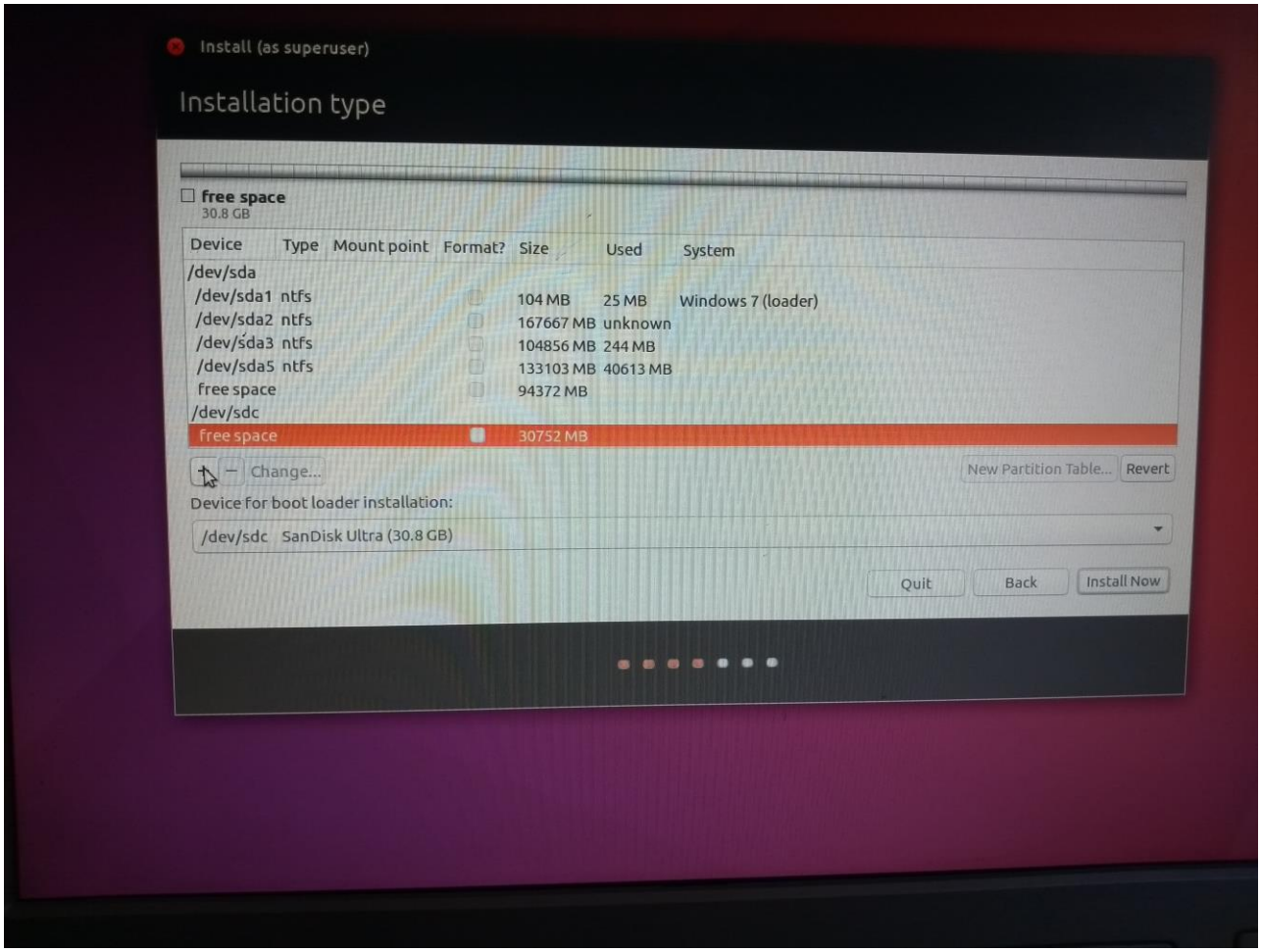
Boot (/boot) - 1024 MB

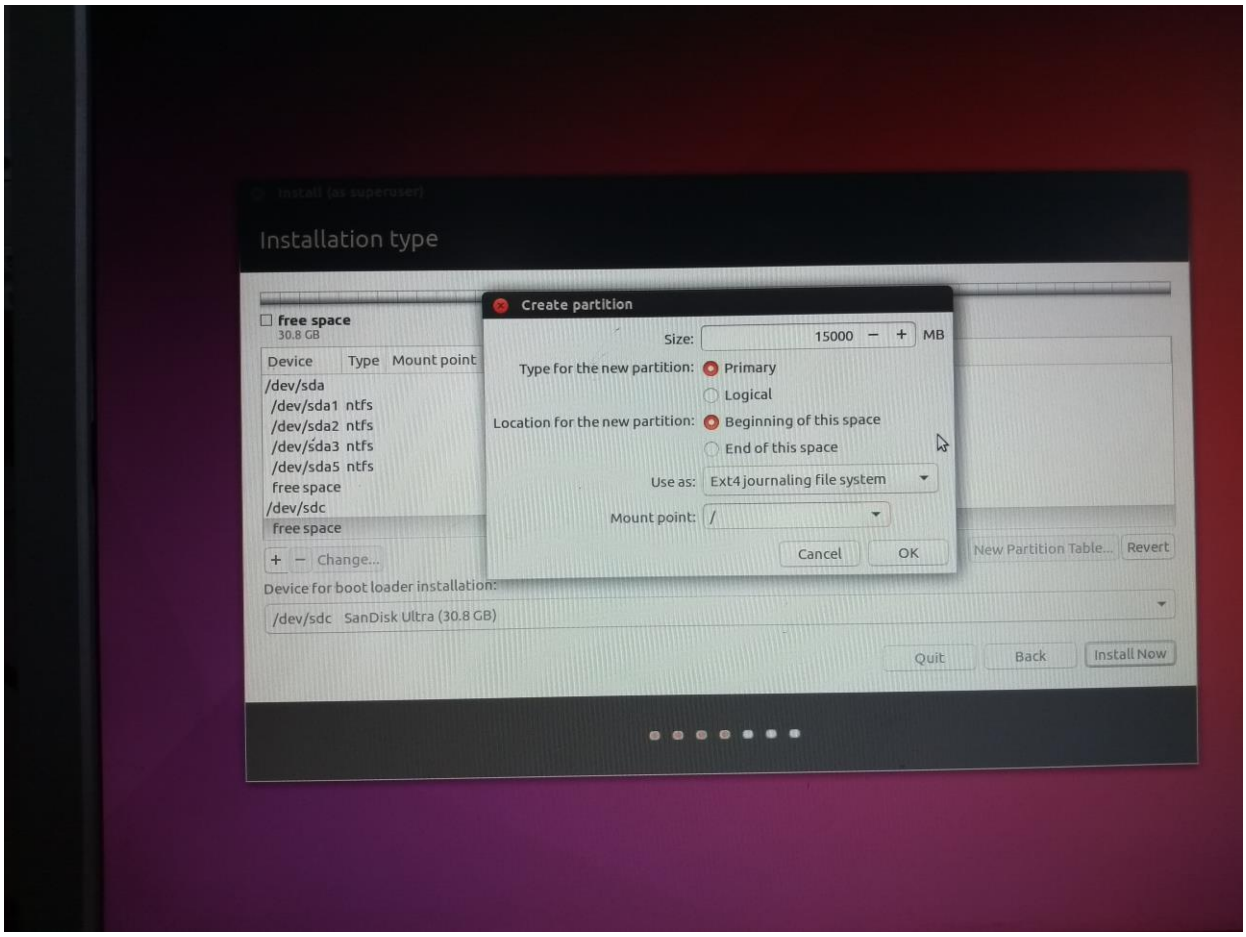
Temp (/temp) - 2048 MB

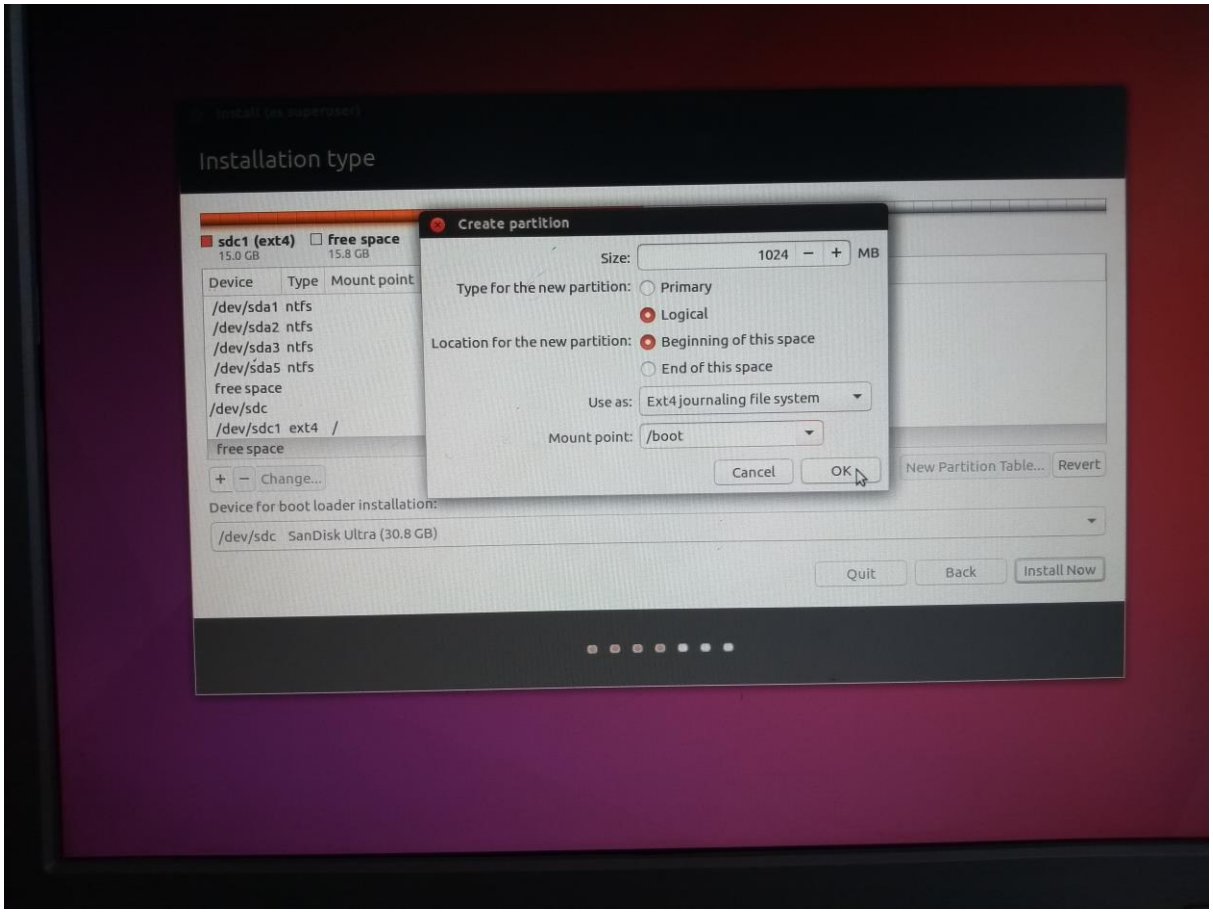
Home (/home) - 7832 MB

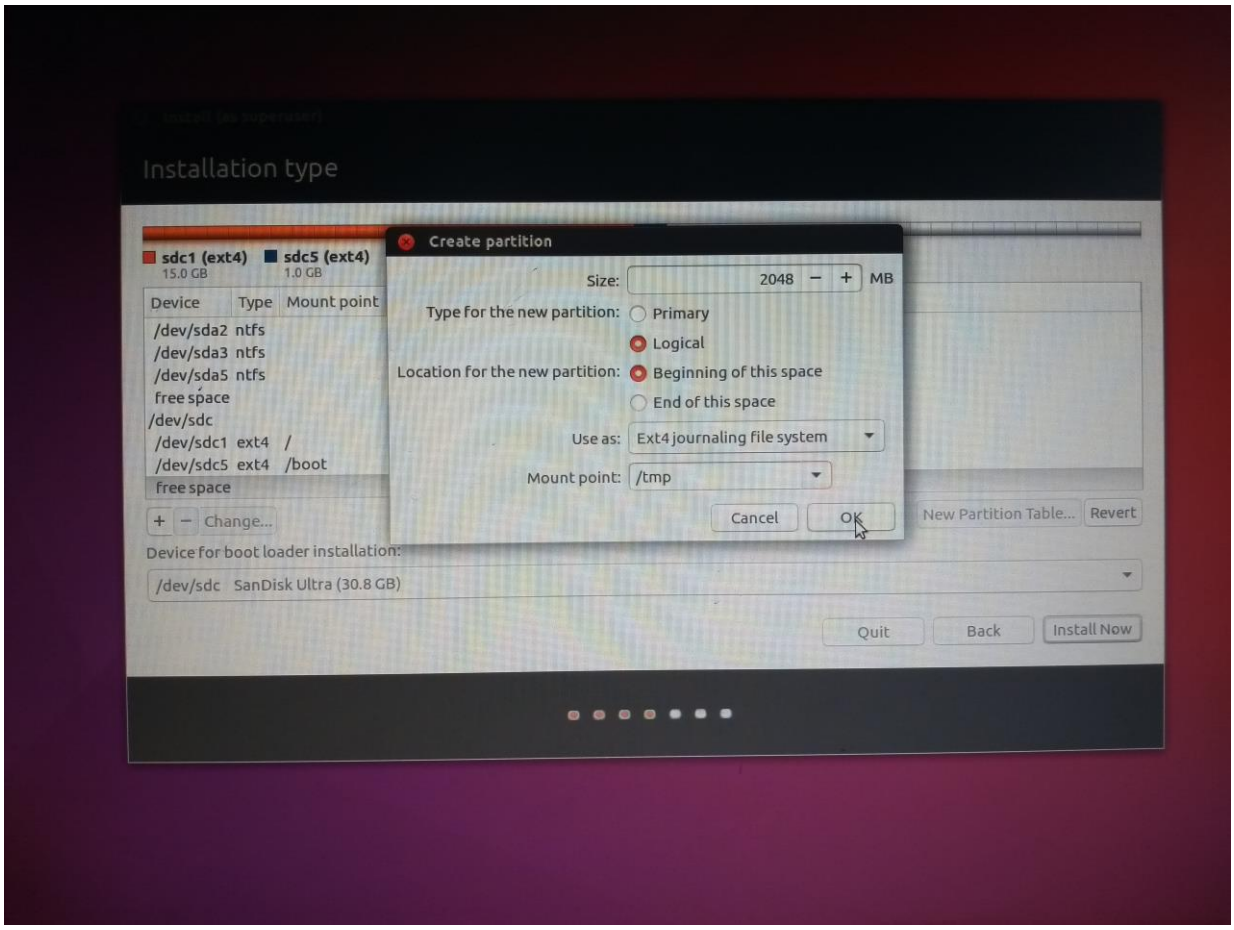
swap area – Reaming MBs

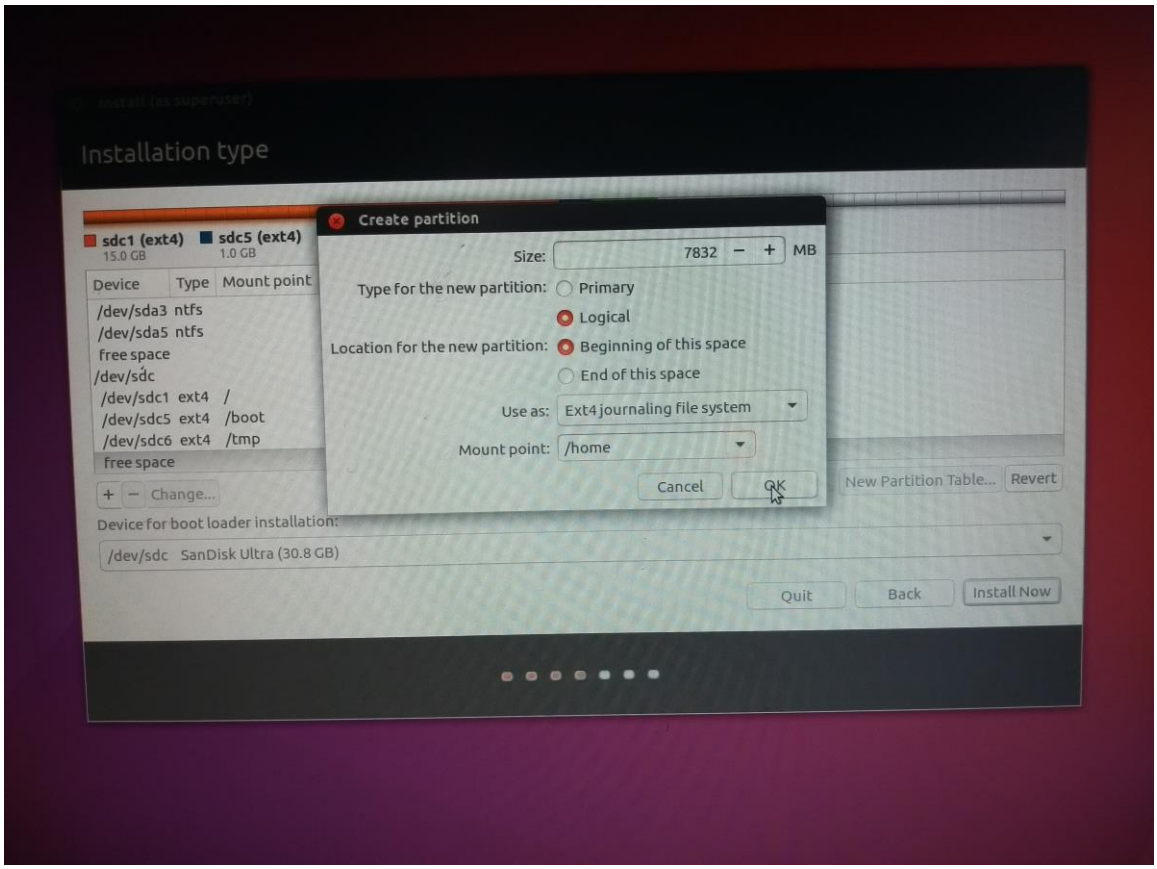


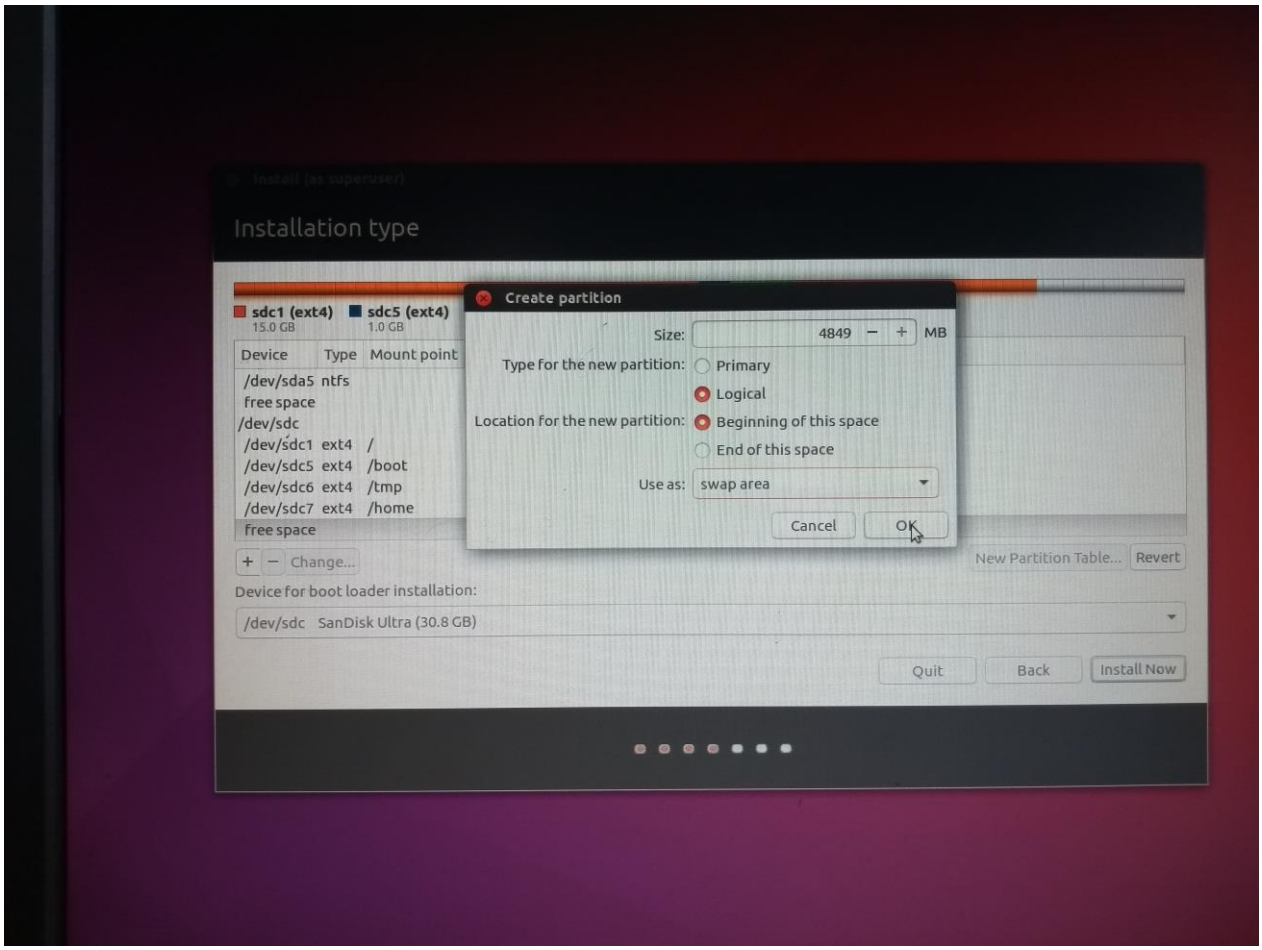

Install (as superuser)
Installation type
free space
30.8 GB
Device Type Mount point Format? Size Used System
/dev/sda
/dev/sda1 ntfs 104 MB 25 MB Windows 7 (loader)
/dev/sda2 ntfs 167667 MB unknown
/dev/sda3 ntfs 104856 MB 244 MB
/dev/sda5 ntfs 133103 MB 40613 MB
free space 94372 MB
/dev/sdc
free space 30752 MB
Change...
New Partition Table...
Revert
Device for boot loader installation:
/dev/sdc SanDisk Ultra (30.8 GB)
Quit
Back
Install Now


Installation type
free space
30.8 GB
Device
Type
Mount point
/dev/sda
/dev/sda1 ntfs
/dev/sda2 ntfs
/dev/sda3 ntfs
/dev/sda5 ntfs
free space
/dev/sdc
free space
Change...
Create partition
Size:
15000
MB
Type for the new partition:
Primary
Logical
Location for the new partition:
Beginning of this space
End of this space
Use as:
Ext4 journaling file system
Mount point:
/
Cancel
OK
New Partition Table...
Revert
Device for boot loader installation:
/dev/sdc SanDisk Ultra (30.8 GB)
Quit
Back
Install Now


Installation type
sdc1 (ext4)
15.0 GB
free space
15.8 GB
Device
Type
Mount point
/dev/sda1 ntfs
/dev/sda2 ntfs
/dev/sda3 ntfs
/dev/sda5 ntfs
free space
/dev/sdc
/dev/sdc1 ext4 /
free space
Change...
Device for boot loader installation:
/dev/sdc SanDisk Ultra (30.8 GB)
Create partition
Size: 1024 MB
Type for the new partition: Primary
Logical
Location for the new partition: Beginning of this space
End of this space
Use as: Ext4 journaling file system
Mount point: /boot
Cancel
OK
New Partition Table...
Revert
Quit
Back
Install Now

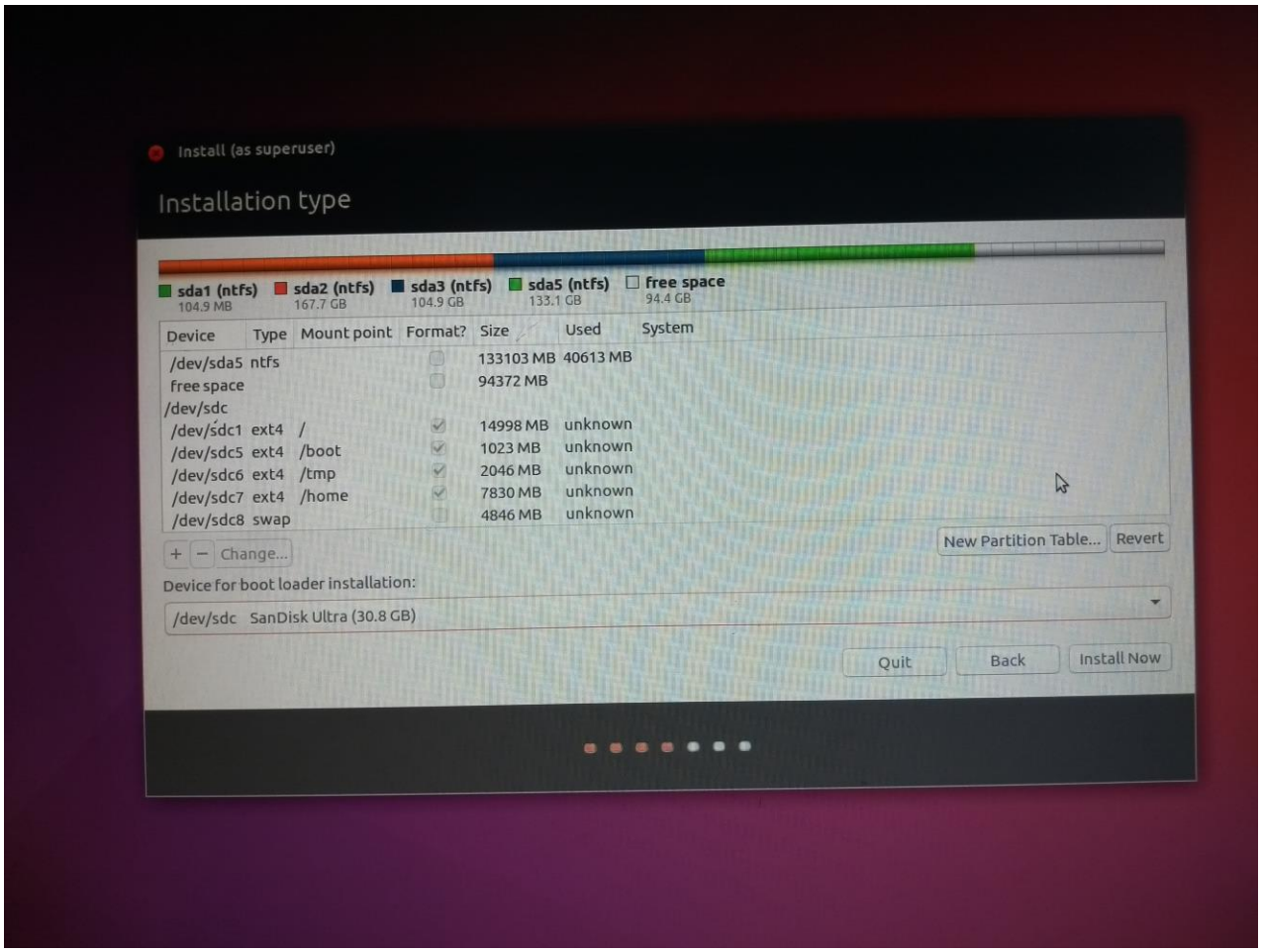

Installation type
sdc1 (ext4)
15.0 GB
sdc5 (ext4)
1.0 GB
Device
Type
Mount point
/dev/sda2 ntfs
/dev/sda3 ntfs
/dev/sda5 ntfs
free space
/dev/sdc
/dev/sdc1 ext4 /
/dev/sdc5 ext4 /boot
free space
Change...
Create partition
Size: 2048 MB
Type for the new partition: Primary
Logical
Location for the new partition: Beginning of this space
End of this space
Use as: Ext4 journaling file system
Mount point: /tmp
Cancel
OK
New Partition Table...
Revert
Device for boot loader installation:
/dev/sdc SanDisk Ultra (30.8 GB)
Quit
Back
Install Now


Installation type
sdc1 (ext4)
15.0 GB
sdc5 (ext4)
1.0 GB
Device
Type
Mount point
/dev/sda3 ntfs
/dev/sda5 ntfs
free space
/dev/sdc
/dev/sdc1 ext4 /
/dev/sdc5 ext4 /boot
/dev/sdc6 ext4 /tmp
free space
Change...
Create partition
Size:
7832
MB
Type for the new partition:
Primary
Logical
Location for the new partition:
Beginning of this space
End of this space
Use as:
Ext4 journaling file system
Mount point:
/home
Cancel
OK
New Partition Table...
Revert
Device for boot loader installation:
/dev/sdc SanDisk Ultra (30.8 GB)
Quit
Back
Install Now


Installation type
sdc1 (ext4)
15.0 GB
sdc5 (ext4)
1.0 GB
Device
Type
Mount point
/dev/sda5 ntfs
free space
/dev/sdc
/dev/sdc1 ext4 /
/dev/sdc5 ext4 /boot
/dev/sdc6 ext4 /tmp
/dev/sdc7 ext4 /home
free space
Change...
Create partition
Size:
4849
MB
Type for the new partition:
Primary
Logical
Location for the new partition:
Beginning of this space
End of this space
Use as:
swap area
Cancel
OK
New Partition Table...
Revert
Device for boot loader installation:
/dev/sdc SanDisk Ultra (30.8 GB)
Quit
Back
Install Now

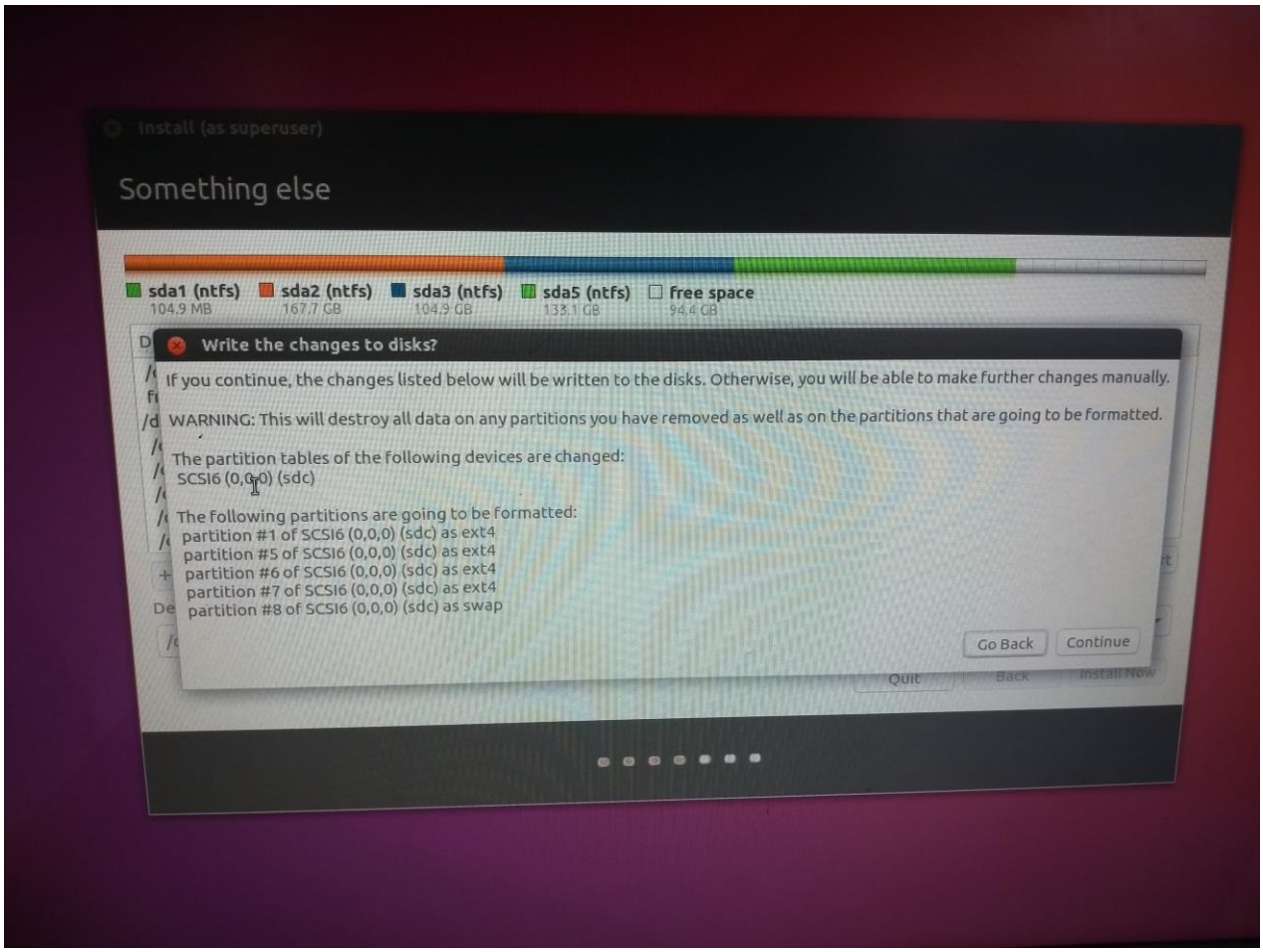
Step -11

After Competing the Partition Process, click on the “install Now”



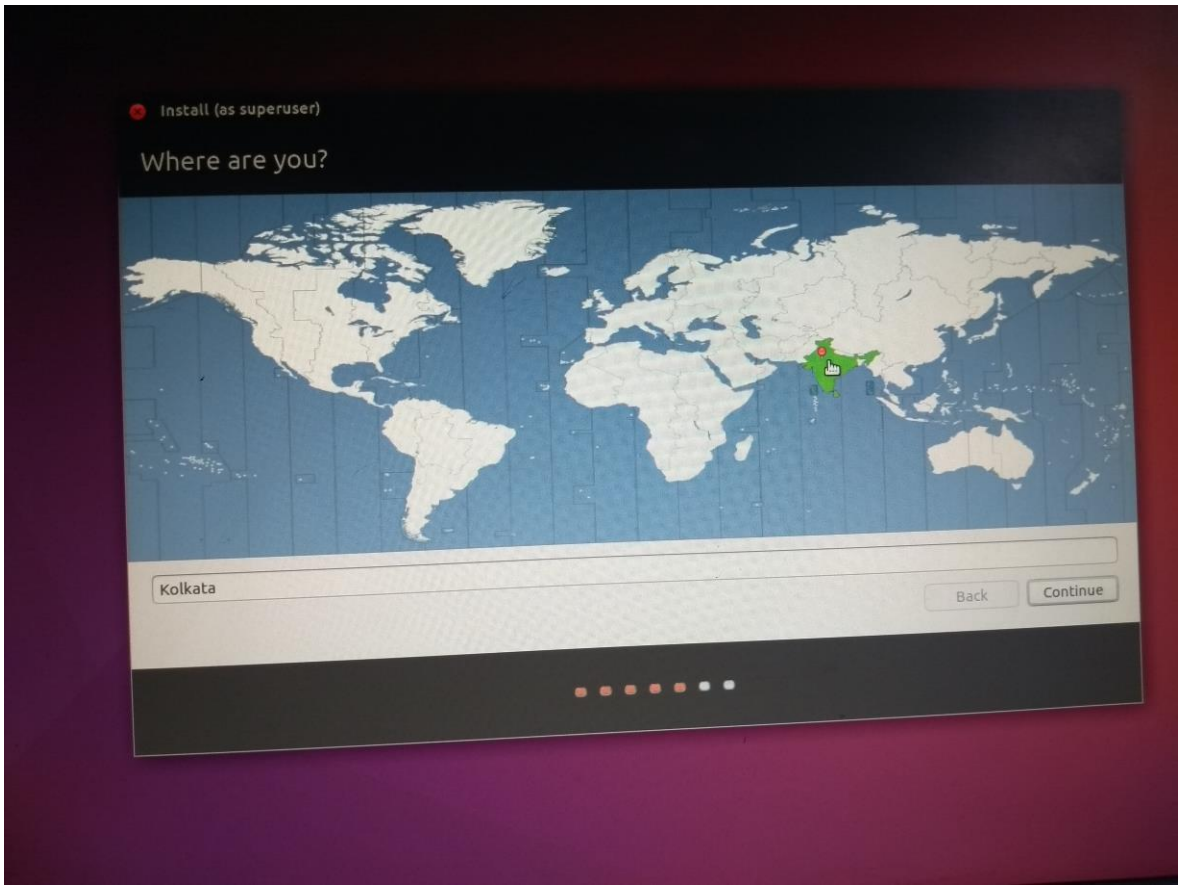
Step -12

Click on “Continue”



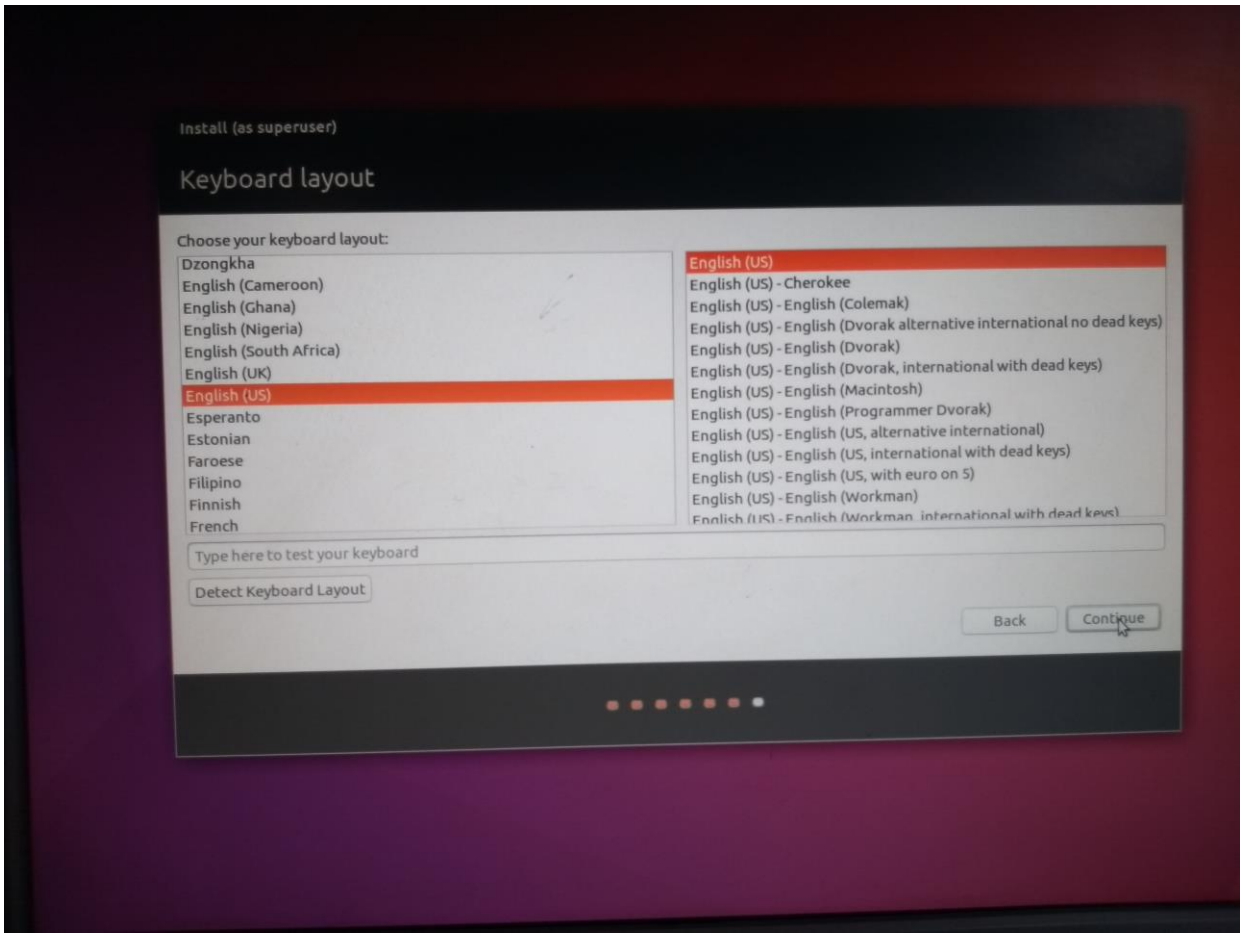
Step-13

Select region of Country INDIA and Click on Continue



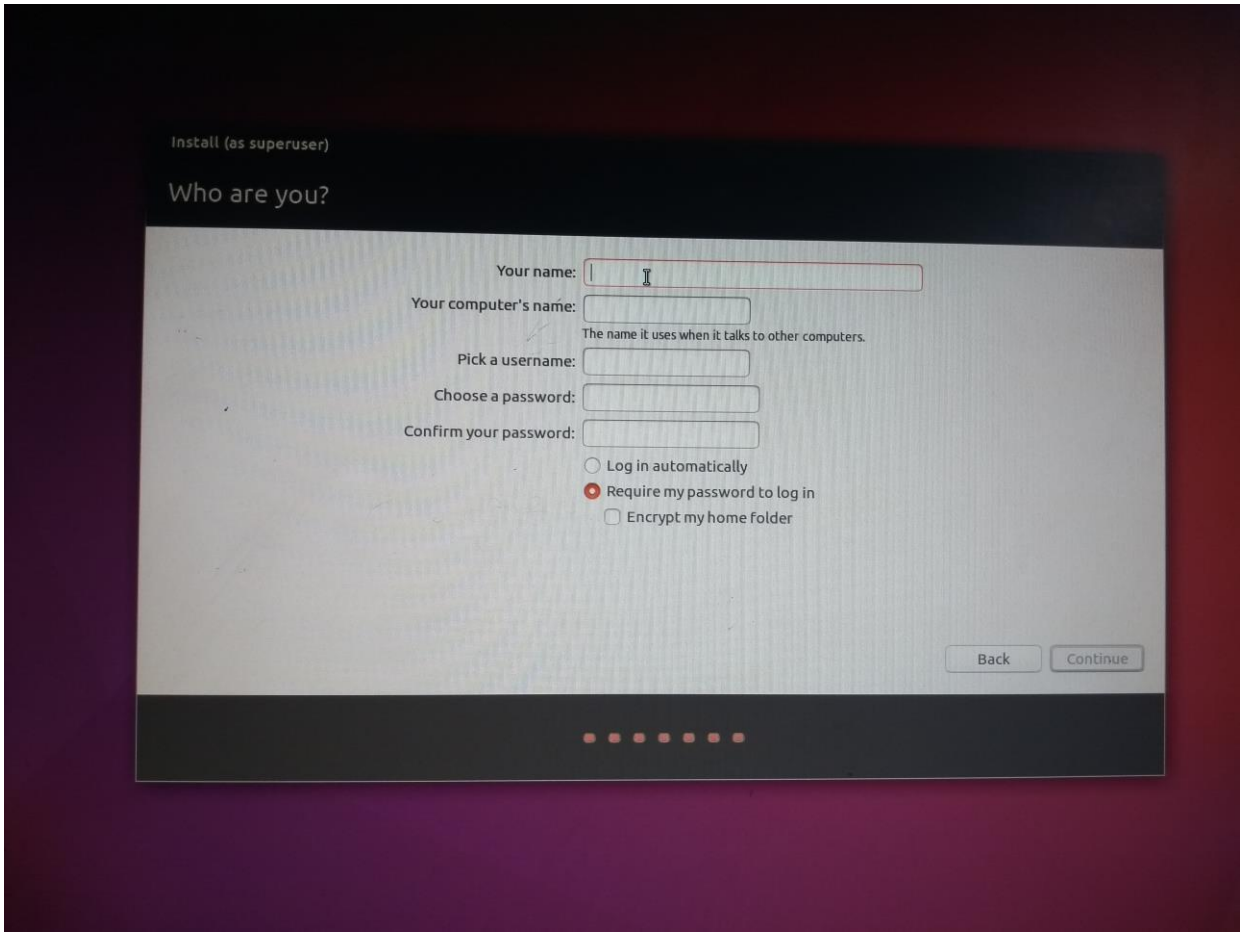
Step -14

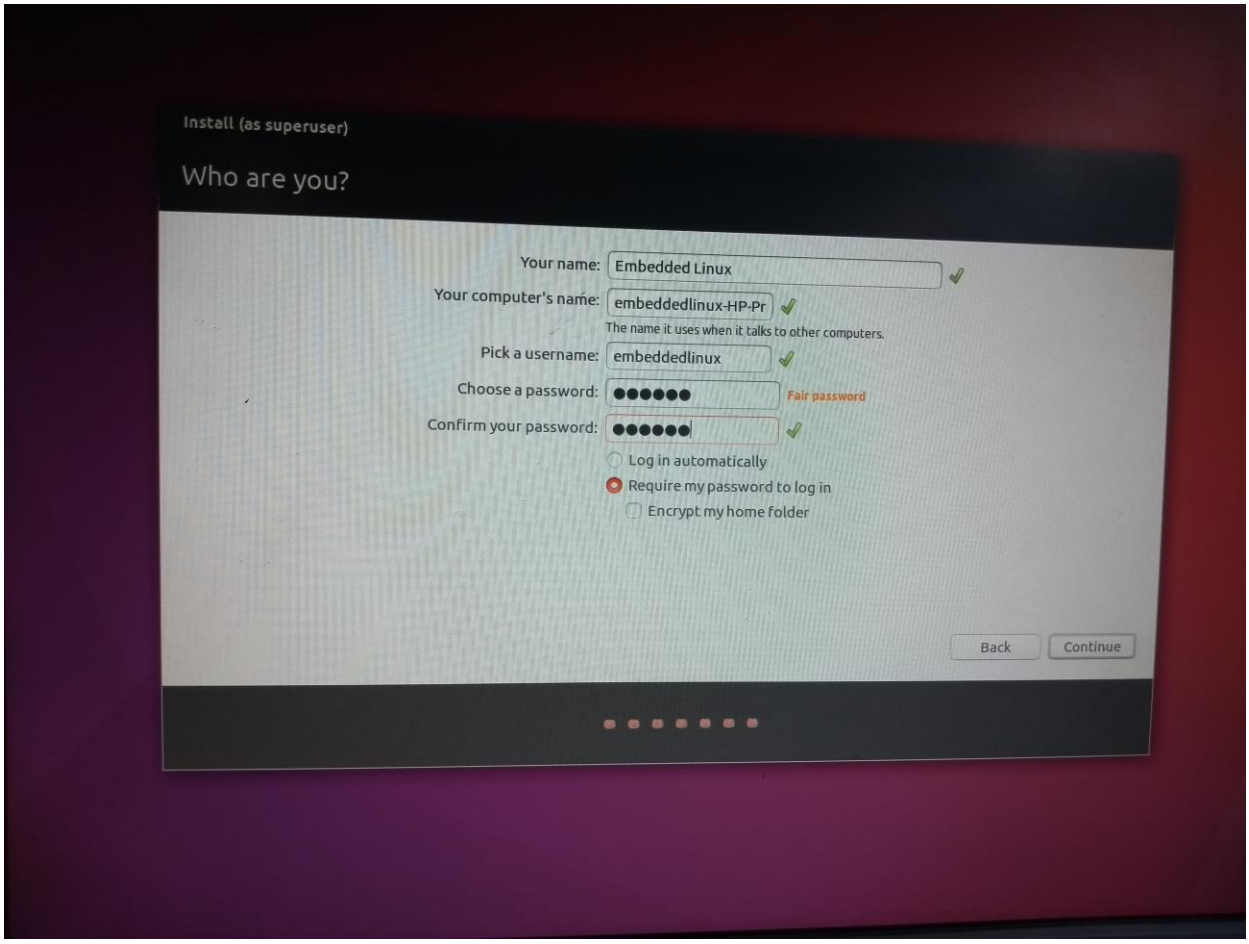
Select Keyboard Layout as “English US” and Click on “Continue”



Step -15

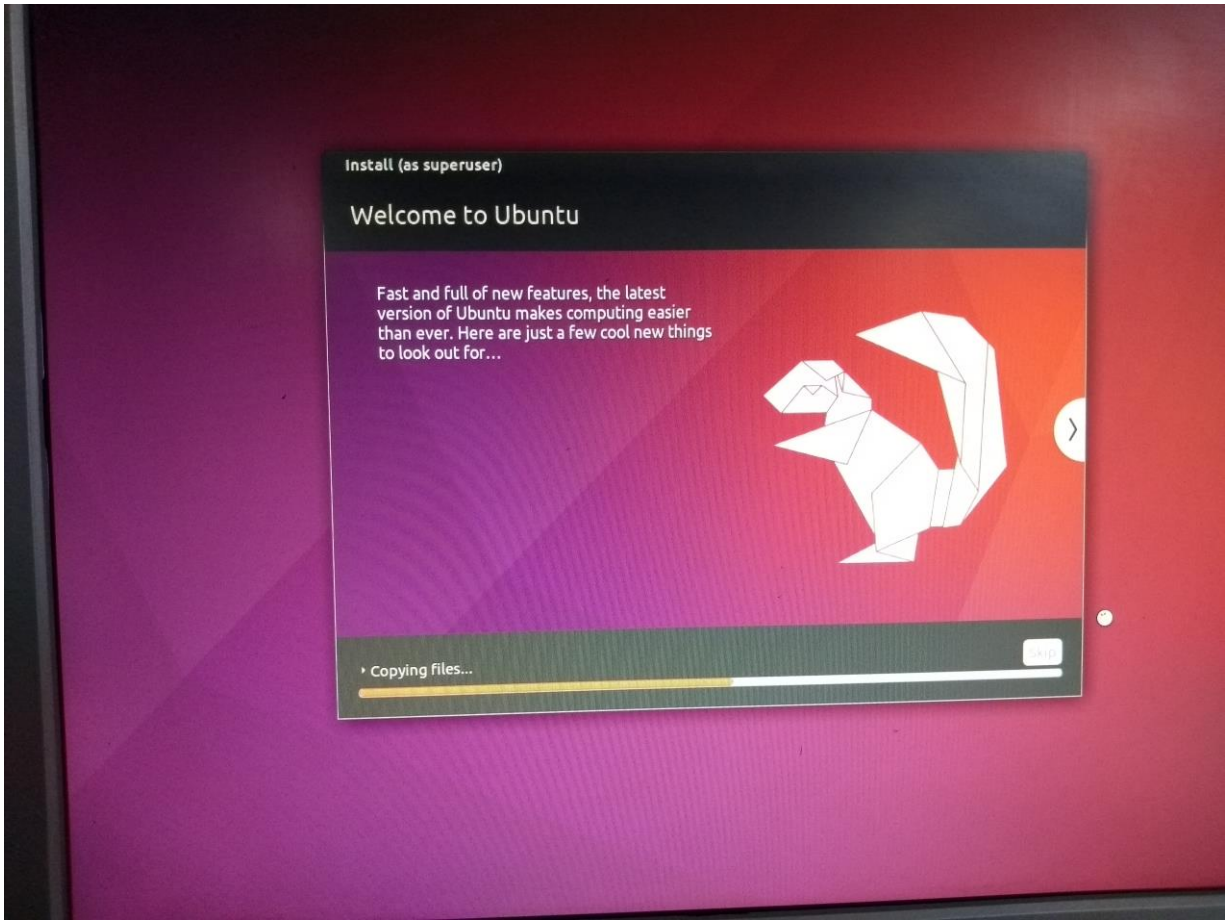
Give System Name, Username and Password and click on Continue




Install (as superuser)
Who are you?
Your name: Embedded Linux
Your computer's name: embeddedlinux-HP-Pr
The name it uses when it talks to other computers.
Pick a username: embeddedlinux
Choose a password:
Fair password
Confirm your password:
Log in automatically
Require my password to log in
Encrypt my home folder
Back
Continue

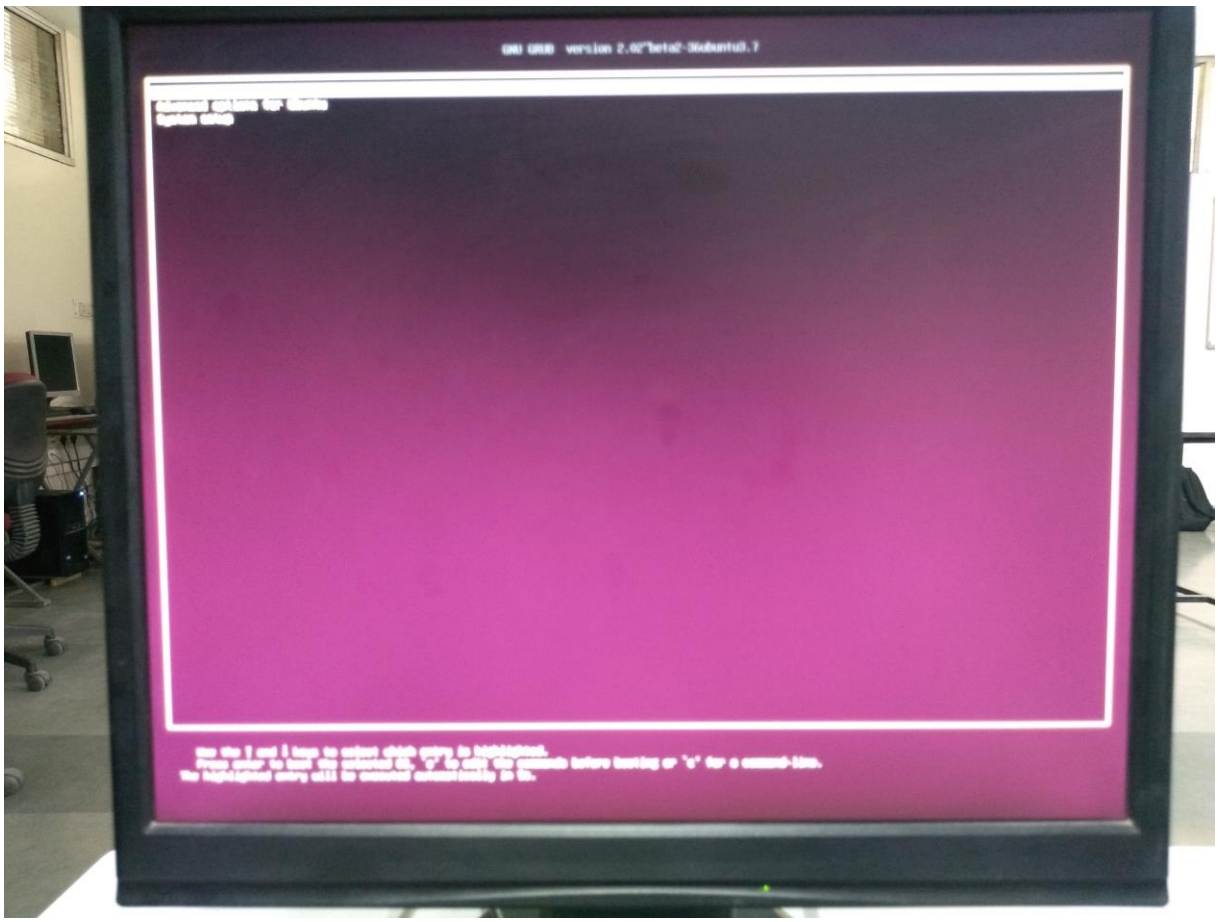
Step -16

Now Seat back and relax. Linux is going to install in your pendrive.



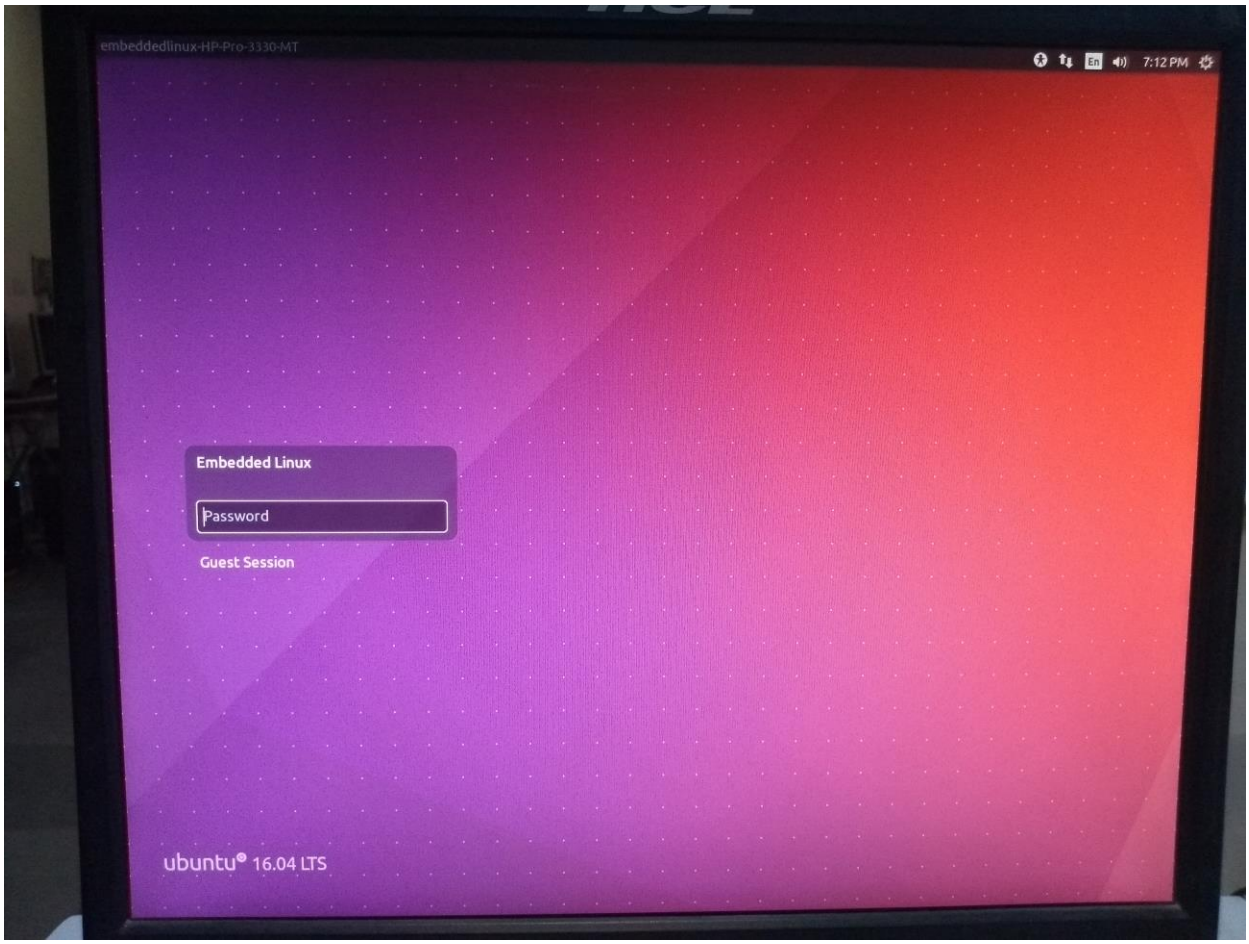
Step -17

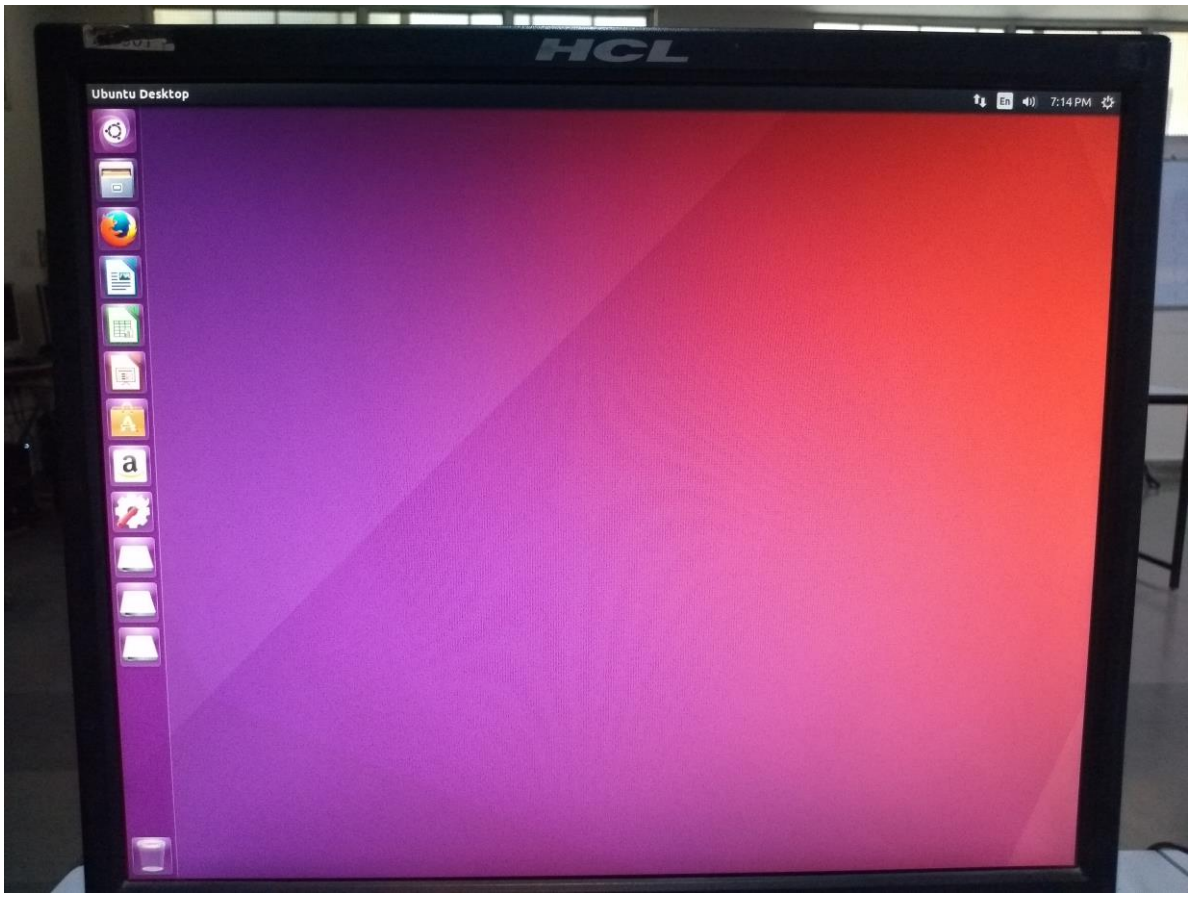
When system is completed the installation process, it will ask weather you want to Restart Now or later on. Click on Restart Now.



Step -18

Give the password to log in into the system.

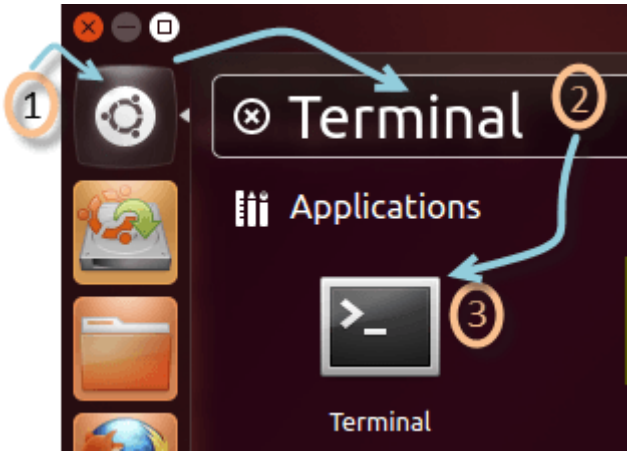



HCL
Ubuntu Desktop
7:14 PM

Experiment -1
Linux Command with Examples

There are 2 ways to launch the terminal.

1) Go to the Dash and type terminal



2) Or you can press **CTRL + Alt + T** to launch the Terminal

Once you launch the CLI (Terminal), you would find something as

```
miral@miral-Inspiron-N4050:~$
```

1) The first part of this line is the name of the **user**

2) The second part is the computer name or the host name. The hostname helps identify a computer over the network. In a server environment, host-name becomes important.

3) The **':'** is a simple separator

4) The tilde '~' sign shows that the user in working in the **home directory**. If you change the directory, this sign will vanish.

```
miral@miral-Inspiron-N4050:~$ cd /bin/
miral@miral-Inspiron-N4050:/bin$ cd /home//miral
miral@miral-Inspiron-N4050:~$
```

In the above illustration, we have moved from the /home directory to /bin using the **'cd' command**. The ~ sign does not display while working in /bin directory. It appears while moving back to the home directory.

5) The '$' sign suggests that you are working as a regular user in Linux. While working as a root user, '#' is displayed.

```
root@miral-Inspiron-N4050:/home/miral#
```

Present Working Directory

The directory that you are currently browsing is called the Present working directory. You log on to the home directory when you boot your PC. If you want to determine the directory you are presently working on, use the command -

pwd

```
miral@miral-Inspiron-N4050:~$ pwd
/home/miral
miral@miral-Inspiron-N4050:~$
```

pwd command stands for **p**rint **w**orking **d**irectory

Above figure shows that **/home/miral** is the directory we are currently working on.

Changing Directories

If you want to change your current directory use the '**cd**' command.

cd /tmp

Consider the following example.

```
miral@miral-Inspiron-N4050:~$ cd /tmp
miral@miral-Inspiron-N4050:/tmp$ cd /bin
miral@miral-Inspiron-N4050:/bin$ cd /usr
miral@miral-Inspiron-N4050:/usr$ cd /tmp
miral@miral-Inspiron-N4050:/tmp$
```

Here, we moved from directory /tmp to /bin to /usr and then back to /tmp.

Navigating to home directory

If you want to navigate to the home directory, then type **cd**.

cd

```
miral@miral-Inspiron-N4050:/tmp$ cd
miral@miral-Inspiron-N4050:~$
```

You can also use the **cd ~** command.

```
miral@miral-Inspiron-N4050:/tmp$ cd ~
miral@miral-Inspiron-N4050:~$
```

cd ~

Moving to root directory

The root of the file system in Linux is denoted by '/'. Similar to 'c:\' in Windows.

Note: In Windows, you use backward slash "\" while in UNIX/Linux, forward slash is used "/"

Type 'cd /' to move to the root directory.

cd /

```
miral@miral-Inspiron-N4050:~$ cd /
miral@miral-Inspiron-N4050:/$
```

TIP: Do not forget space between **cd** and /. Otherwise, you will get an error.

Navigating through multiple directories

You can navigate through multiple directories at the same time by specifying its complete path.

Example: If you want to move the /cpu directory under /dev, we do not need to break this operation in two parts.

Instead, we can type '/dev/cpu' to reach the directory directly.

cd /dev/cpu

```
miral@miral-Inspiron-N4050:~$ cd /dev/cpu
miral@miral-Inspiron-N4050:/dev/cpu$
```

Moving up one directory level

For navigating up one directory level, try.

cd ..

```
miral@miral-Inspiron-N4050:/dev/cpu$ cd ..
miral@miral-Inspiron-N4050:/dev$ cd ..
miral@miral-Inspiron-N4050:/$
```

Here by using the 'cd ..' command, we have moved up one directory from '/dev/cpu' to '/dev'.

Then by again using the same command, we have jumped from '/dev' to '/' root directory.

Relative and Absolute Paths

A path in computing is the address of a file or folder.

Example - **In Windows**

C:\documentsandsettings\user\downloads

In Linux

/home/user/downloads

There are two kinds of paths:

1. Absolute Path:

Let's say you have to browse the images stored in the Pictures directory of the home folder 'miral'.

The absolute file path of Pictures directory **/home/miral/Pictures**

To navigate to this directory, you can use the command.

cd /home/miral/Pictures

```
miral@miral-Inspiron-N4050:~$ cd /home/miral/Pictures
miral@miral-Inspiron-N4050:~/Pictures$
```

This is called absolute path as you are specifying the full path to reach the file.

2. Relative Path:

The Relative path comes in handy when you have to browse another subdirectory within a given directory.

It saves you from the effort to type complete paths all the time.

Suppose you are currently in your Home directory. You want to navigate to the Downloads directory.

You do no need to type the absolute path

cd /home/miral/Downloads

```
miral@miral-Inspiron-N4050:~$ cd /home/miral/Downloads/
miral@miral-Inspiron-N4050:~/Downloads$
```

Instead, you can simply type **'cd Downloads'** and you would navigate to the Downloads directory as you are already present within the **'/home/miral'** directory.

cd Downloads

```
miral@miral-Inspiron-N4050:~$ cd Downloads
miral@miral-Inspiron-N4050:~/Downloads$
```

This way you do not have to specify the complete path to reach a specific location within the same directory in the file system.

Listing files (ls)

If you want to see the list of files on your UNIX or Linux system, use the 'ls' command. It shows the files /directories in your current directory.

Note:

- Directories are denoted in blue color.
- Files are denoted in white.

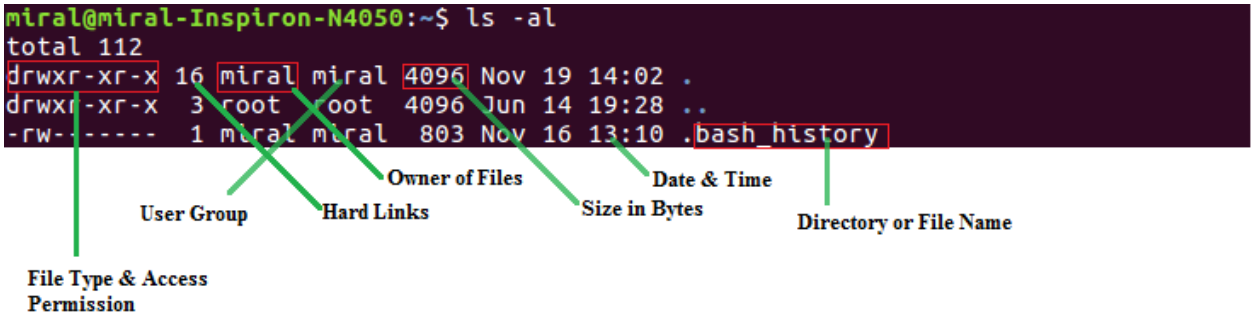
```
miral@miral-Inspiron-N4050:~$ ls
Desktop    Downloads         Music     Public     Videos
Documents  examples.desktop  Pictures  Templates
miral@miral-Inspiron-N4050:~$
```

Suppose, your main folder have sub-directories and files. You can use '**ls -R**' to shows all the files not only in directories but also subdirectories.

```
miral@miral-Inspiron-N4050:~$ ls -R
.:
Desktop    Downloads         Music     Public     Videos
Documents  examples.desktop  Pictures  Templates

./Desktop:
Linux Command Document.odt  Linux_Command Results

./Desktop/Linux_Command Results:
Absolute_Path.png              Moving Up One Directory Level.png
cd_1_command.png               Navigate_Home_Directory.png
cd_command.png                 Navigate_Multiple_Directory.png
cd_~.png                       pwd_command.png
Linux_Terminal_Saperator.png   Relative_Path_1.png
ls.png                         Relative_Path_2.png
Move_from_home_to_bin.png      root_Directory.png
Moving to root directory.png

./Documents:

./Downloads:

./Music:
```

'**ls -al**' gives detailed information of the files. The command provides information in a columnar format. The columns contain the following information:

1st Column	File type and access permissions
2nd Column	# of HardLinks to the File
3rd Column	Owner and the creator of the file
4th Column	Group of the owner
5th Column	File size in Bytes
6th Column	Date and Time
7th Column	Directory or File name



Listing Hidden Files

Hidden items in UNIX/Linux begin with ● Symbol - at the start, of the file or directory.

Any Directory/file starting with a '.' will not be seen unless you request for it. To view hidden files, use the command.

ls –a

```
miral@miral-Inspiron-N4050:~$ ls -a
.               Desktop           .ICEauthority  .sudo_as_admin_successful
..              .dmrc             .local         Templates
.bash_history   Documents         .mozilla       Videos
.bash_logout    Downloads         Music          .Xauthority
.bashrc         examples.desktop  Pictures       .xsession-errors
.cache          .gconf            .profile       .xsession-errors.old
.config         .gnupg            Public
miral@miral-Inspiron-N4050:~$
```

Try Following Commands:

1) ls -l

2) ls -il

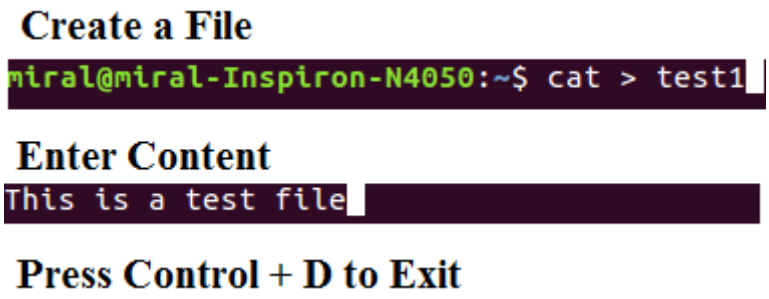
3) ls -r

Creating & Viewing Files

The 'cat' command is used to display text files. It can also be used for copying, combining and creating new text files. Let's see how it works.

To create a new file, use the command

- cat > filename
- Add content
- Press 'ctrl + d' to return to command prompt.



To view a file, use the command –

cat filename

```
miral@miral-Inspiron-N4050:~$ cat test1
This is a test filemiral@miral-Inspiron-N4050:~$
```

Let's see another file test2

```
miral@miral-Inspiron-N4050:~$ cat > test2
this is test2 file
```

The syntax to combine 2 files is –

cat file1 file2 > newfilename

Let's combine test1 and test2.

```
miral@miral-Inspiron-N4050:~$ cat test1 test2 > test
miral@miral-Inspiron-N4050:~$
```

As soon as you insert this command and hit enter, the files are concatenated, but you do not see a result. This is because **Bash Shell (Terminal) is silent type**. It will never give you a

confirmation message like "OK" or "Command Successfully Executed". It will only show a message when something goes wrong or when an error has occurred.

To view the new combo file "test" use the command

cat test

```
miral@miral-Inspiron-N4050:~$ cat test
This is a test filethis is test2 filemiral@miral-Inspiron-N4050:~$
```

Note: Only text files can be displayed and combined using this command.

Deleting Files

The 'rm' command removes files from the system without confirmation.

To remove a file use syntax -

rm filename

List Current Content of Directory

```
miral@miral-Inspiron-N4050:~$ ls
Desktop    Downloads         Music     Public     test    test2
Documents  examples.desktop  Pictures  Templates  test1   Videos
miral@miral-Inspiron-N4050:~$
```

Remove the file test1

```
miral@miral-Inspiron-N4050:~$ rm test1
```

List Directory, to check file has been deleted

```
miral@miral-Inspiron-N4050:~$ ls
Desktop    Downloads         Music     Public     test    Videos
Documents  examples.desktop  Pictures  Templates  test2
miral@miral-Inspiron-N4050:~$
```

Moving and Re-naming files

To move a file, use the command.

mv filename new_file_location

Suppose we want to move the file "test2" to location /home/miral/Documents. Executing the command

For renaming file:

mv filename newfilename

```
miral@miral-Inspiron-N4050:~$ mv test test1
miral@miral-Inspiron-N4050:~$ ls
Desktop    Downloads         Music     Public     test1
Documents  examples.desktop  Pictures  Templates  Videos
miral@miral-Inspiron-N4050:~$
```

Copy files

cp stands for **copy**. This command is used to copy files or group of files or directory. It creates an exact image of a file on a disk with different file name. *cp* command require at least two filenames in its arguments.

cp command works on three principal modes of operation and these operations depend upon number and type of arguments passed in cp command :

1. **Two file names**: If the command contains two file names, then it copy the contents of 1st file to the 2nd file. If the 2nd file doesn't exist, then first it creates one and content is copied to it. But if it existed then it is simply overwritten without any warning. So be careful when you choose destination file name.

   cp Src_file Dest_file

Suppose there is a directory named *test* having a text file a.txt.

```
$ ls
a.txt

$ cp a.txt b.txt

$ ls
a.txt  b.txt
```

2. **One or more arguments** : If the command has one or more arguments, specifying file names and following those arguments, an argument specifying directory name then this command copies each source file to the destination directory with the same name, created if not existed but if already existed then it will be overwritten, so be careful !!.

   cp Src_file1 Src_file2 Src_file3 Dest_directory

Suppose there is a directory named *test* having a text file **a.txt, b.txt** and a directory name **new** in which we are going to copy all files.

```
$ ls
a.txt  b.txt  new

Initially new is empty
$ ls new

$ cp a.txt b.txt new

$ ls new
a.txt  b.txt
```

Note: For this case last argument *must* be a **directory** name. For the above command to work, *Dest_directory* must exist because **cp** command won't create it.

3. **Two directory names**: If the command contains two directory names, cp copies all files of the source directory to the destination directory, creating any files or directories needed. This mode of operation requires an additional option, typically R, to indicate the recursive copying of directories.

```
$ ls test/
a.txt  b.txt  b.txt~  Folder1  Folder2

Without -r option, error
$ cp test gfg
cp: -r not specified; omitting directory 'test'

With -r, execute successfully
$ cp -r test gfg

$ ls gfg/
a.txt  b.txt  b.txt~  Folder1  Folder2
```

Copying using * wildcard: The star wildcard represents anything i.e. all files and directories. Suppose we have many text document in a directory and wants to copy it another directory, it takes lots of time if we copy files 1 by 1 or command becomes too long if specify all these file names as the argument, but by using * wildcard it becomes simple.

```
Initially Folder1 is empty
$ ls
a.txt  b.txt  c.txt  d.txt  e.txt  Folder1

$ cp *.txt Folder1
```

```
$ ls Folder1
a.txt  b.txt  c.txt  d.txt  e.txt
```

Directory Manipulations

Directories can be created on a Linux operating system using the following command

mkdir directoryname

This command will create a subdirectory in your present working directory, which is usually your "Home Directory".

For example,

mkdir mydirectory

```
miral@miral-Inspiron-N4050:~$ mkdir mydirectory
miral@miral-Inspiron-N4050:~$ ls
Desktop    Downloads          Music        Pictures  Templates  Videos
Documents  examples.desktop   mydirectory  Public    test1
miral@miral-Inspiron-N4050:~$
```

If you want to create a directory in a different location other than 'Home directory', you could use the following command –

mkdir

For example:

mkdir /tmp/MUSIC

will create a directory 'Music' under '/tmp' directory

```
miral@miral-Inspiron-N4050:~$ mkdir /tmp/MUSIC
miral@miral-Inspiron-N4050:~$ ls /tmp
config-err-lKj9zE
lu3625sblzdf.tmp
MUSIC
OSL_PIPE_1000_SingleOfficeIPC_e17b5f4d595a6c19b73ad3092c8744d
systemd-private-5246c87a9eb14a86af7dd75a323f0661-colord.service-rVcwz4
systemd-private-5246c87a9eb14a86af7dd75a323f0661-rtkit-daemon.service-1dOlk4
systemd-private-5246c87a9eb14a86af7dd75a323f0661-systemd-timesyncd.service-6605I
T
unity_support_test.0
miral@miral-Inspiron-N4050:~$
```

You can also create more than one directory at a time.

```
miral@miral-Inspiron-N4050:~$ mkdir dir1 dir2 dir3
miral@miral-Inspiron-N4050:~$ ls
Desktop  dir2  Documents  examples.desktop  mydirectory  Public     test1
dir1     dir3  Downloads  Music             Pictures     Templates  Videos
miral@miral-Inspiron-N4050:~$
```

Removing Directories

To remove a directory, use the command -

rmdir directoryname

Example

rmdir mydirectory

will delete the directory mydirectory

```
miral@miral-Inspiron-N4050:~$ rmdir mydirectory
miral@miral-Inspiron-N4050:~$ ls
Desktop  dir2  Documents  examples.desktop  Pictures  Templates  Videos
dir1     dir3  Downloads  Music             Public    test1
miral@miral-Inspiron-N4050:~$
```

Tip: Ensure that there is no file / sub-directory under the directory that you want to delete. Delete the files/sub-directory first before deleting the parent directory.

```
miral@miral-Inspiron-N4050:~$ rmdir Documents
rmdir: failed to remove 'Documents': Directory not empty
miral@miral-Inspiron-N4050:~$
```

Renaming Directory

The 'mv' (move) command (covered earlier) can also be used for renaming directories. Use the below-given format:

mv directoryname newdirectoryname

Let us try it:

```
miral@miral-Inspiron-N4050:~$ mv mydirectory newdirectory
miral@miral-Inspiron-N4050:~$ ls
Desktop  dir2  Documents  examples.desktop  newdirectory  Public     test1
dir1     dir3  Downloads  Music             Pictures      Templates  Videos
miral@miral-Inspiron-N4050:~$
```

The 'Man' command

Man stands for manual which is a reference book of a Linux operating system. It is similar to HELP file found in popular software.

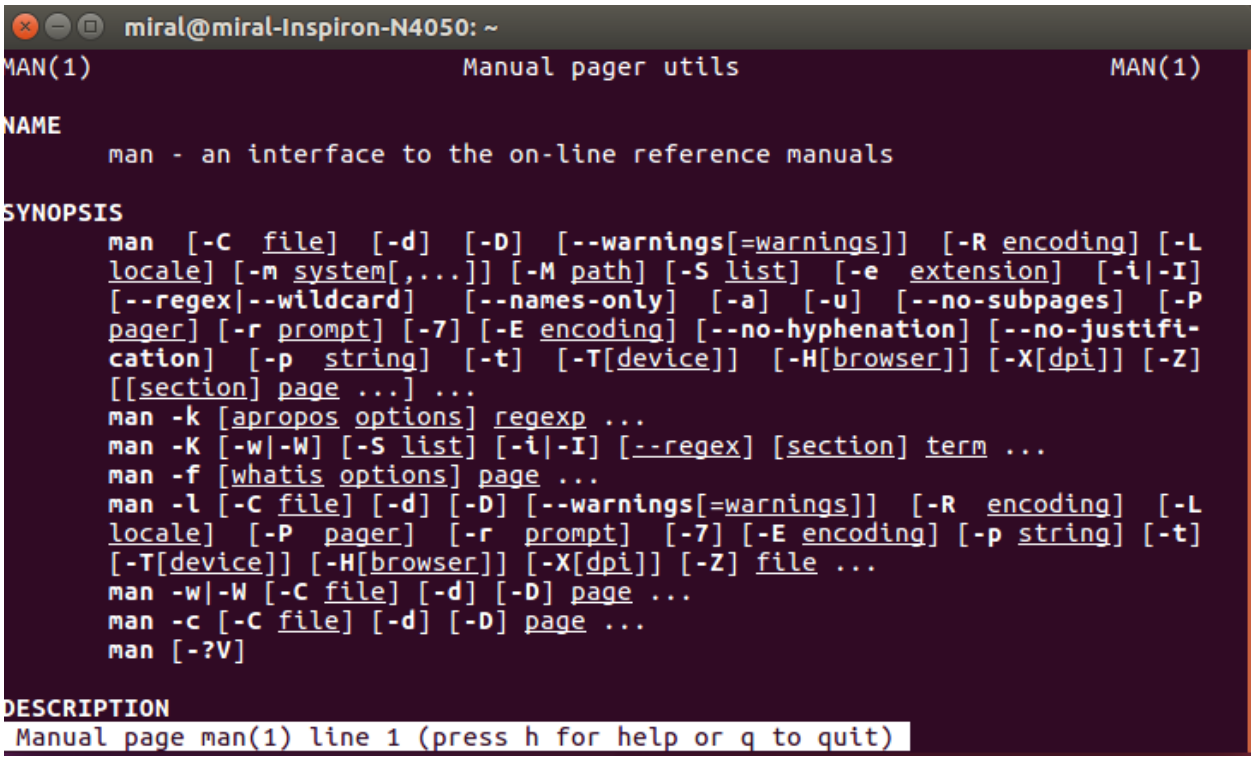
To get help on any command that you do not understand, you can type

man

The terminal would open the manual page for that command.

For an example, if we type *man man* and hit enter; terminal would give us information on man command

```
miral@miral-Inspiron-N4050:~$ man man
```



The History Command

History command shows all the commands that you have used in the past for the current terminal session. This can help you refer to the old commands you have entered and re-used them in your operations again.

```
miral@miral-Inspiron-N4050:~$ history
    1  ls
    2  cd ..
    3  ls
    4  cd
    5  ls
    6  cd /root/
    7  ls
    8  cd
    9  clear
   10  cd /bin
   11  ls
   12  cd /home
   13  ls
   14  clear
   15  cd ..
   16  ls
   17  clear
   18  cd ..
   19  clear
   20  cd /bin
   21  clear
   22  cd ..
   23  clear
```

The clear command

This command clears all the clutter on the terminal and gives you a clean window to work on, just like when you launch the terminal.

```
  188  mkdir dir1 dir2 dir3
  189  ls
  190  clear
  191  ls
  192  clear
  193  rmdir mydirectory
  194  ls
  195  rmdir Documents
  196  clear
  197  rmdir Documents
  198  clear
  199  ls
  200  mkdir mydirectory
  201  ls
  202  clear
  203  ls
  204  clear
  205  mv mydirectory newdirectory
  206  ls
  207  clear
  208  man man
  209  clear
  210  history
miral@miral-Inspiron-N4050:~$ clear
```

The Window gets Cleared

```
miral@miral-Inspiron-N4050:~$
```

Pasting commands into the terminal

Many times you would have to type in long commands on the Terminal. Well, it can be annoying at times, and if you want to avoid such a situation then copy, pasting the commands can come to rescue.

For copying, the text from a source, you would use **Ctrl + c**, but for pasting it on the Terminal, you need to use **Ctrl + Shift + p**. You can also try **Shift + Insert or select Edit>Paste on the menu**

NOTE: With Linux upgrades, these shortcuts keep changing. You can set your preferred shortcuts via Terminal> Edit> Keyboard Shortcuts.

Installing Software

In windows, the installation of a program is done by running the setup.exe file. The installation bundle contains the program as well various dependent components required to run the program correctly.

In Linux/UNIX, installation files are distributed as packages. But the package contains only the program itself. Any dependent components will have to be installed separately which are usually available as packages themselves.

You can use the **apt** commands to install or remove a package. Let's update all the installed packages in our system using command -

sudo apt-get update

```
miral@miral-Inspiron-N4050:~$ sudo apt-get update
[sudo] password for miral:
```

Sudo program allows regular users to run programs with the security privileges of the superuser or root.

Sudo command will ask for password authentication. Though, you do not need to know the root password. You can supply your own password. After authentication, the system will invoke the requested command.

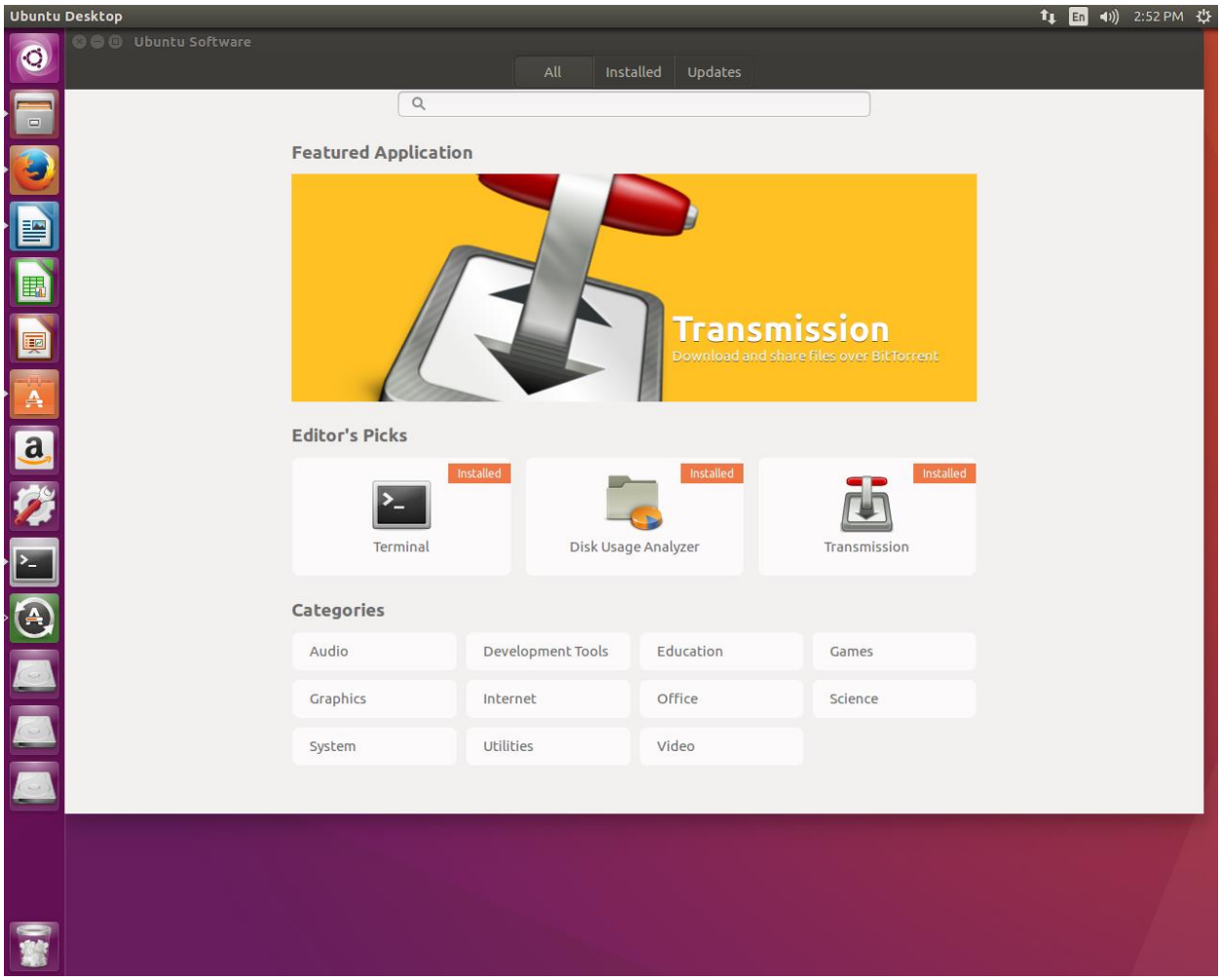
Sudo maintains a log of each command run. System administrators can trackback the person responsible for undesirable changes in the system.

```
Get:2 http://security.ubuntu.com/ubuntu xenial-security InRelease [107 kB]
Get:3 http://in.archive.ubuntu.com/ubuntu xenial-updates InRelease [109 kB]
Get:4 http://in.archive.ubuntu.com/ubuntu xenial-backports InRelease [107 kB]
Get:5 http://security.ubuntu.com/ubuntu xenial-security/main i386 Packages [500
kB]
Get:6 http://in.archive.ubuntu.com/ubuntu xenial/main i386 Packages [1,196 kB]
Get:7 http://security.ubuntu.com/ubuntu xenial-security/main Translation-en [244
 kB]
Get:8 http://security.ubuntu.com/ubuntu xenial-security/main i386 DEP-11 Metadat
a [67.7 kB]
Get:9 http://security.ubuntu.com/ubuntu xenial-security/main DEP-11 64x64 Icons
[68.0 kB]
Get:10 http://security.ubuntu.com/ubuntu xenial-security/restricted i386 Package
s [7,224 B]
Get:11 http://in.archive.ubuntu.com/ubuntu xenial/main Translation-en [568 kB]
Get:12 http://security.ubuntu.com/ubuntu xenial-security/restricted Translation-
en [2,152 B]
Get:13 http://security.ubuntu.com/ubuntu xenial-security/restricted i386 DEP-11
Metadata [199 B]
Get:14 http://security.ubuntu.com/ubuntu xenial-security/universe i386 Packages
[346 kB]
Get:15 http://security.ubuntu.com/ubuntu xenial-security/universe Translation-en
 [155 kB]
85% [14 Packages store 0 B] [11 Translation-en 494 kB/568 kB 87%] [15 Translati
```

Updates all installed packages

```
Get:56 http://in.archive.ubuntu.com/ubuntu xenial-backports/universe Translation
-en [4,184 B]
Get:57 http://in.archive.ubuntu.com/ubuntu xenial-backports/universe i386 DEP-11
 Metadata [5,104 B]
Get:58 http://in.archive.ubuntu.com/ubuntu xenial-backports/universe DEP-11 64x6
4 Icons [1,789 B]
Get:59 http://in.archive.ubuntu.com/ubuntu xenial-backports/multiverse i386 DEP-
11 Metadata [216 B]
Get:60 http://in.archive.ubuntu.com/ubuntu xenial-backports/multiverse DEP-11 64
x64 Icons [29 B]
Fetched 31.5 MB in 1min 22s (380 kB/s)
AppStream cache update completed, but some metadata was ignored due to errors.
Reading package lists... Done
miral@miral-Inspiron-N4050:~$
```

The easy and popular way to install programs on Ubuntu is by using the Software center as most of the software packages are available on it and it is far more secure than the files downloaded from the internet.



- Unix system has a “de facto standard” way to install a software.

  configure, make & make install

- Typical software installation procedure as following.
  - Download source code. Usually, it’s archived with tar command and compressed with gzip command.
  - configure command creates Makefile automatically which is used to compile the source.
  - Program compilation is written in Makefile.

Commands

- gzip compress a file
- gunzip uncompress a file

- tar archive or expand files
- configure create Makefile
- make compile & install software

Example: parallel programming library installation

gunzip software.tar.gz

tar –xvf software.tar

cd software

./install OR make all OR ./build etc…

File Permissions in Linux/Unix with Example

Linux is a clone of UNIX, the multi-user operating system which can be accessed by many users simultaneously. Linux can also be used in mainframes and servers without any modifications. But this raises security concerns as an unsolicited or malign user can corrupt, change or remove crucial data. For effective security, Linux divides authorization into 2 levels.

- Ownership
- Permission

The concept of permissions and ownership is crucial in Linux. Here, we will discuss both of them. Let us start with the Ownership.

Ownership of Linux files

Every file and directory on your Unix/Linux system is assigned 3 types of owner, given below.

User

A user is the owner of the file. By default, the person who created a file becomes its owner. Hence, a user is also sometimes called an owner.

Group

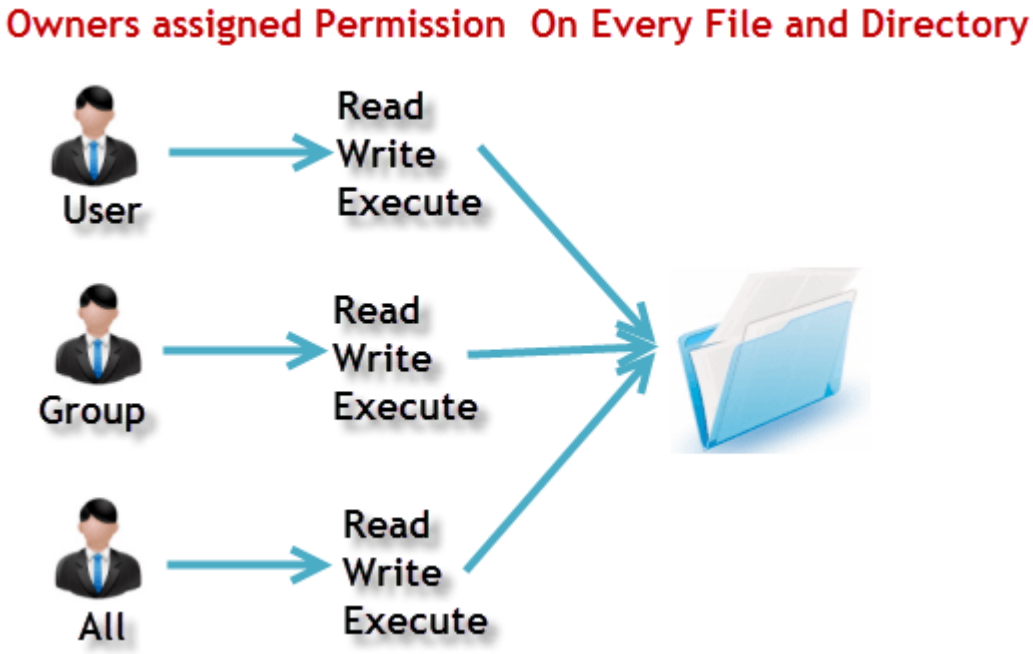
A user- group can contain multiple users. All users belonging to a group will have the same access permissions to the file. Suppose you have a project where a number of people require

access to a file. Instead of manually assigning permissions to each user, you could add all users to a group, and assign group permission to file such that only this group members and no one else can read or modify the files.

Other

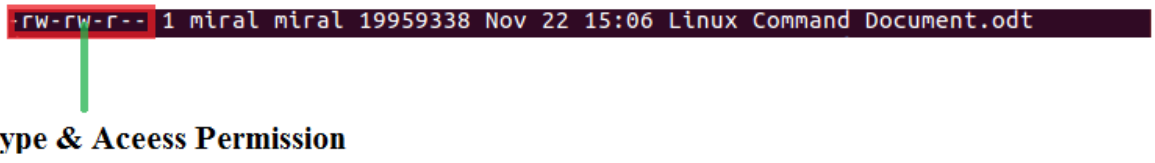
Any other user who has access to a file. This person has neither created the file, nor he belongs to a user group who could own the file. Practically, it means everybody else. Hence, when you set the permission for others, it is also referred as set permissions for the world.

Now, the big question arises how does **Linux distinguish** between these three user types so that a user 'A' cannot affect a file which contains some other user 'B's' vital information/data. It is like you do not want your colleague, who works on your Linux computer, to view your images. This is where **Permissions** set in, and they define **user behavior**.



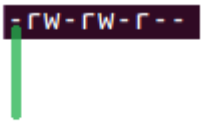
ls - l on terminal gives

ls - l

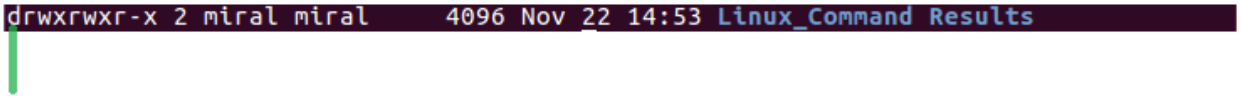


Here, we have highlighted '**-rw-rw-r--**'and this weird looking code is the one that tells us about the permissions given to the owner, user group and the world.

Here, the first '-' implies that we have selected a file.p>



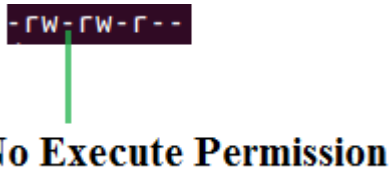
Else, if it were a directory, d would have been shown.



The characters are pretty easy to remember.
r = read permission
w = write permission
x = execute permission
- = no permission

Let us look at it this way.

The first part of the code is '**rw-**'. This suggests that the owner 'Home' can:



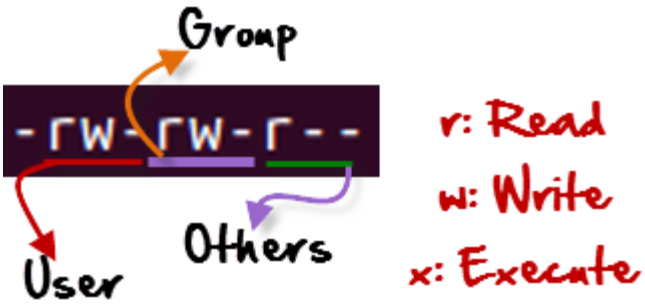
- Read the file
- Write or edit the file
- He cannot execute the file since the execute bit is set to '-'.

The second part is '**rw-**'. It for the user group 'Home' and group-members can:

- Read the file
- Write or edit the file

The third part is for the world which means any user. It says 'r--'. This means the user can only:

- Read the file



Changing file/directory permissions with 'chmod' command

Say you do not want your colleague to see your personal images. This can be achieved by changing file permissions.

We can use the '**chmod**' command which stands for 'change mode'. Using the command, we can set permissions (read, write, execute) on a file/directory for the owner, group and the world.

Syntax:

chmod permissions filename

There are 2 ways to use the command -

- Absolute mode
- Symbolic mode

Absolute (Numeric) Mode

In this mode, file permissions are not represented as characters but a three-digit octal number.

The table below gives numbers for all for permissions types.

Number	Permission Type	Symbol
0	No Permission	---
1	Execute	--x
2	Write	-w-
3	Execute + Write	-wx
4	Read	r--

5	Read + Execute	r-x
6	Read +Write	rw-
7	Read + Write +Execute	rwx

Let's see the chmod command in action.

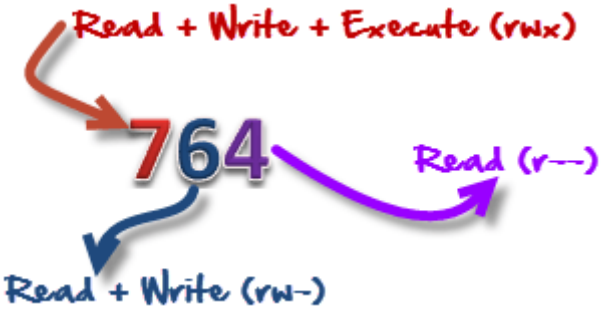
Checking Current File Permissions

```
miral@miral-Inspiron-N4050:~$ ls -l test1
-rw-rw-r-- 1 miral miral 37 Nov 19 14:41 test1
```

chmod 746 and checking permisssion again

```
miral@miral-Inspiron-N4050:~$ chmod 746 test1
miral@miral-Inspiron-N4050:~$ ls -l test1
-rwxr--rw- 1 miral miral 37 Nov 19 14:41 test1
```

In the above-given terminal window, we have changed the permissions of the file 'sample to '764'.



'764' absolute code says the following:

- Owner can read, write and execute
- Usergroup can read and write
- World can only read

This is shown as '-rwxrw-r-

This is how you can change the permissions on file by assigning an absolute number.

Symbolic Mode

In the Absolute mode, you change permissions for all 3 owners. In the symbolic mode, you can modify permissions of a specific owner. It makes use of mathematical symbols to modify the file permissions.

Operator	Description

+	Adds a permission to a file or directory
-	Removes the permission
=	Sets the permission and overrides the permissions set earlier.

The various owners are represented as –

User Denotations	
u	user/owner
g	group
o	other
a	all

We will not be using permissions in numbers like 755 but characters like rwx. Let's look into an example

Current File Permission

```
miral@miral-Inspiron-N4050:~$ ls -l test1
-rw-rw-r-- 1 miral miral 37 Nov 19 14:41 test1
```

Setting Permission to 'other' users

```
miral@miral-Inspiron-N4050:~$ chmod o=rwx test1
miral@miral-Inspiron-N4050:~$ ls -l test1
-rw-rw-rwx 1 miral miral 37 Nov 19 14:41 test1
```

Adding 'Execute' permission to the usergroup

```
miral@miral-Inspiron-N4050:~$ chmod g+x test1
miral@miral-Inspiron-N4050:~$ ls -l test1
-rw-rwxrwx 1 miral miral 37 Nov 19 14:41 test1
```

Removing 'read' permission for 'user'

```
miral@miral-Inspiron-N4050:~$ chmod u-r test1
miral@miral-Inspiron-N4050:~$ ls -l test1
--w-rwxrwx 1 miral miral 37 Nov 19 14:41 test1
```

Changing Ownership and Group

For changing the ownership of a file/directory, you can use the following command:

chown user

In case you want to change the user as well as group for a file or directory use the command

chown user:group filename

Let's see this in action

Check the current file ownership using ls -l

```
-rw-rw-r-- 1 root  miral        4 Nov 26 16:51 sample.txt
```

change file owner to miral. You will need sudo

```
miral@miral-Inspiron-N4050:~/Desktop$ sudo chown miral sample.txt
```

ownership changed to miral

```
-rw-rw-r-- 1 miral miral        4 Nov 26 16:51 sample.txt
```

changing user and group to root 'chown user:group file'

```
miral@miral-Inspiron-N4050:~/Desktop$ sudo chown root:root sample.txt
```

user and group ownership changed to root

```
-rw-rw-r-- 1 root  root         4 Nov 26 16:51 sample.txt
```

In case you want to change group-owner only, use the command

chgrp group_name filename

'**chgrp**' stands for change group

check the current file ownership using ls -l

```
-rw-rw-r-- 1 miral cdrom 31 Nov 26 17:03 test
```

chenge the file owner to root. you will need sudo

```
miral@miral-Inspiron-N4050:~$ sudo chgrp root test
```

group ownership change to root

```
miral@miral-Inspiron-N4050:~$ ls -dl test
-rw-rw-r-- 1 miral root 31 Nov 26 17:03 test
```

Tip

- The file /etc/group contains all the groups defined in the system
- You can use the command "groups" to find all the groups you are a member of

```
miral@miral-Inspiron-N4050:~$ groups
miral adm cdrom sudo dip plugdev lpadmin sambashare
```

- You can use the command newgrp to work as a member a group other than your default group

```
miral@miral-Inspiron-N4050:~$ newgrp cdrom
miral@miral-Inspiron-N4050:~$ cat > test
this is a test to change group
^C
miral@miral-Inspiron-N4050:~$ ls -dl test
-rw-rw-r-- 1 miral cdrom 31 Nov 26 17:03 test
miral@miral-Inspiron-N4050:~$
```

- You cannot have 2 groups owning the same file.
- You do not have nested groups in Linux. One group cannot be sub-group of other
- x- executing a directory means Being allowed to "enter" a dir and gain possible access to sub-dirs

'bc' command

work as basic calculator

```
miral@miral-Inspiron-N4050:~$ bc
bc 1.06.95
Copyright 1991-1994, 1997, 1998, 2000, 2004, 2006 Free Software Foundation, Inc.
This is free software with ABSOLUTELY NO WARRANTY.
For details type `warranty'.
2+2
4
2-2
0
2*2
4
2/2
1
```

Try following command

bc -v

‘cal Command’

Display calender of current month

```
miral@miral-Inspiron-N4050:~$ cal
   December 2018
Su Mo Tu We Th Fr Sa
                   1
 2  3  4  5  6  7  8
 9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29
30 31
miral@miral-Inspiron-N4050:~$
```

Try following command

cal jan 2019

cal 2020

cal -m 2

cal -h

‘date’ command

displaying current date

```
miral@miral-Inspiron-N4050:~$ date
Tue Dec 11 20:46:01 IST 2018
```

‘echo’ command

print message on screen

```
miral@miral-Inspiron-N4050:~$ echo hello world
hello world
```

‘expr’ command

for any arithmatic and logical operation

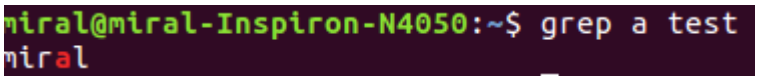
```
miral@miral-Inspiron-N4050:~$ expr 2+3
2+3
```

Try following commnad

exper length

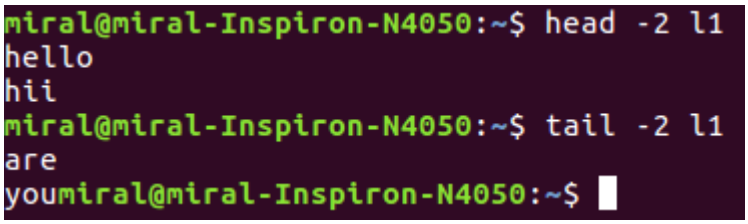
‘grep’ command

it is use to search pattern / word from file



‘head & tail command’

for displaying first and last lines of respective file

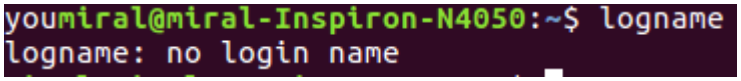


Try following command

head -n filename | tail -m

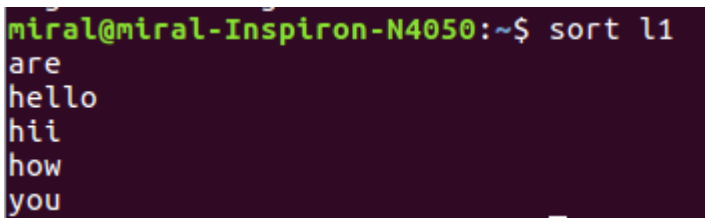
‘logname’ command

login information



‘sort’ command

sort file content



‘touch’ command

create any number of files

```
miral@miral-Inspiron-N4050:~$ touch l2 l3 l4
miral@miral-Inspiron-N4050:~$ ls
abc          dir2              f1        j      Music          Templates
a.sh         dir3              ffff      j.sh   mysecondtest   test
calc.sh      Documents         f.sh      k      mytest         test1
cline.sh     Downloads         g.sh      k.sh   nestedif.sh    Videos
command.sh   d.sh              hello.sh  l1     newdirectory
c.sh         e.sh              i         l2     number
Desktop      examples.desktop  if.sh     l3     Pictures
dir1         f                 i.sh      l4     Public
```

‘tty’ command

terminal information

```
miral@miral-Inspiron-N4050:~$ tty
/dev/pts/2
miral@miral-Inspiron-N4050:~$
```

‘uname’ command

information about system

```
miral@miral-Inspiron-N4050:~$ uname
Linux
```

‘who’ command

give all the working users in the system

```
miral@miral-Inspiron-N4050:~$ who
miral    tty7         2018-12-11 20:13 (:0)
```

‘who am i’ command

give name of that user which is currently logged in.

```
miral@miral-Inspiron-N4050:~$ who am i
miral@miral-Inspiron-N4050:~$
```

‘compgen’ command

list existing users and groups in the system.

```
miral@miral-Inspiron-N4050:~$ compgen -u
root
daemon
bin
sys
sync
games
man
lp
mail
news
uucp
proxy
www-data
backup
list
irc
gnats
```

```
miral@miral-Inspiron-N4050:~$ compgen -g
root
daemon
bin
sys
adm
tty
disk
lp
mail
news
uucp
man
proxy
kmem
```

‘addgroup’ command

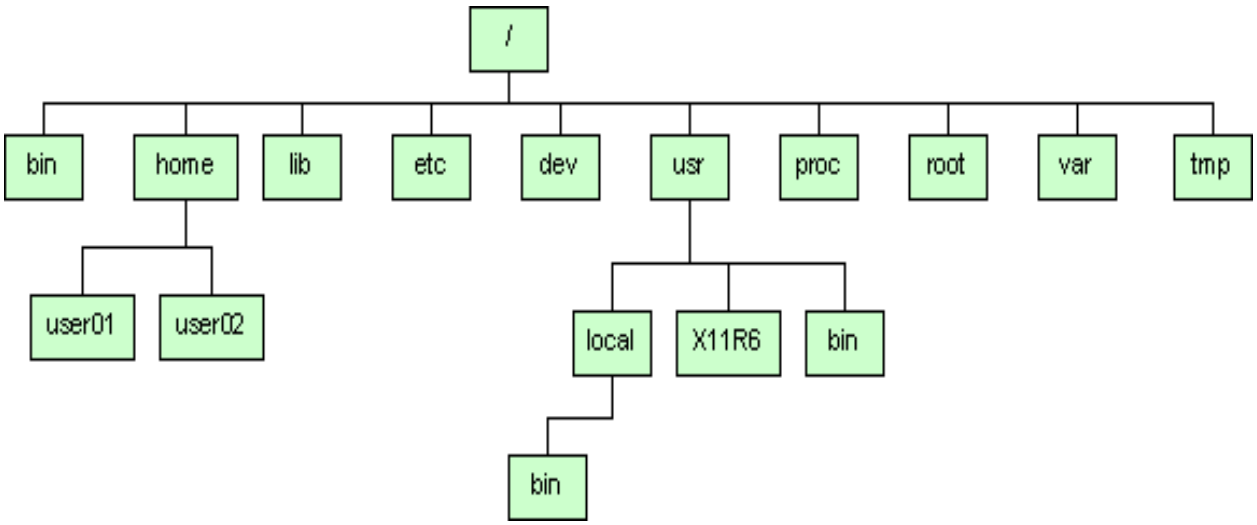
add group in the root directory

```
miral@miral-Inspiron-N4050:~$ sudo su
[sudo] password for miral:
root@miral-Inspiron-N4050:/home/miral# addgroup g3
Adding group `g3' (GID 1001) ...
Done.
```

```
postdrop
g3
```

Directories Found On Linux Systems

- In Linux files are put in a directory.
- All directories are in a hierarchical structure (tree structure).
- User can put and remove any directories on the tree.
- The Top directory is "/", which is called slash or root.
- Users have the own directory which is called home directory.



Directory	Comments
`/`	The root directory. Where everything begins.
`/bin`	Contains binaries (programs) that must be present for the system to boot and run.
`/boot`	Contains the Linux kernel, initial RAM disk image (for drivers needed at boot time), and the boot loader. Interesting files: • `/boot/grub/grub.conf` or `menu.lst`, which are used to configure the boot loader. • `/boot/vmlinuz`, the Linux kernel
`/dev`	This is a special directory which contains *device nodes*. "Everything is a file" also applies to devices. Here is where the kernel maintains a list of all the devices it understands.

Directory	Comments
`/etc`	The `/etc` directory contains all of the system-wide configuration files. It also contains a collection of shell scripts which start each of the system services at boot time. Everything in this directory should be readable text. Interesting files: While everything in `/etc` is interesting, here are some of my all-time favorites: • `/etc/crontab`, a file that defines when automated jobs will run. • `/etc/fstab`, a table of storage devices and their associated mount points. • `/etc/passwd`, a list of the user accounts.
`/home`	In normal configurations, each user is given a directory in `/home`. Ordinary users can only write files in their home directories. This limitation protects the system from errant user activity.
`/lib`	Contains shared library files used by the core system programs. These are similar to DLLs in Windows.
`/lost+found`	Each formatted partition or device using a Linux file system, such as ext3, will have this directory. It is used in the case of a partial recovery from a file system corruption event. Unless something really bad has happened to your system, this directory will remain empty.

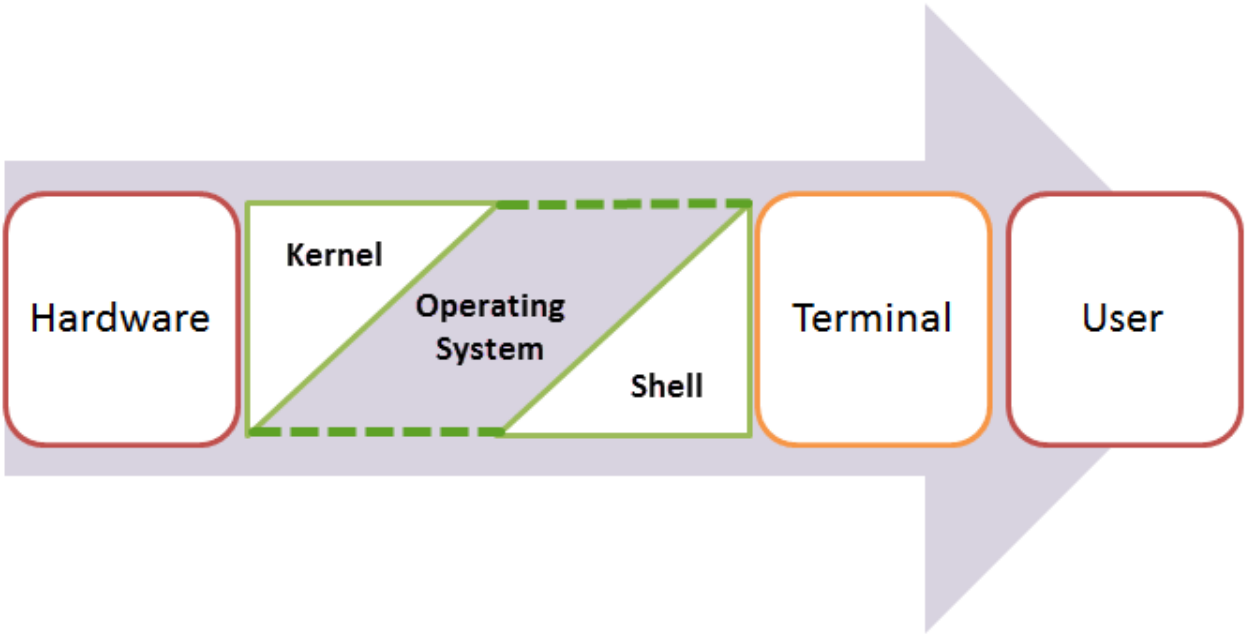
`/media`	On modern Linux systems the `/media` directory will contain the mount points for removable media such as USB drives, CD-ROMs, etc. that are mounted automatically at insertion.
`/mnt`	On older Linux systems, the `/mnt` directory contains mount points for removable devices that have been mounted manually.
`/opt`	The `/opt` directory is used to install "optional" software. This is mainly used to hold commercial software products that may be installed on your system.

Experiment - 2

Shell Scripting

An Operating is made of many components, but its two prime components are -

- Kernel
- Shell



A Kernel is at the nucleus of a computer. It makes the communication between the hardware and software possible. While the Kernel is the innermost part of an operating system, a shell is the outermost one.

A shell in a Linux operating system takes input from you in the form of commands, processes it, and then gives an output. It is the interface through which a user works on the programs, commands, and scripts. A shell is accessed by a terminal which runs it.

When you run the terminal, the Shell issues **a command prompt (usually $),** where you can type your input, which is then executed when you hit the Enter key. The output or the result is thereafter displayed on the terminal.

The Shell wraps around the delicate interior of an Operating system protecting it from accidental damage. Hence the name **Shell**.

- **Types of Shell**

There are two main shells in Linux:

1. The **Bourne Shell**: The prompt for this shell is $ and its derivatives are listed below:

- POSIX shell also is known as sh
- Korn Shell also knew as sh
- **B**ourne **A**gain **SH**ell also knew as bash (most popular)

2. The C shell: The prompt for this shell is %, and its subcategories are:

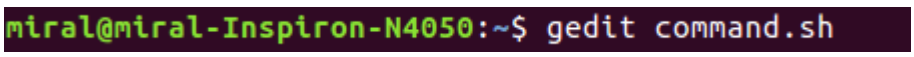
- C shell also is known as csh
- Tops C shell also is known as tcsh

▪ What is Shell Scripting?

Shell scripting is writing a series of command for the shell to execute. It can combine lengthy and repetitive sequences of commands into a single and simple script, which can be stored and executed anytime. This reduces the effort required by the end user.

Let us understand the steps in creating a Shell Script.

1) Go to the Linux Terminal and type **gedit command.sh**



2) Write the following command in command.sh

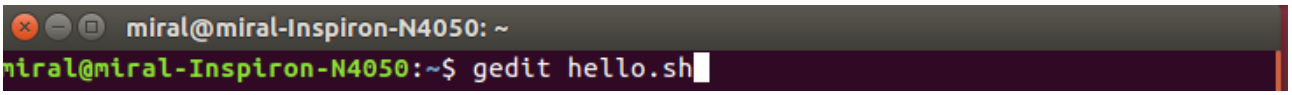
3) close the gedit terminal by pressing **ctrl + c**

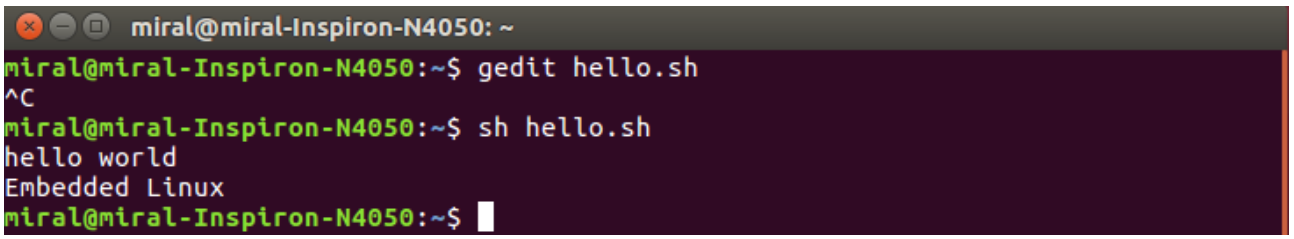
4) Type command **sh command.sh** to execute your script.

```
miral@miral-Inspiron-N4050:~$ sh command.sh
abc          dir1        examples.desktop  i       Music          Public
a.sh         dir2        f                 if.sh   mysecondtest   Templates
calc.sh      dir3        f1                i.sh    mytest         test
cline.sh     Documents   ffff              j       nestedif.sh    test1
command.sh   Downloads   f.sh              j.sh    newdirectory   Videos
c.sh         d.sh        g.sh              k       number
Desktop      e.sh        hello.sh          k.sh    Pictures
/home/miral
```

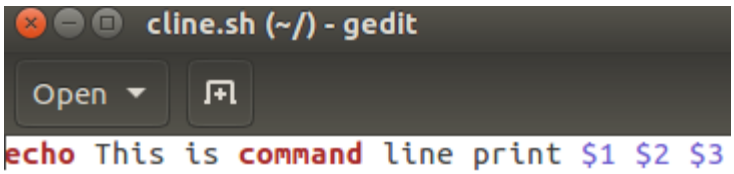
Implement a Shell Scripts Program.

Program : Write a Shell Script which will display the “Hello World”.





Program: Write a Shell Script by using command line.



Output:

```
miral@miral-Inspiron-N4050: ~
miral@miral-Inspiron-N4050:~$ gedit cline.sh
^C
miral@miral-Inspiron-N4050:~$ sh cline.sh Hello Embedded Linux
This is command line print Hello Embedded Linux
miral@miral-Inspiron-N4050:~$
```

Program: Write a Shell Script which calculates the arithmetic operation.

```
calc.sh (~/) - gedit
Open                                                    Save
a=10;
b=20;
echo Addition is:
echo `expr $a + $b`
echo Subtraction is:
echo `expr $a - $b`
echo Multiplication is:
echo `expr $a \* $b`
echo Division is:
echo `expr $a / $b`
echo Modulo is:
echo `expr $a % $b`
```

Output

```
miral@miral-Inspiron-N4050: ~
miral@miral-Inspiron-N4050:~$ sh c
calc.sh   cline.sh
miral@miral-Inspiron-N4050:~$ sh calc.sh
Addition is:
30
Subtraction is:
-10
Multiplication is:
200
Division is:
0
Modulo is:
10
miral@miral-Inspiron-N4050:~$
```

Program: Write a Shell Script using if condition.

if.sh (~/) - gedit

Open Save

```
a=10
b=10
if [$a -eq $b]
then
 echo $a and $b is equal
else
 echo $a and $b is unequal
fi
```

Output

```
miral@miral-Inspiron-N4050: ~
miral@miral-Inspiron-N4050:~$ sh if.sh
10 and 10 is equal
miral@miral-Inspiron-N4050:~$
```

Program: Write a Shell Script using nested if condition.

nestedif.sh (~/) - gedit

Open Save

```
echo "Enter the Number :"
read number
if [ $number = '100' ]
then
        echo "Enter a :"
        read a
        if [ $a = '10' ]
        then
        echo "valid number"
        else
        echo "invalid number"
        fi
else
echo not enter properly
fi
```

Output:

```
miral@miral-Inspiron-N4050: ~
miral@miral-Inspiron-N4050:~$ sh nestedif.sh
Enter the Number :
100
Enter a :
10
valid number
miral@miral-Inspiron-N4050:~$
```

Program:

a) Write a script called 'a' which outputs the following:

- **your username**
- **the time and date**
- **who is logged on**
- **Also output a line of asterices (*********) after each section.**

```
a.sh (~/) - gedit
Open                                                    Save
echo "**************************";
echo "Username :"`uname`
echo "**************************";
echo "Date :"`date`
echo "**************************";
echo "Logged In :"`whoami`
echo "**************************";
```

Output

```
miral@miral-Inspiron-N4050:~$ sh a.sh
**************************
Username :Linux
**************************
Date :Thu Dec 6 20:29:19 IST 2018
**************************
Logged In :miral
**************************
miral@miral-Inspiron-N4050:~$
```

b) Put the command a into your .login file so that the script is executed every time that

You log on

```
sh /home/miral/a.sh
# ~/.bashrc: executed by bash(1) for non-login shells.
# see /usr/share/doc/bash/examples/startup-files (in the package bash-doc)
# for examples
```

Output

```
miral@miral-Inspiron-N4050: ~
**************************
Username :Linux
**************************
Date :Thu Dec 6 20:37:33 IST 2018
**************************
Logged In :miral
**************************
miral@miral-Inspiron-N4050:~$
```

c) Write a shell program to simulate a simple calculator.

```
read a
read b
read c
case $c in
        +)
           echo `expr $a + $b`
           ;;
        -)
           echo `expr $a - $b`
           ;;
        \*)
           echo `expr $a \* $b`
           ;;
        \/)
           echo `expr $a \/ $b`
           ;;
        *)
esac
```

Output

```
miral@miral-Inspiron-N4050:~$ gedit c.sh
^C
miral@miral-Inspiron-N4050:~$ sh c.sh
5
7
+
12
miral@miral-Inspiron-N4050:~$ sh c.sh
2
3
-
-1
miral@miral-Inspiron-N4050:~$
```

(d) Write a shell script to combine any three text files into a single file (append them in the Order as they appear in the arguments) and display the word count.

Program

```
echo "enter first"
read a
echo "enter second"
read b
echo "enter third"
read c

read f

cat $a $b $c > $f

echo f

echo "Number of Words"

wc -w $f
```

Output

```
bad morningmiral@miral-Inspiron-N4050:~$ sh e.sh
enter first
i
enter second
j
enter third
k
f
f
Number of Words
5 f
```

(e) Write a shell program to count the following in a text file.

- **Number of vowels in a given text file.**
- **Number of characters.**
- **Number of symbols.**
- **Number of lines**

Program

```
echo Enter the filename
read fn
echo char
wc -c $fn
echo lines
wc -l $fn
echo Special Symbol
grep -o -i "[@#$%&^*]" $fn
echo Vowel
grep -o -i "[aeiou]" $fn
```

Output

```
miral@miral-Inspiron-N4050:~$ sh f.sh
Enter the filename
mytest
char
18 mytest
lines
1 mytest
Special Symbol
@
Vowel
i
a
e
a
i
e
o
```

(f) Write a shell program to find the largest integer among the three integers given as Arguments.

Program

```
echo "enter first integer"
read a
echo "enter second integer"
read b
echo "enter third integer"
read c

if [ $a -gt $b ]
then
        if [ $a -gt $c ]
        then
                echo $a" is larger"
        else
                echo $c" is larger"
        fi
else
        if [ $b -gt $c ]
        then
                echo $b"is larger"
        else
                echo $c"is larger"
        fi
fi
```

Output

```
miral@miral-Inspiron-N4050:~$ sh g.sh
enter first integer
5
enter second integer
1
enter third integer
9
9 is larger
```

(g) Write a shell script that searches for a single word pattern recursively in the current Directory and displays the no. of times it occurred.

Program

```
echo -n "enter filename::"
read fn
echo -n "enter one word pattern::"
read pattern
grep -c $pattern $fn
```

Output

```
miral@miral-Inspiron-N4050:~$ sh i.sh
enter filename::mysecondtest
enter one word pattern::hi
3
```

(h) Write a shell program to sort a given file which consists of a list of numbers, in ascending Order.

Program

```
file=""
echo -n "Enter Filename : "
read file

if [ ! -f $file ]
then
echo "$file not a file!"
exit 1
fi

sort -n $file
```

Output

```
miral@miral-Inspiron-N4050:~$ sh j.sh
Enter Filename : number
1
2
3
```

Experiment-3

Aim: Introduction to Linux text editors : Vi, Vim, Nano & Gedit.

Theory:

Text editors can be used for writing code, editing text files such as configuration files, creating user instruction files and many more. In Linux, text editor are of two kinds that is graphical user interface (GUI) and command line text editors (console or terminal).

1) Vi/Vim Editor:-

Vi is a text editor in linux. It is a command line text editor. Vim is a powerful command line based text editor that has enhanced the functionalities of the old Unix Vi text editor. It is one the most popular and widely used text editors among System Administrators and programmers that is why many users often refer to it as a programmer's editor. It enables syntax highlighting when writing code or editing configuration files.

Essential Vi Commands:-

- Open a file:
  - vi filename
- To go into edit mode:
  - press ESC and type I
- To go into command mode:
  - press ESC
- To save a file
  - press ESC and type :w fileName
- To save a file and quit:
  - press ESC and type :wq
    OR
  - press ESC and type :x
- To jump to a line:
  - press ESC and type :the line number
- To Search for a string:
  - Press ESC and type /wordToSearch
- To quit vi:
  - Press ESC and type :q
- Save the following into a file called hello_world:

```
echo "Hello, World!"
```

- Save and close the file. You can run the script as follows:
  - chmod +x hello_world

- ./hello_world
- Sample Outputs:
  - Hello World

Benefits of Vim over Vi editor:-

Vim (vi-improved) is the enhanced and improved version of vi editor. It was developed by Bram Moolenarr in 1991. It includes everything which vi has. Besides all benefits of vi, vim provides following advantages over vi editor.

- It includes more features for programing language such as syntax highlighting, code folding, text formatting etc.
- It includes inbuilt utility for comparing files.
- It includes undo/redo facility.
- Vim supports external scripting language.
- It can edit compressed files.
- It can edit remote files over network protocol.
- It supports plugins for additional functionality.

2) Gedit:-

This is a general purpose GUI based text editor and is installed by default text editor on Gnome desktop environment. It is simple to use, highly pluggable and a powerful editor with the following features:

1. Support for UTF-8
2. Use of configurable font size and colors
3. Highly customizable syntax highlighting
4. Undo and redo functionalities
5. Reverting of files
6. Remote editing of files
7. Search and replace text
8. Clipboard support functionalities and many more

3) Nano editor :

Saving and exiting

If you want to save the changes you've made, press Ctrl + O. To exit nano, type Ctrl + X. If you ask nano to exit from a modified file, it will ask you if you want to save it. Just press N in case you don't, or Y in case you do. It will then ask you for a filename. Just type it in and press Enter.

If you accidentally confirmed that you want to save the file but you actually don't, you can always cancel by pressing Ctrl + C when you're prompted for a filename.

Cutting and pasting

To cut a single line, paste it, you simply move the cursor to where you want to paste it and punch Ctrl + U. The line reappears. To move multiple lines, simply cut them with several Ctrl + K in a row, then paste them with a single Ctrl + U. The whole paragraph appears wherever you want it.

If you need a little more fine-grained control, then you have to mark the text. Move the cursor to the beginning of the text you want to cut. Hit Ctrl + 6 (or Alt + A). Now move your cursor to the end of the text you want to cut: the marked text gets highlighted. If you need to cancel your text marking, simply hit Ctrl + 6 again. Press Ctrl + K to cut the marked text. Use Ctrl + U to paste it.

Example Program:

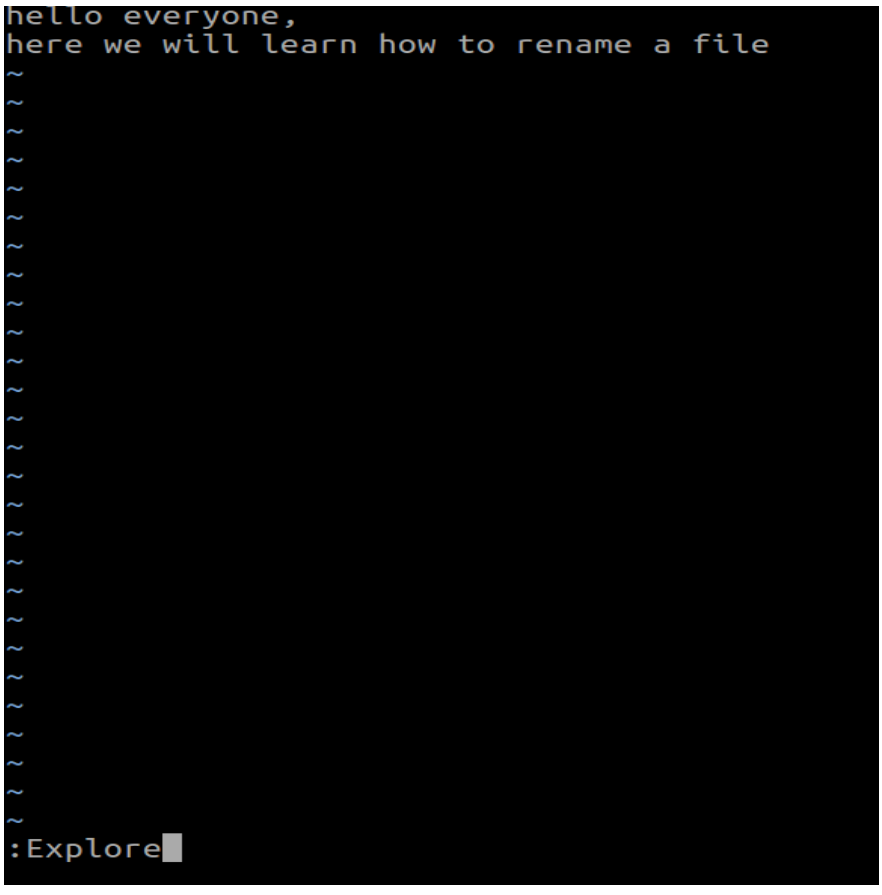
```
/* -----------------------------------------------------------------------------------------
File Name: hello_world
Description: This example demonstrate the basic hello world program using
             two different type of editor using vi/vim or gedit.
             Author: Embedded Linux Team, EC Department, CSPIT,
             CHARUSAT
-----------------------------------------------------------------------------------------*/
```

Execution Steps for rename vi/vim editor file:

1):Firstly , we open a file named file3.

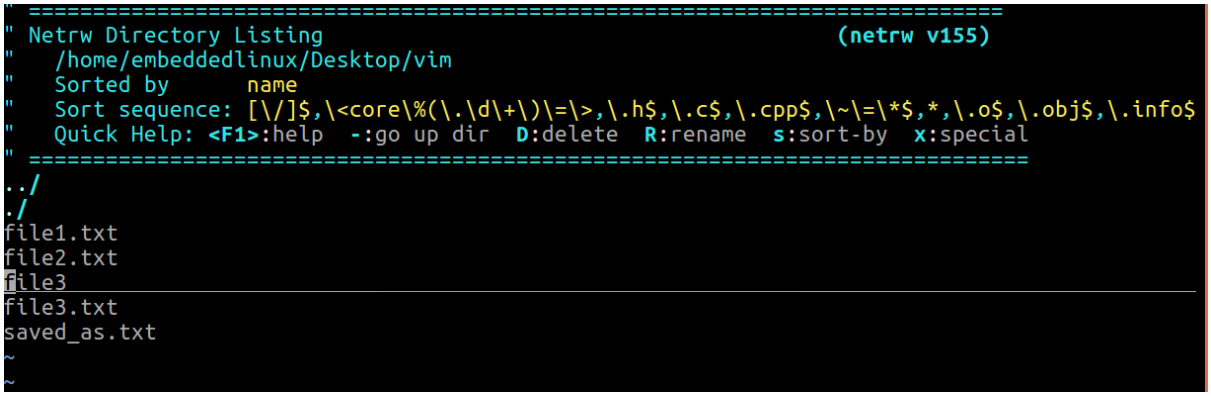
```
embeddedlinux@embeddedlinux-HP-280-G1-MT:~/Desktop/vim$ vi file3
```

Below we can see the file3 opened in vi editor.

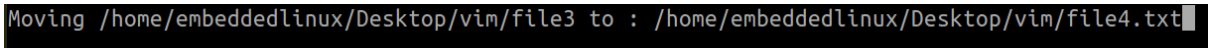


2): Now we fire “Explore” command to go to the advance function menu.

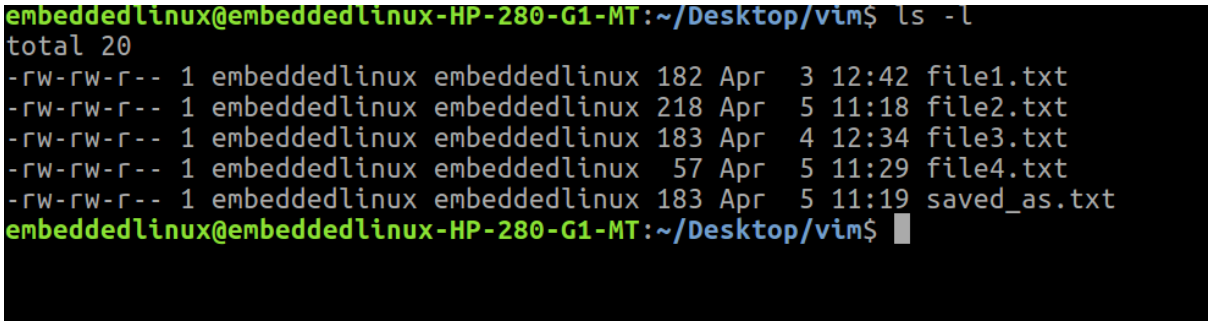
Below we can see the advance menu and also the files present in the current working directory>
Here we need to bring the long line cursor to the file that we want to rename.



After that as shown in the menu options, we have to press “shift + r”.
Once we do that we will see a path of the file as
Moving /home/embeddedlinux/Desktop/vim/file3
To /home/embeddeedlinux/Desktop/vim/file4.txt
In second path “file4.txt” is the new name that we want to assign to the old file “file3”
And the we simply press Enter.

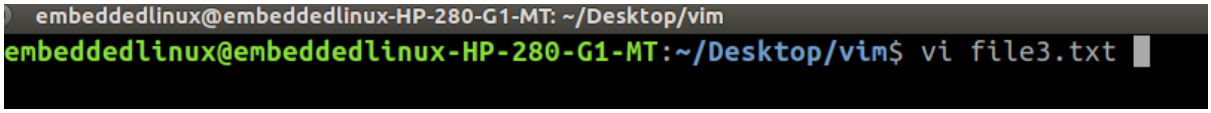


After pressing enter, we fire the command “:q” to get back to our bash screen and then we simply fire “ls –l“ command to chek our rem=named file As “ file4.txt “.



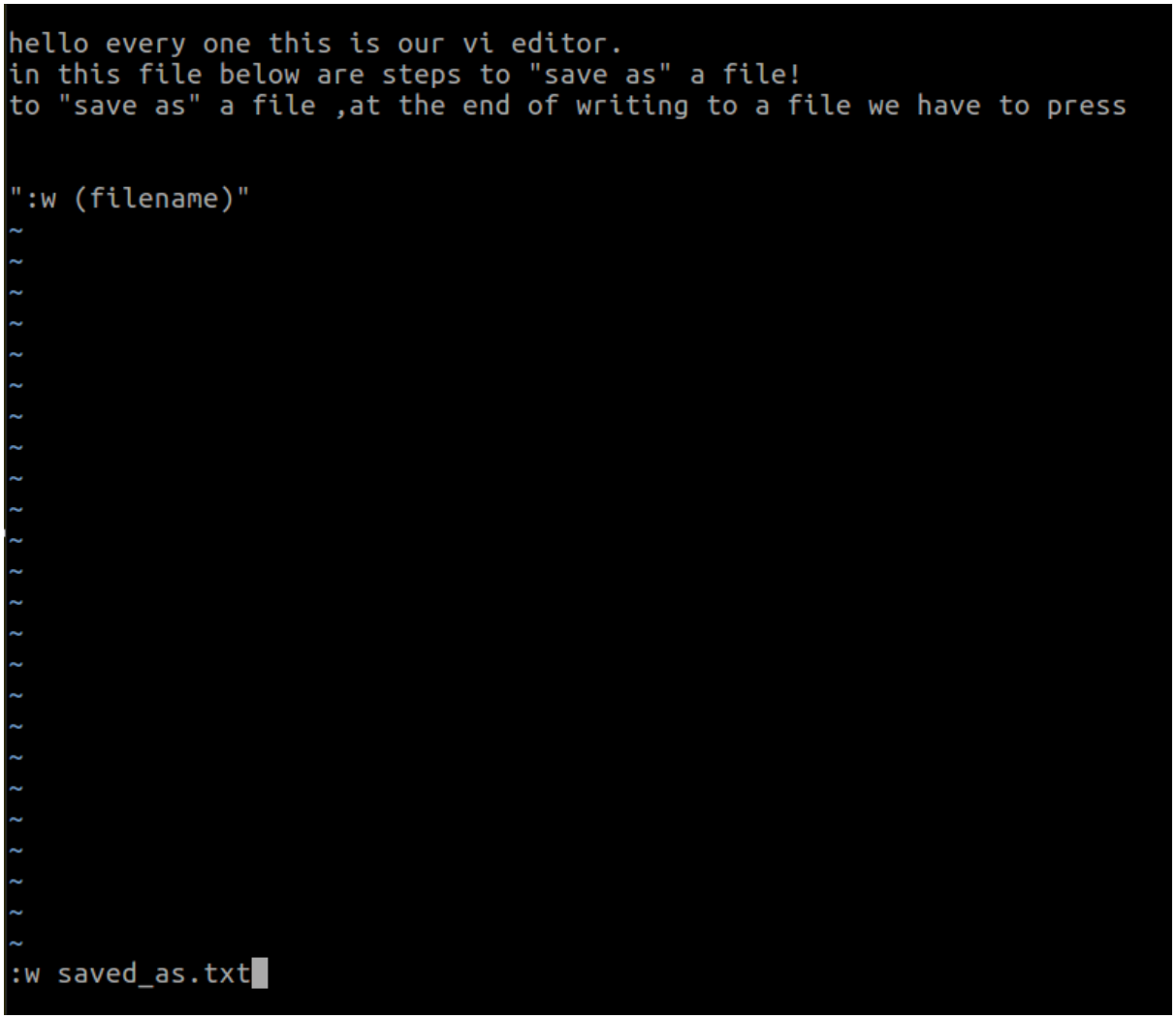
Execution Steps to save updated file as another file (just like "save as" option) in vi/vim editor file:

1): Opening a file named file3.txt.

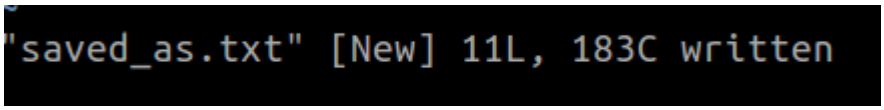


2): Now after updating the file and to save it as another file with different name, we give the command
" :w (new_file_name) "

This will save the file with new name which we use in above syntax.



3) Here we have given name “saved_as.txt” and after pressing Enter we will see the status at command line as below.

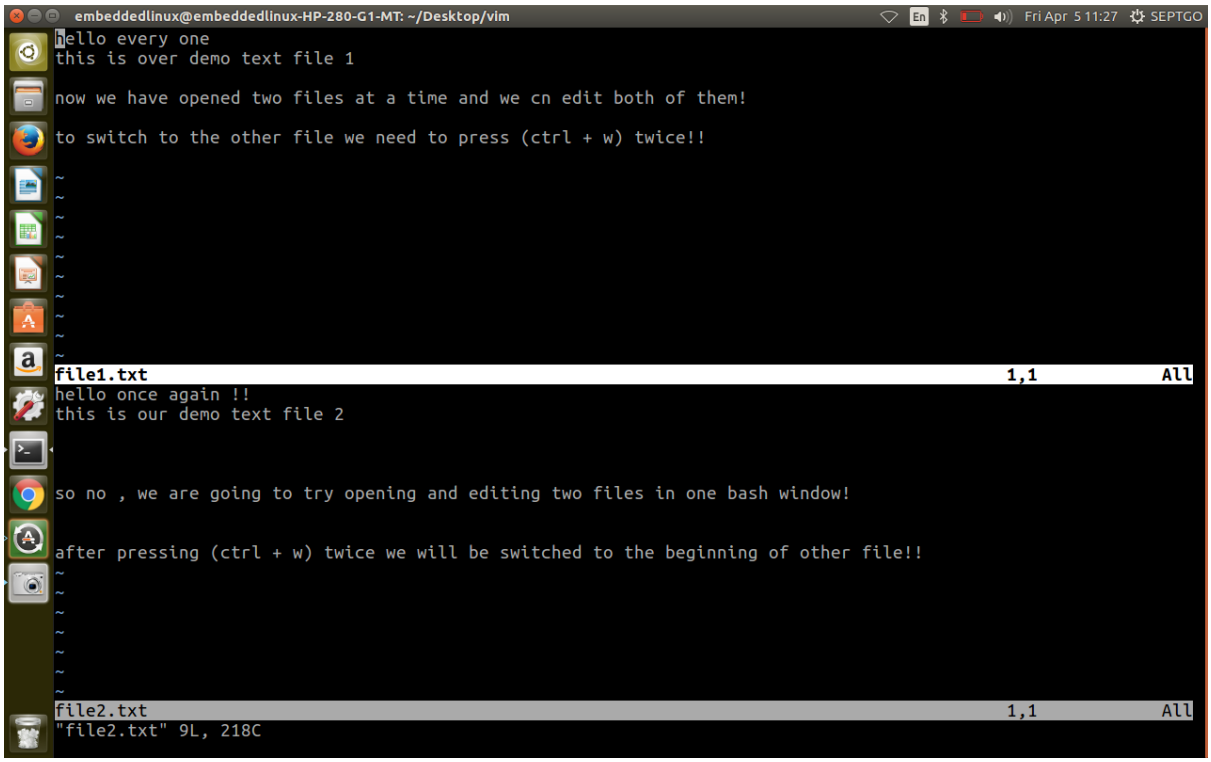


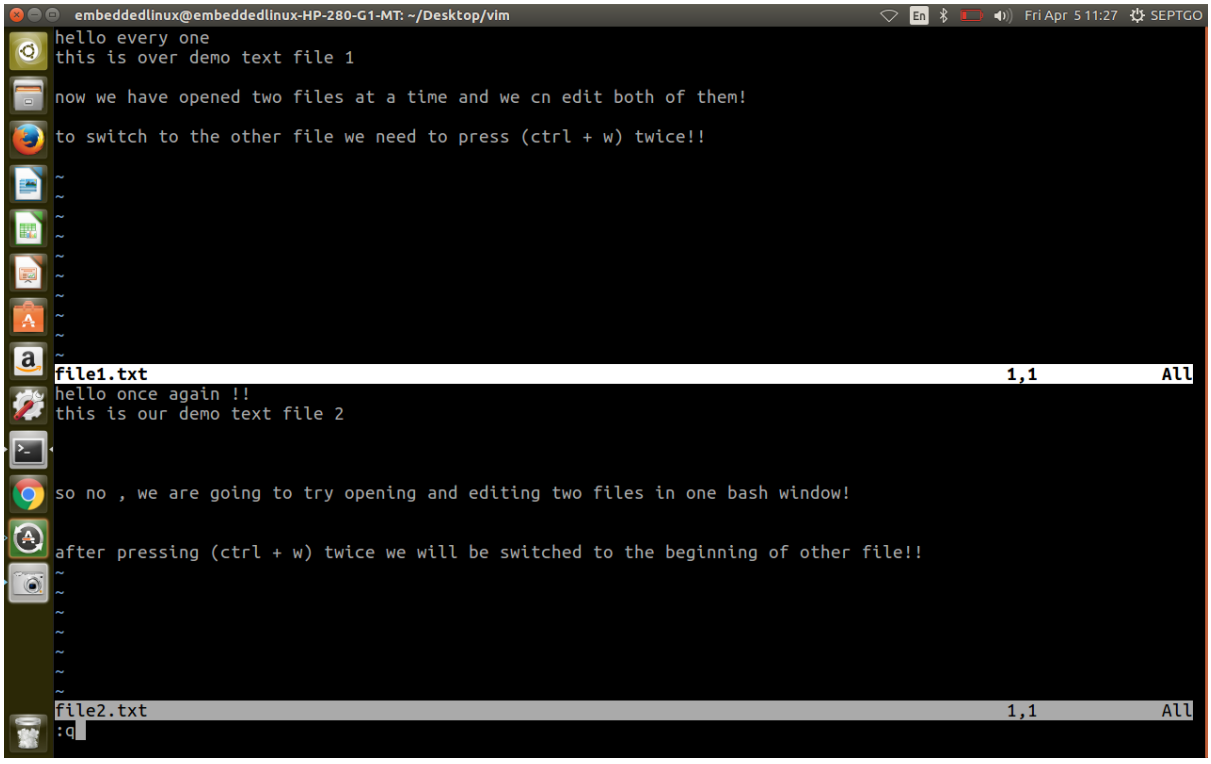
4) Now we simply us “ls – l” command to see our newly saved file as below.

Execution Steps for opening multiple files in vi/vim editor :

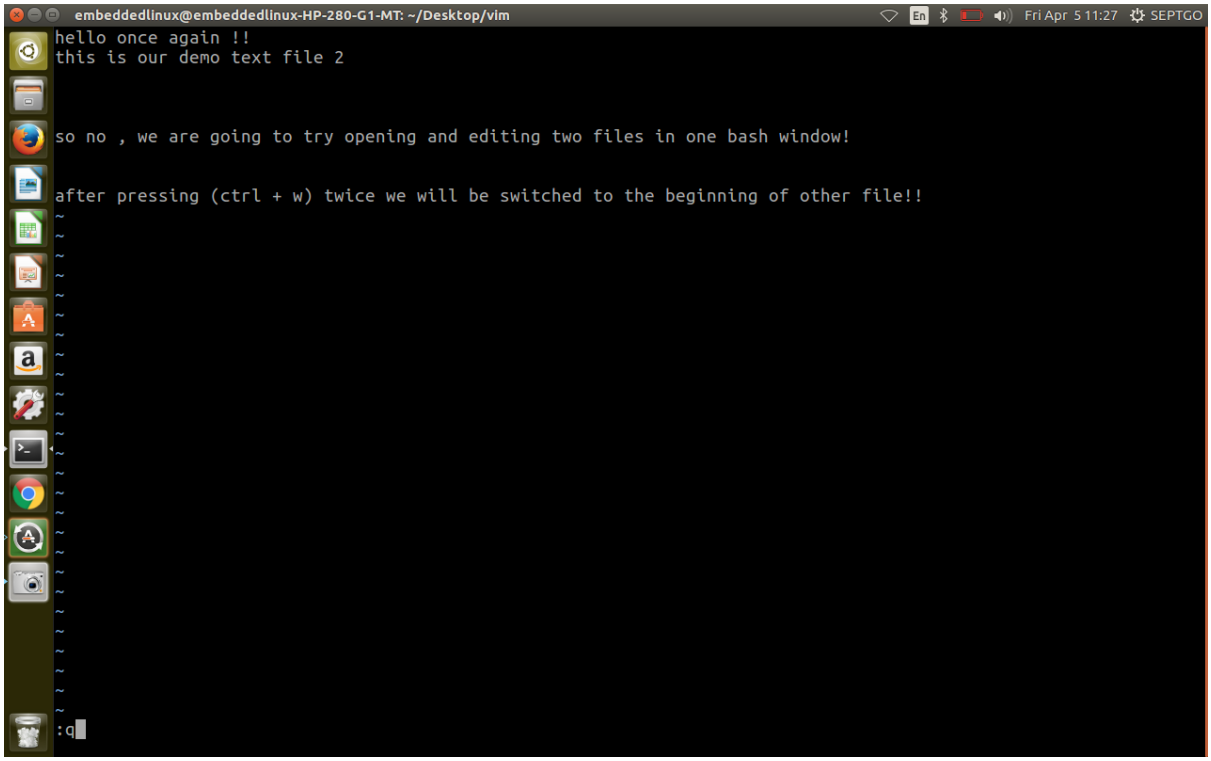
1) For opening multiple files to be edited in vim editor we use following command:
“ vim –o (file1) (file2)”
Here our file 1 is file1.txt
And file 2 is file2.txt

2) Here we can see two files opened together in vim editor
now whenever we want to switch the files to edit we press “ctrl + w” twice!. When we hit the button combination twice we will get our cursor to the beginning of other file.





3) Now when we type ":q" command to quit the file and execute it, firstly file1.txt will quitted and then again we execute the same command to quit the second file file2.txt.

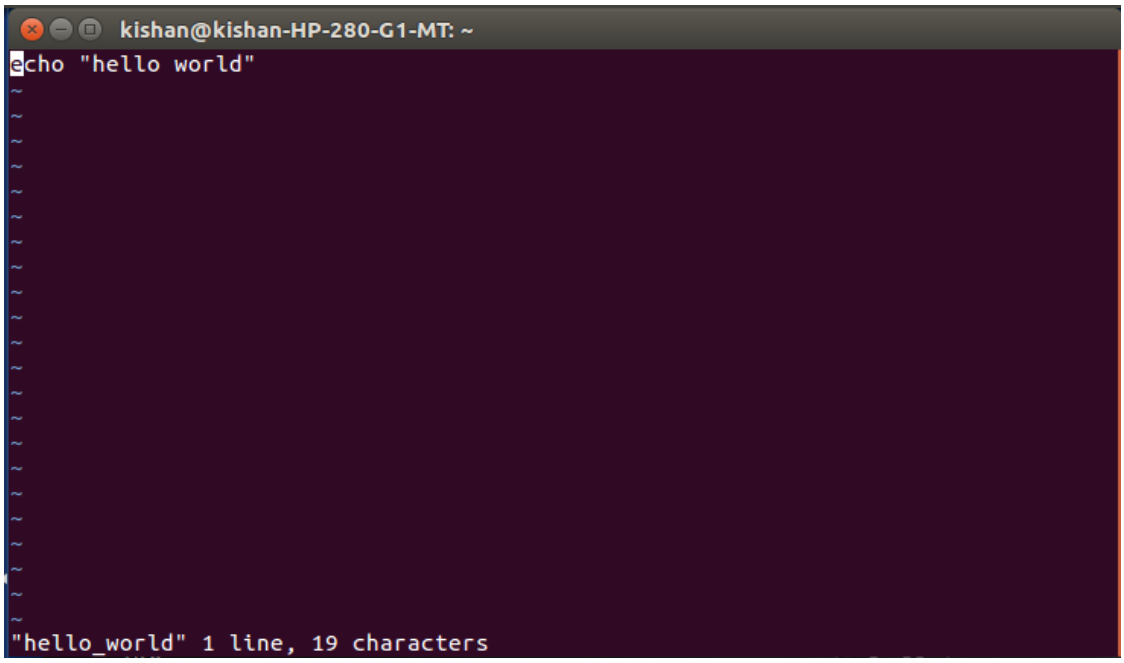


Execution Steps for vi/vim editor and result :

1) Here we first create a file by " vi hello_world"

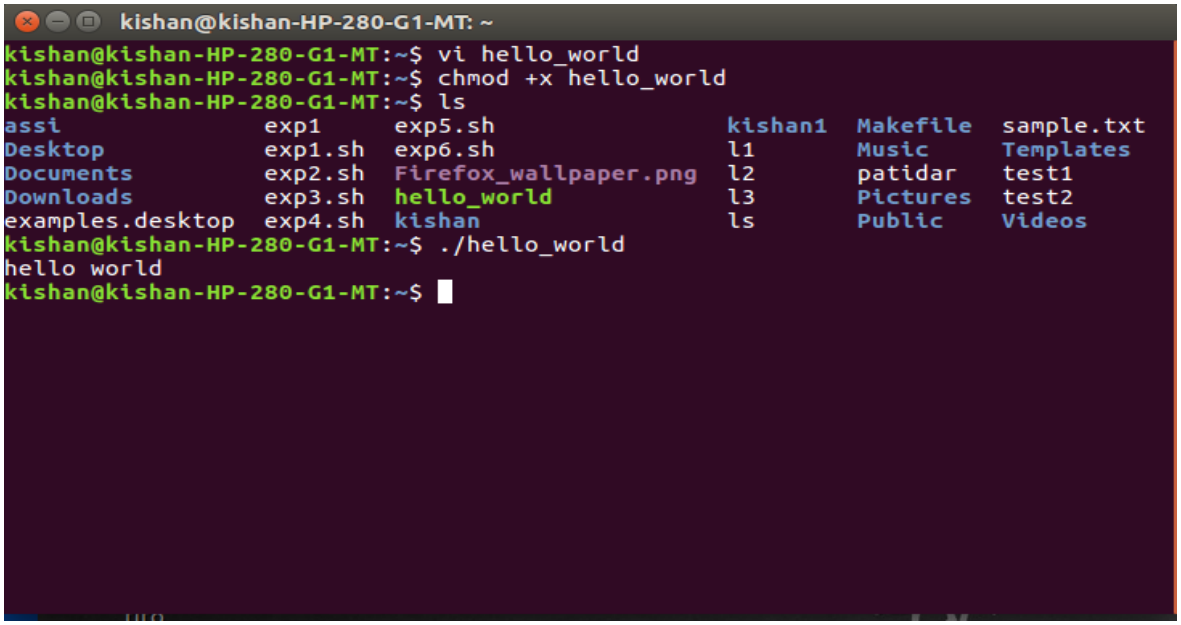
In that we give a command:

echo "hello world" as shown below.



2) Later then after saving and quitting the file, we make executable file of our original file by

“chmod –x hello-world”



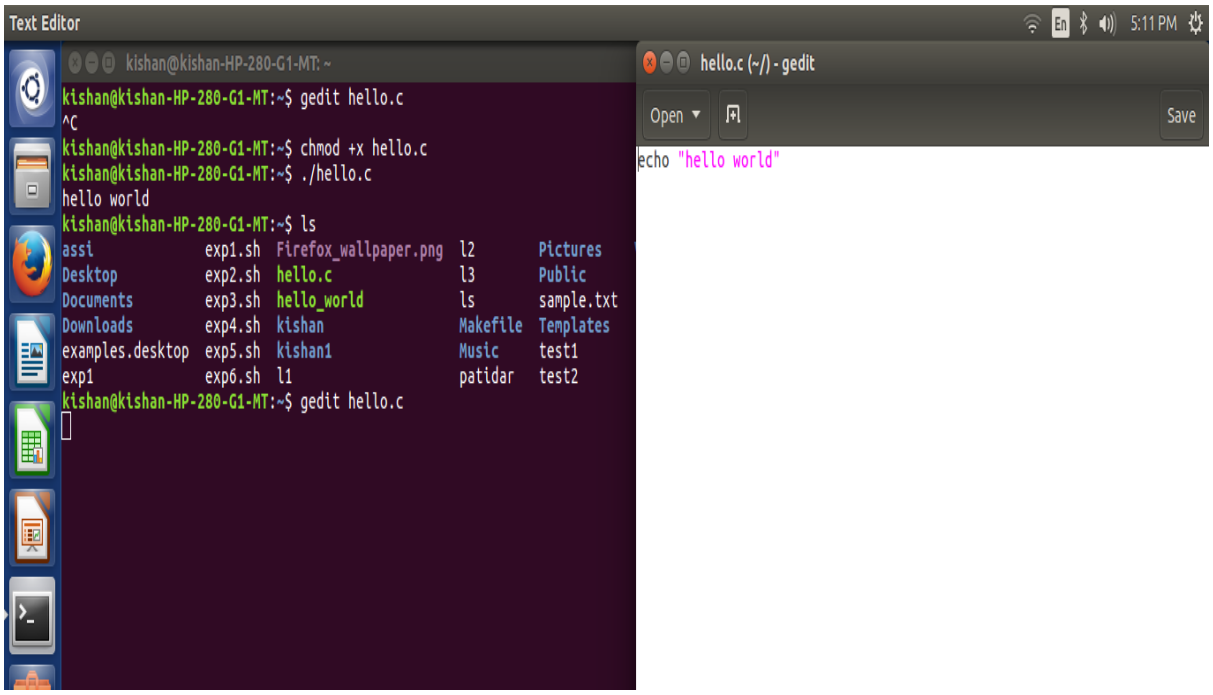
3) Then by listing the files we can see the green coloured " hello_world" that is our executable file.

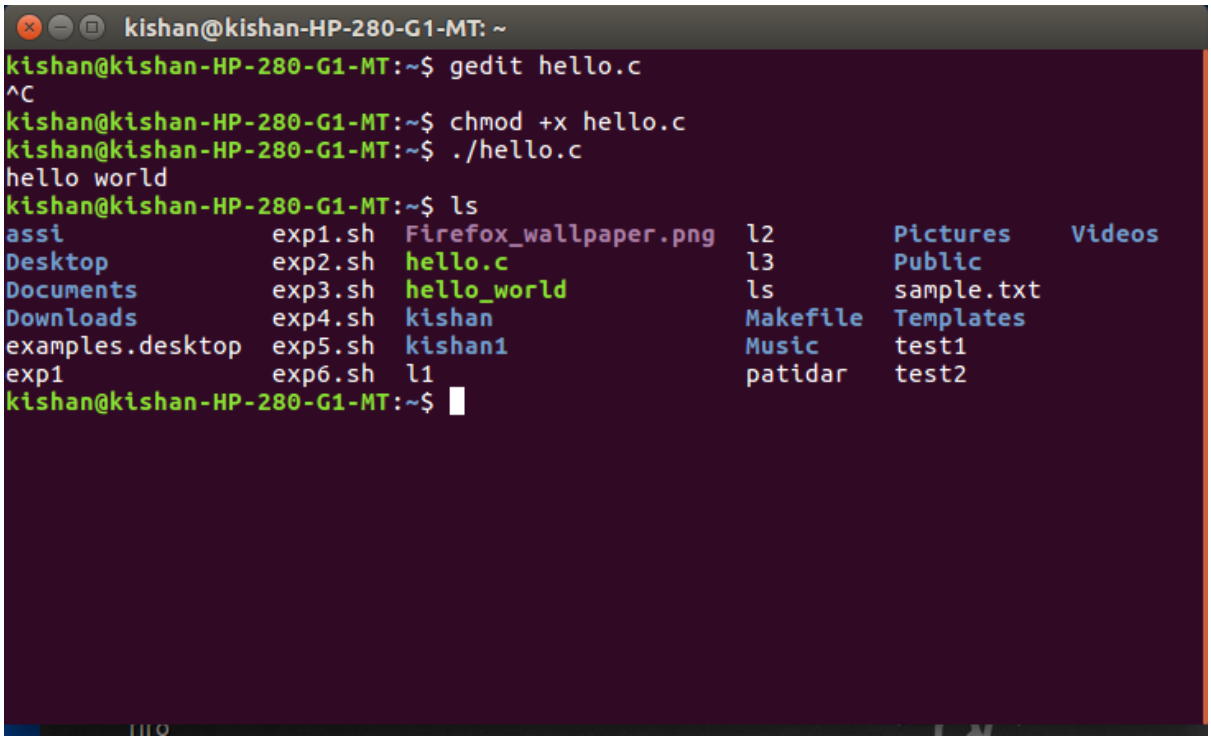
Now we execute that file by typing:

" ./hello_world

And see the output " hello world" , message in the linux display as shown in above screenshot.

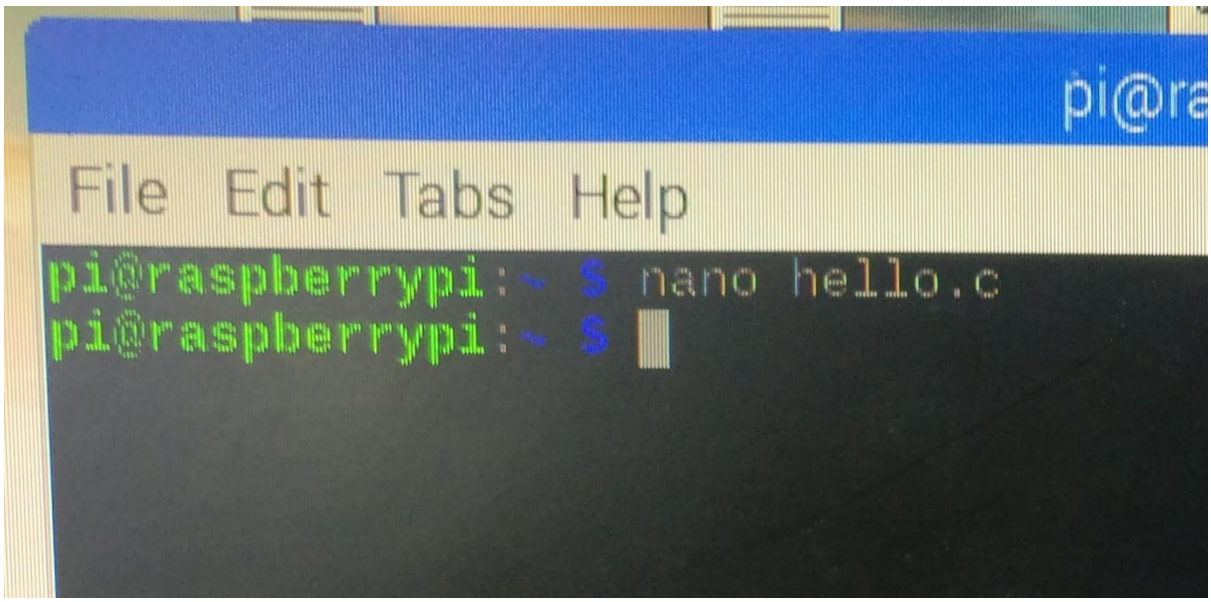
Execution Steps for gedit editor and result :



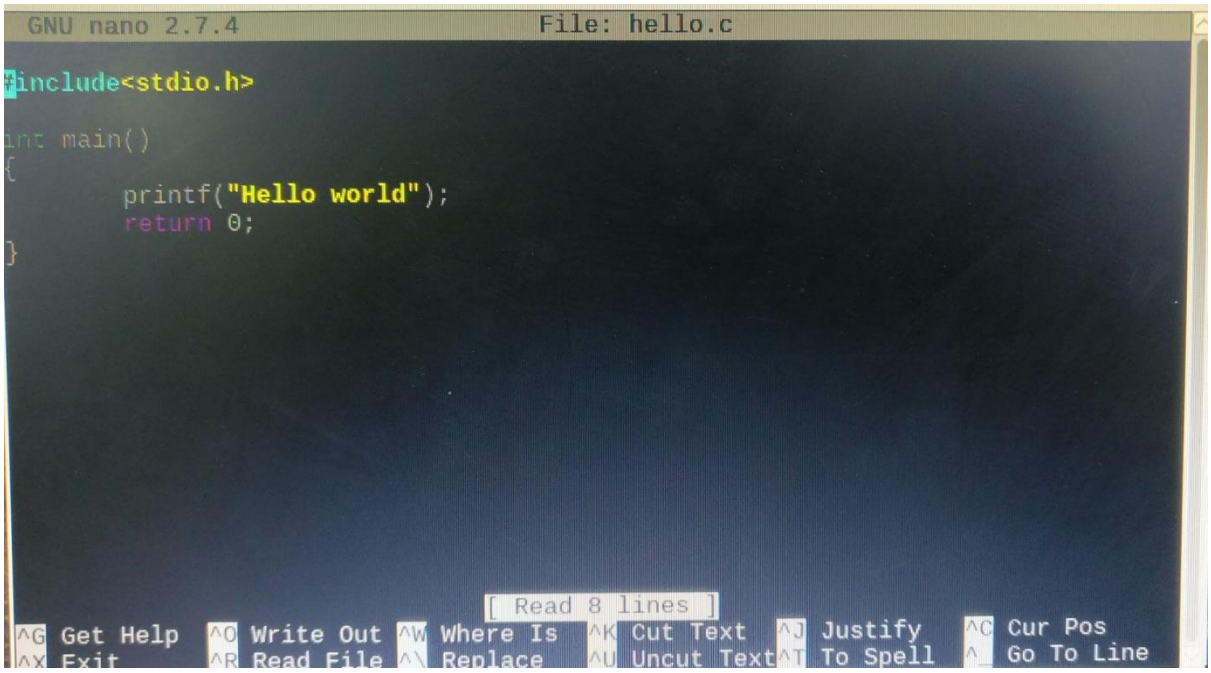


Execution steps for Nano editor and results:

1) Opening the file hello.c in Nano editor

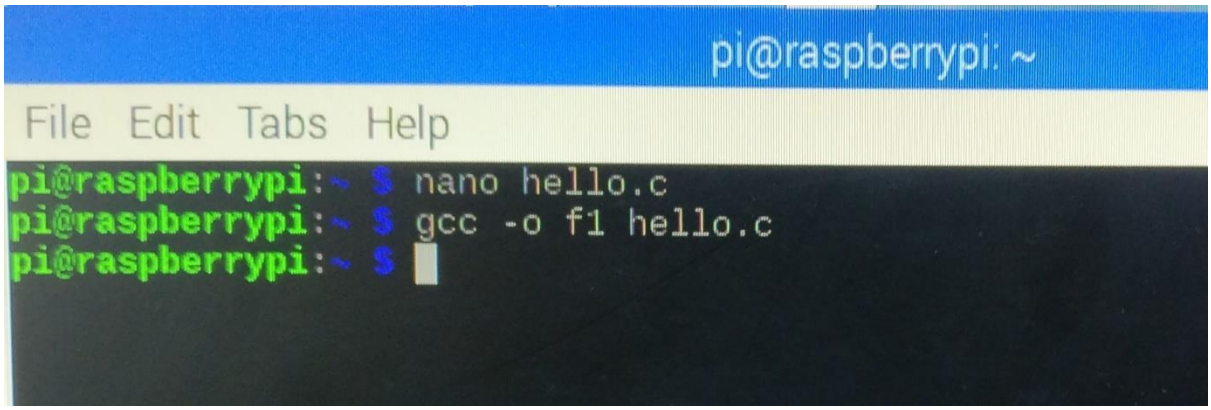


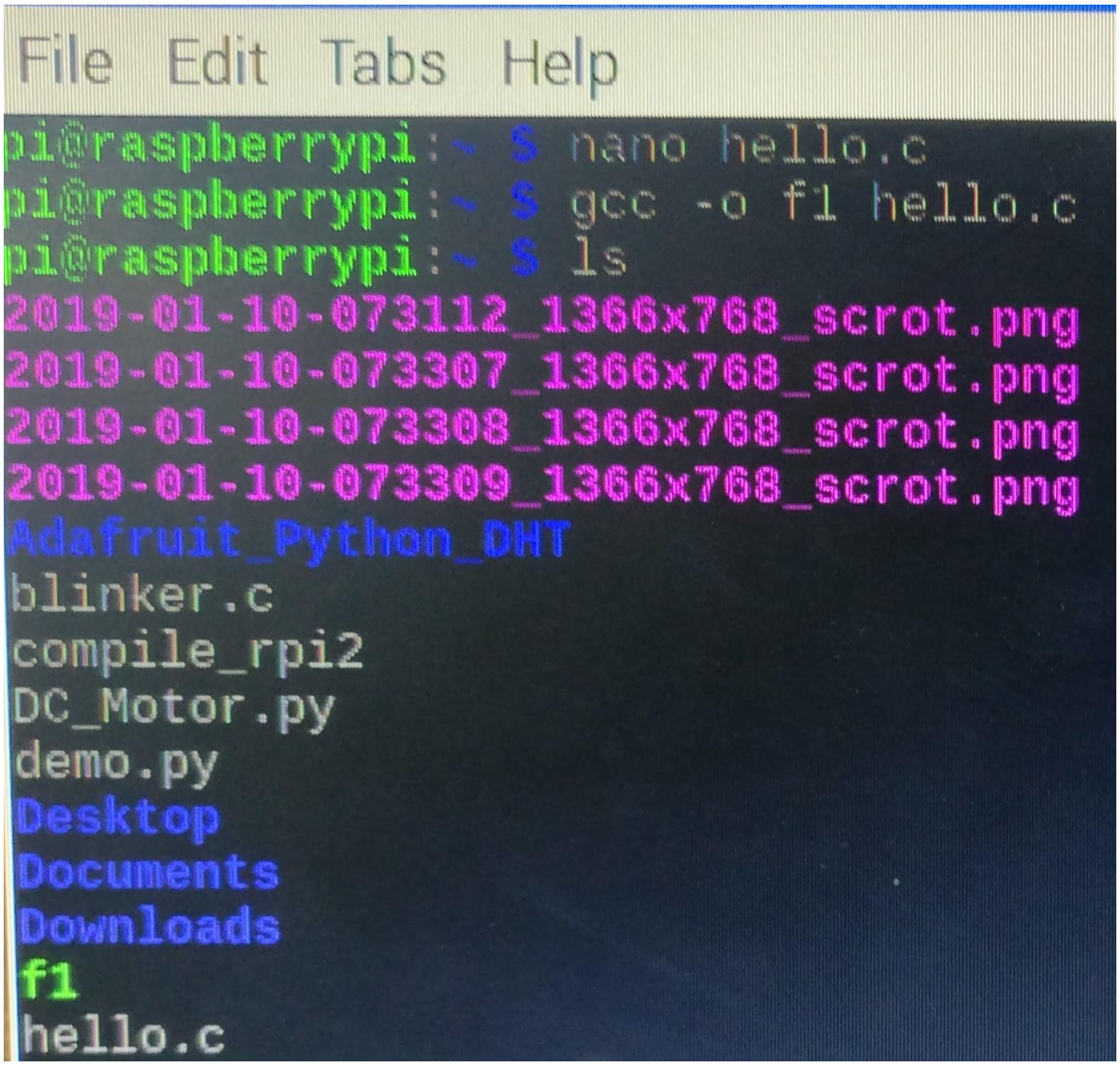
2) For saving file press CTRL+O and then press CTRL+X after that press enter to exit.



3) Creating the executable file of hello.c by following command:

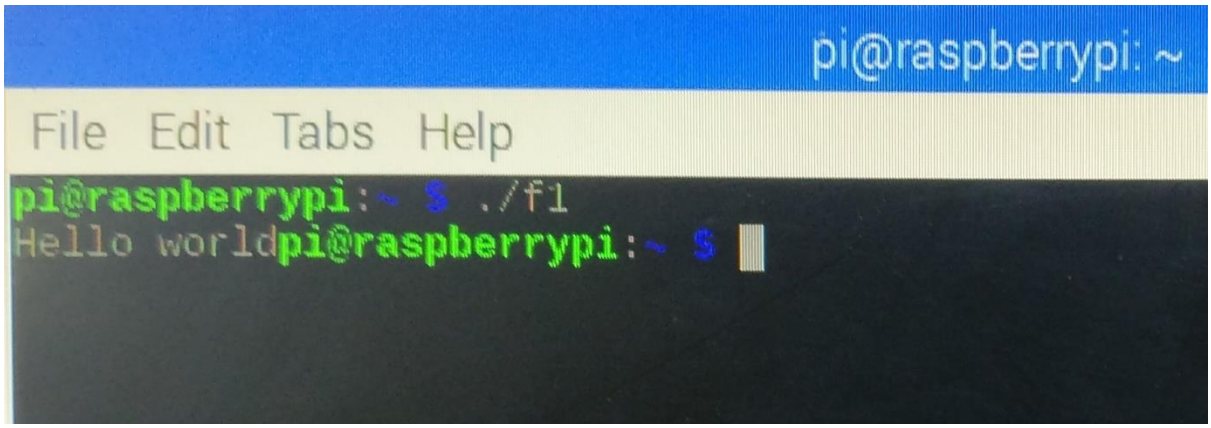
gcc –o {executable filename} {file name.c}





Output:

By using ./{Executable filename}

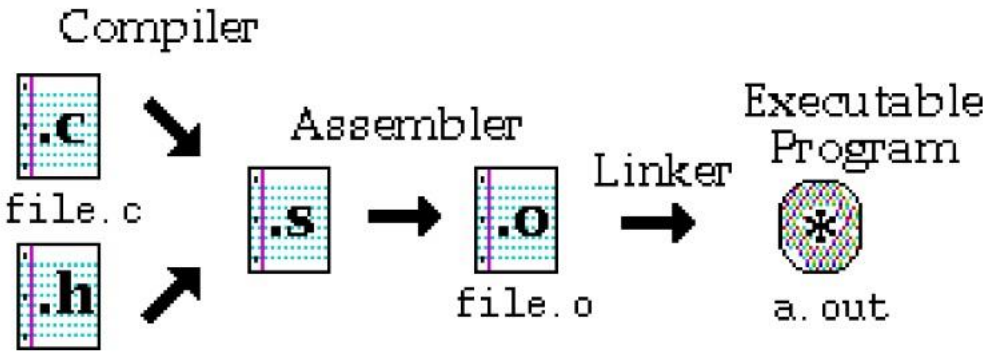


Experiment No-4

Aim: Implementation of Makefile concept.

Theory:

The “make” utility in Unix is one of the original tools designed by S. I. Fieldman of AT&T Bell labs circa 1977. There are many version. In Unix, when you type the command “make” the operating system looks for a file called either “makefile” or “Makefile”. This file contains a series of directives that tell the “make” utility how to compile your program and in what order. Compiling a small C program requires at least a single .c file, with .h files as appropriate. There are 3 steps to obtain the final executable program, as shown:



1. **Compiler stage:** All C language code in the .c file is converted into a lower-level language called Assembly language; making .s files.

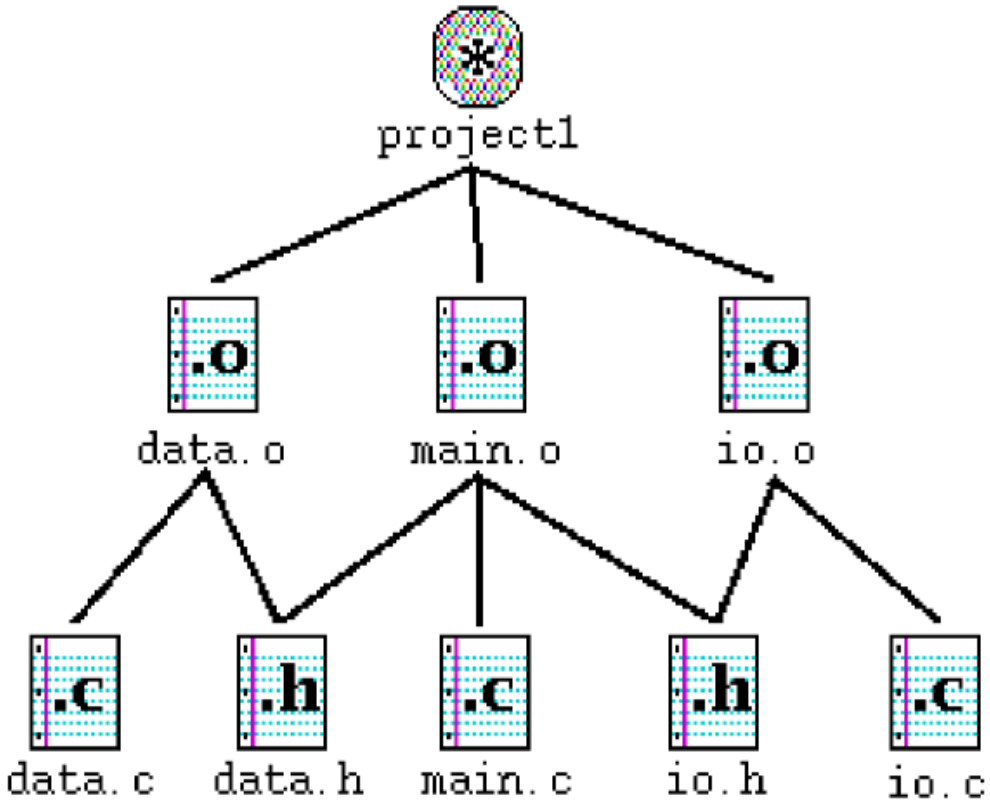
2. **Assembler stage:** The assembly language code made by the previous stage is then converted into object code which are fragments of code which the computer understands directly. An object code file ends with .o.

3. **Linker stage:** The final stage in compiling a program involves linking the object code to code libraries which contain certain "built-in" functions, such as printf. This stage produces an executable program, which is named a.out by default.

Each file will be associated with a list of other files by which it is dependent. This is called a dependency line. ”Make” creates programs according to the file dependencies. For example, in order to create an object file, program.o, we require at least the file program.c. (There may be other dependencies,

such as a .h file). The make program gets its dependency "graph" from a text file called makefile or Makefile which resides in the same directory as the source files.

Dependency graph:



Makefile uses the following format:

target : source file(s)
command (must be preceded by a tab)

A target given in the Makefile is a file which will be created or updated when any of its source files are modified. The command(s) given in the subsequent line(s) (which must be preceded by a tab character) are executed in order to create the target file.

Listing dependencies:

```
project1: data.o main.o io.o
        cc data.o main.o io.o -o project1
data.o: data.c data.h
```

```
	cc -c data.c

main.o: data.h io.h main.c

	cc -c main.c

io.o: io.h io.c

	cc -c io.c
```

Advantages:

We can place the commands to compile a program in a Unix script but this will cause ALL modules to be compiled every time. The “make” utility allows us to only compile those that have changed and the modules that depend upon them.

Make checks the modification times of the files, and when ever a file becomes "newer" than something that depends on it, (in other words, modified) it runs the compiler accordingly.

CMake is with supportability of cross-platform make. **CMake** invokes the right sequence of commands for type of target. Therefore, there is no explicit specification of commands like $(CC).

List of System calls:

CC or **GCC** command is stands for C Compiler, usually an alias command to *gcc* or *clang*. As the name suggests, executing the cc command will usually call the gcc on Linux systems.

Example Programs:

PROGRAM 1:

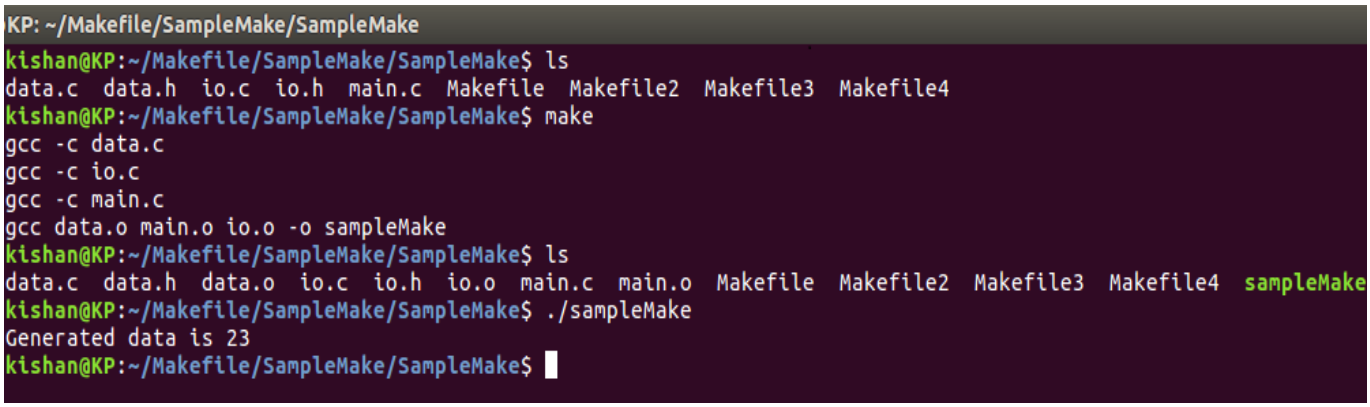
```
sampleMake: data.o io.o main.o

	gcc data.o main.o io.o -o sampleMake

data.o: data.c data.h

	gcc -c data.c

main.o: data.h io.h main.c

	gcc -c main.c

io.o: io.c io.h gcc -c io.c
```

```
clean: rm *.o sampleMake
```

Execution steps for program 1 to 3:

1) Make
2) ./sampleMake

OUTPUT 1 :



Drawbacks:

Requirement of adding manually the object as well as 'C' files regardless of their number.

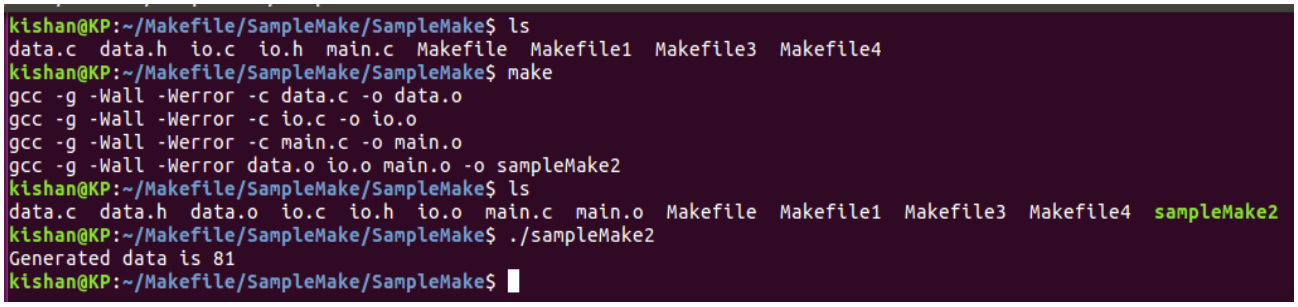
This manual addition makes coding a tedious job.

PROGRAM 2:

```
CC = gcc

CFLAGS = -g -Wall -Werror

        #-Wall is for enabling set of all warnings

        #- Werror means every warning is treated as an error

OUTPUT = sampleMake2

OBJFILES = data.o io.o main.o

${OUTPUT}: ${OBJFILES}

     $(CC) $(CFLAGS) $(OBJFILES) -o $(OUTPUT) %.o: %.c
```

```
	$(CC) $(CFLAGS) -c $< -o $@

clean: rm *.o $(OUTPUT)
```

OUTPUT 2:



Drawbacks:

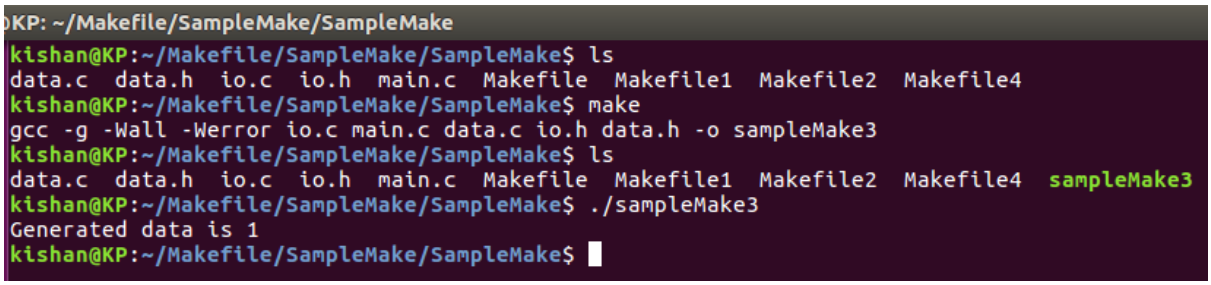
All the object files need to be manually listed in “OBJFILES” object that has been created.

One needs to type and verify all the files so as to include them regardless of their count.

PROGRAM 3:

```
CC = gcc

CFLAGS = -g -Wall -Werror

	#-Wall is for enabling set of all warnings

	 #- Werror means every warning is treated as an error

	#Should be equivalent to your list of C files, if you
	#don't build selectively

SRC=$(wildcard *.c *.h)

sampleMake3: ${SRC}

	$(CC) $(CFLAGS) $^ -o $@

clean: rm sampleMake3
```

OUTPUT 3:



Drawbacks:

We need to create Executable file every time whenever the makefile will run, which makes the job tedious.

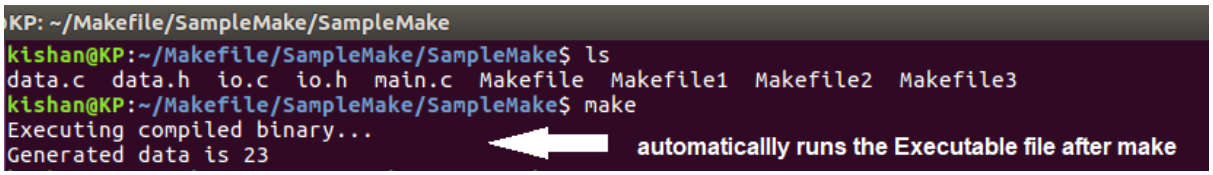
PROGRAM 4:

```
CC = gcc

CFLAGS = -g -Wall -Werror

     # Should be equivalent to your list of C files, if you
     #don't build selectively

SRC=$(wildcard *.c *.h)

sampleMake4: compile

     @echo "Executing compiled binary..."              #message
to terminal

     @./$@

compile: ${SRC}

     @$(CC) $(CFLAGS) $^ -o sampleMake4

clean: rm sampleMake4 || true
```

Execution steps for program 4:

1) Make

OUTPUT 4:



Advantages:

We do not need to create Executable file every time whenever the makefile will run the executable file will be generated.

Experiment No:5

Aim: To implement Process Management in Linux.

Theory:

A program is a series of instruction that tells the computer what to do. When we run a program, those instructions are copied into memory and space is allocated for variables and other stuff which is required to manage its execution. This running instance of a program is called a process.

There are basically 2 types of processes:

- Zombie Process
- Orphan Process

Zombie Process:

A process which has finished the execution but still has entry in the process table to report to its parent process is known as a zombie process. A child process always first becomes a zombie before being removed from the process table. The parent process reads the exit status of the child process which reaps off the child process entry from the process table.

Advantages: A zombie is created when a parent process does not use wait() system call after child dies to read it's data to exit data.

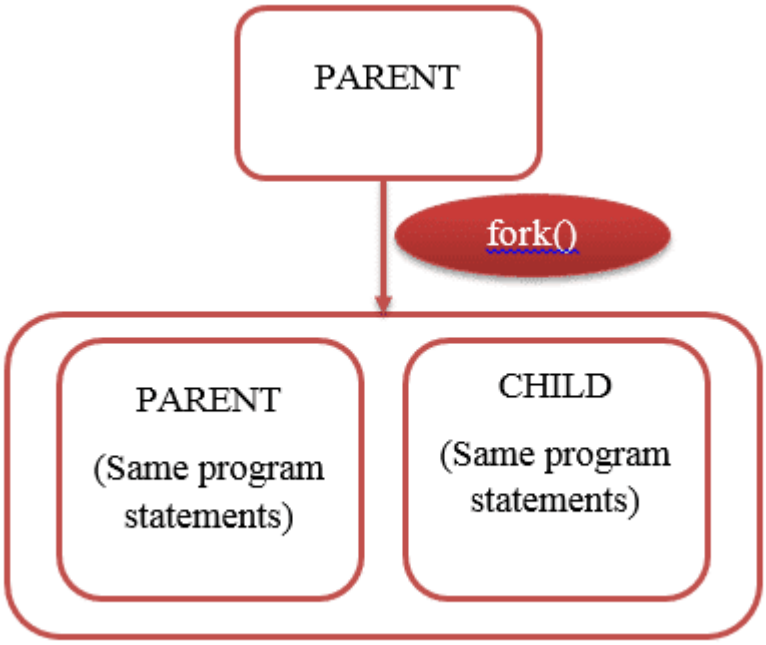
Disadvantages: The process table is a finite resource, and when it gets completely 'clogged' by zombies, the system won't be able to start any new processes.

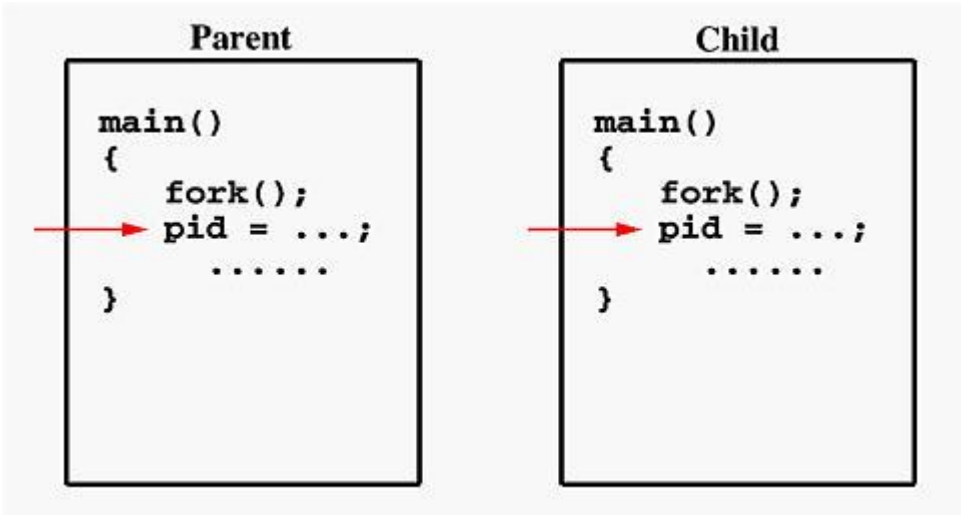
Orphan process:

An orphan process is a process whose parent process no more exists i.e. either finished or terminated though it remains running itself. In Linux operating system any orphaned process will be immediately adopted by the special init system process. This operation is called re-parenting and occurs automatically. Even though technically the process has the init process as its parent, it is still called an orphan process since the process that originally created it no longer exists.

Advantages: Orphan is a child process reclaimed by init when the original parent process terminates before the child.

Disadvantages: Orphan process still **takes resources,** and having too many orphan process will overload the init/systemd process.





List of System calls:

fork( )	System call fork() is used to create processes. It takes no arguments and returns a process ID. The purpose of fork() is to create a new process, which becomes the child process of the caller. After a new child process is created, both processes will execute the next instruction following the fork() system call. If the call to fork() is executed successfully, Linux will make two identical copies of address spaces, one for the parent and the other for the child. When the main program executes fork(), an identical copy of its address space, including the program and all data, is created. System call fork() returns the child process ID to the parent and returns 0 to the child process.
getpid( )	Returns the pid of the invoking processes.

Program 1:

*/----------

File Name: process.c

Description: In this example program the fork( ) system call creates two identical processes namely parent process and child process. Here, the fork() returns child PID to parent process and returns 0 to child process. As soon as parent process is kill by the system the child process becomes orphan and is handled by any other process or init process.

Author: Embedded Linux Team, EC Department, CSPIT, CHARUSAT

Date & Time: 20th March 2019 & 10:10 am

----------*/

```
#include <stdio.h>
#include <sys/wait.h>
#include <sys/types.h>
#include <unistd.h>
void main()
{
     int id;
*/fork() system call is used to create two identical copies of
parent process and child process*/
     if(fork())
     {
          while(1)
          {
               */getpid() function is used to retrieve the
               process ID assign to the process*/
               printf("in parent,PID=%d\n",getpid());
               sleep(5);
          }
     }
     else
     {
          while(1)
          {
               printf("in child,PID=%d\n",getpid());
               sleep(10);
          }
     }
}
```

Execution Steps:

```
drashti@ThinkPad-Edge-E440: ~/Desktop/Process
drashti@ThinkPad-Edge-E440:~$ cd Desktop/
drashti@ThinkPad-Edge-E440:~/Desktop$ cd Process/
drashti@ThinkPad-Edge-E440:~/Desktop/Process$ gcc -o pr process.c
drashti@ThinkPad-Edge-E440:~/Desktop/Process$ ls
pr  process1.c  process.c
drashti@ThinkPad-Edge-E440:~/Desktop/Process$ ./pr
```

Result:

```
drashti@ThinkPad-Edge-E440:~/Desktop/Process$ ./pr
in parent,PID=3287
in child,PID=3288
in parent,PID=3287
in child,PID=3288
in parent,PID=3287
```

Now, open another terminal and fire ps –el. This command is used to provide information about the currently running processes, including their process identification numbers (PIDs), UID, TTY,TIME and command.

```
1 S     0  2897     2  0  80   0 -     0 -       ?        00:00:00 kworker/2:0
1 S  1000  3022  1280  0  80   0 -  1088 hrtime ?        00:00:00 pr
1 S     0  3085     2  0  80   0 -     0 -       ?        00:00:00 kworker/1:0
1 S     0  3169     2  0  80   0 -     0 -       ?        00:00:00 kworker/3:0
1 S     0  3172     2  0  80   0 -     0 -       ?        00:00:00 kworker/0:0
4 S  1000  3175  2752  0  80   0 - 396665 poll_s ?        00:00:00 Web Content
4 S     0  3207     1  0  80   0 -  3918 -       ?        00:00:00 systemd-host
0 R  1000  3221  1280  1  80   0 - 167574 poll_s ?        00:00:00 gnome-termin
0 S  1000  3228  3221  0  80   0 -  7438 wait   pts/19   00:00:00 bash
0 S  1000  3287  3228  0  80   0 -  1088 hrtime pts/19   00:00:00 pr
1 S  1000  3288  3287  0  80   0 -  1088 hrtime pts/19   00:00:00 pr
1 S     0  3307     2  0  80   0 -     0 -       ?        00:00:00 kworker/u16:
0 S  1000  3318  3221  1  80   0 -  7438 wait   pts/20   00:00:00 bash
```

Now as soon as you kill the parent process, child process is handled by another random process or init process.

```
drashti@ThinkPad-Edge-E440: ~
1 S     0  3169     2  0  80   0 -     0 -       ?        00:00:00 kworker/3:0
1 S     0  3172     2  0  80   0 -     0 -       ?        00:00:00 kworker/0:0
4 S  1000  3175  2752  0  80   0 - 396665 poll_s ?        00:00:00 Web Content
4 S     0  3207     1  0  80   0 -  3918 -       ?        00:00:00 systemd-hostna
0 R  1000  3221  1280  1  80   0 - 167574 poll_s ?        00:00:00 gnome-terminal
0 S  1000  3228  3221  0  80   0 -  7438 wait    pts/19   00:00:00 bash
0 S  1000  3287  3228  0  80   0 -  1088 hrtime  pts/19   00:00:00 pr
1 S  1000  3288  3287  0  80   0 -  1088 hrtime  pts/19   00:00:00 pr
1 S     0  3307     2  0  80   0 -     0 -       ?        00:00:00 kworker/u16:0
0 S  1000  3318  3221  1  80   0 -  7438 wait    pts/20   00:00:00 bash
4 R  1000  3328  3318  0  80   0 -  8996 -       pts/20   00:00:00 ps
drashti@ThinkPad-Edge-E440:~$ kill 3287
drashti@ThinkPad-Edge-E440:~$
drashti@ThinkPad-Edge-E440: ~/Desktop/Process
in parent,PID=3287
in parent,PID=3287
in child,PID=3288
in parent,PID=3287
Terminated
drashti@ThinkPad-Edge-E440:~/Desktop/Process$ in child,PID=3288
in child,PID=3288
in child,PID=3288
in child,PID=3288
in child,PID=3288
in child,PID=3288
```

```
1 S     0  2897     2  0  80   0 -     0 -       ?        00:00:00 kwor
1 S  1000  3022  1280  0  80   0 -  1088 hrtime ?        00:00:00 pr
1 S     0  3085     2  0  80   0 -     0 -       ?        00:00:00 kwor
1 S     0  3169     2  0  80   0 -     0 -       ?        00:00:00 kwor
1 S     0  3172     2  0  80   0 -     0 -       ?        00:00:00 kwor
4 S  1000  3175  2752  0  80   0 - 396665 poll_s ?        00:00:00 Web
0 S  1000  3221  1280  1  80   0 - 167778 poll_s ?        00:00:01 gnom
0 S  1000  3228  3221  0  80   0 -  7438 wait_w pts/19   00:00:00 bash
1 S  1000  3288  1280  0  80   0 -  1088 hrtime pts/19   00:00:00 pr
1 S     0  3307     2  0  80   0 -     0 -       ?        00:00:00 kwor
0 S  1000  3318  3221  0  80   0 -  7438 wait    pts/20   00:00:00 bash
1 S     0  3353     2  0  80   0 -     0 -       ?        00:00:00 kwor
4 S     0  3376     1  0  80   0 -  3918 -       ?        00:00:00 syst
```

Program 2:

```
*/-------------------------------------------------------------------------------------------
```

File Name: process1.c

Description: In this example program the fork( ) system call is used to create four processes, one Parent and three child processes. Here, the fork( ) returns child PID to parent process and returns 0 to child1 process. Now the child1 process also fork() and create child2 process where child2 PID is return to child1process and 0 is return to child2. Similarly,child2 process fork() and create child3 process, where child2 become parent process and child3 become child process. As soon as parent process is kill by the system the child1 process becomes orphan and is handled by any other process or init process, while child2 and child3 process are not affected.

Author: Embedded Linux Team, EC Department, CSPIT, CHARUSAT

Date & Time: 20th March 2019 & 10:10 am

```
-------------------------------------------------------------------------------*/
```

```
#include <stdio.h>
#include <sys/wait.h>
#include <sys/types.h>
#include <unistd.h>
void main()
{
      if(fork())
      {
            while(1)
            {
                  printf("in parent,PID=%d\n",getpid());
                  sleep(10);
            }
      }
      else if(fork())
      {
            while(1)
            {
                  printf("in child1,PID=%d\n",getpid());
                  sleep(10);
            }
      }
      else if(fork())
      {
            while(1)
            {
```

```
                printf("in child2,PID=%d\n",getpid());
                sleep(10);
            }
        }
        else
        {
            while(1)
            {
                printf("in child3,PID=%d\n",getpid());
                sleep(10);
            }
        }
}
```

Execution Steps:

```
drashti@ThinkPad-Edge-E440: ~/Desktop/Process
drashti@ThinkPad-Edge-E440:~$ cd Desktop/
drashti@ThinkPad-Edge-E440:~/Desktop$ cd Process/
drashti@ThinkPad-Edge-E440:~/Desktop/Process$ gcc -o pr1 process1.c
drashti@ThinkPad-Edge-E440:~/Desktop/Process$ ls
pr  pr1  process1.c  process.c
```

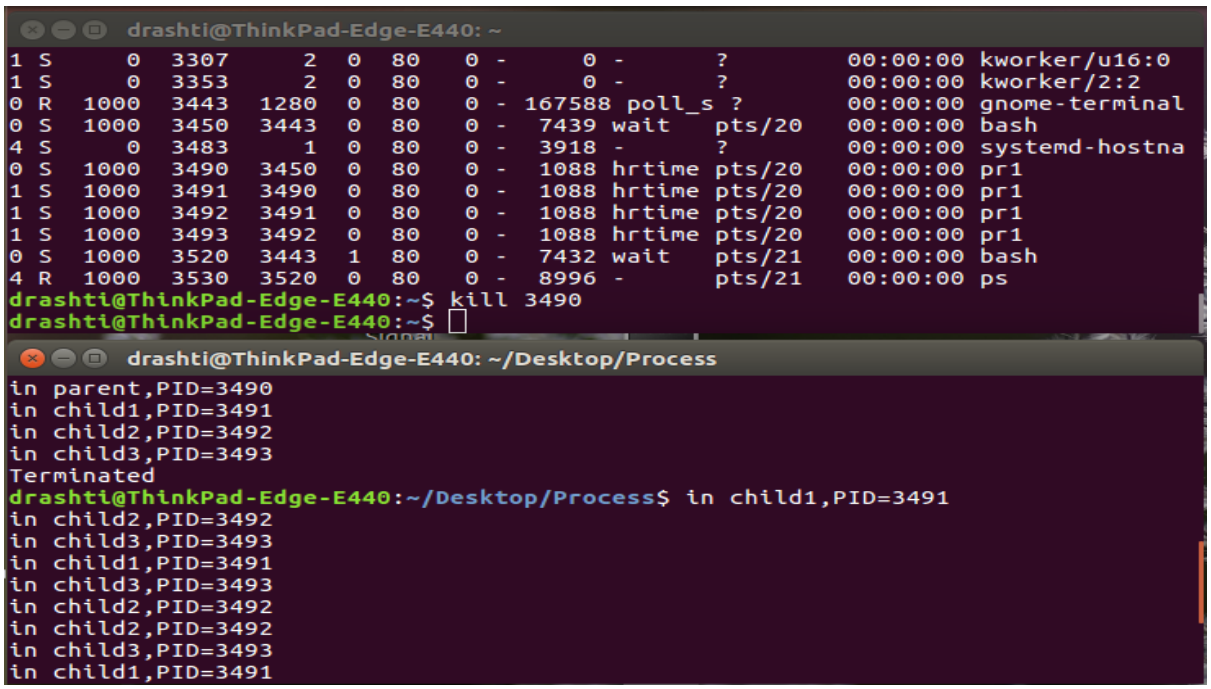
Result:

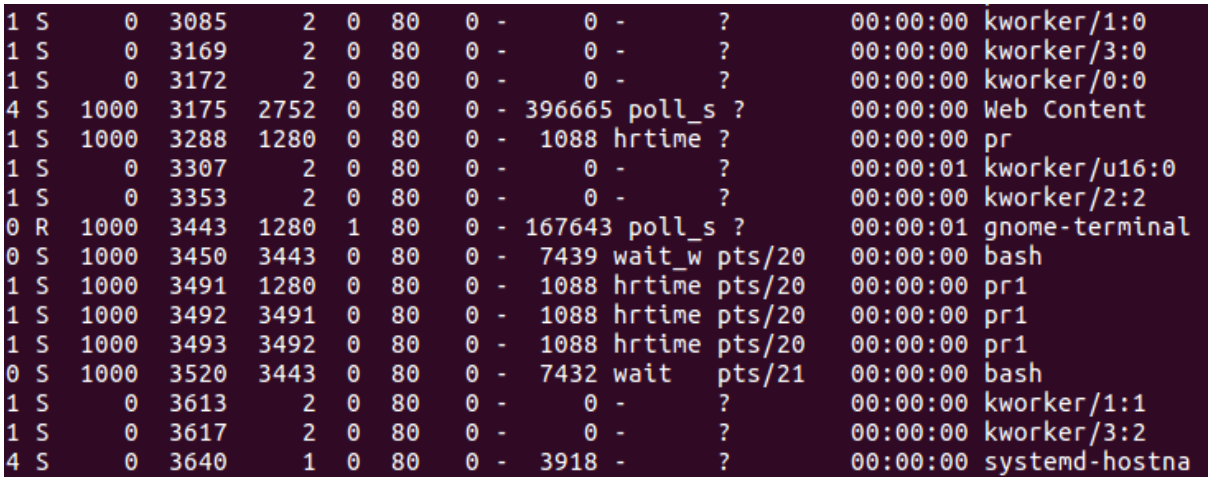
```
drashti@ThinkPad-Edge-E440:~/Desktop/Process$ ./pr1
in parent,PID=3490
in child1,PID=3491
in child2,PID=3492
in child3,PID=3493
in parent,PID=3490
in child1,PID=3491
in child2,PID=3492
in child3,PID=3493
```

Fire ps-el command on 2nd terminal which will display PID's for the currently running processes.

```
drashti@ThinkPad-Edge-E440: ~
1 S  1000  2674  1280  0  80   0 -  1088 hrtime ?        00:00:00 pr1
1 S  1000  2675  2674  0  80   0 -  1088 hrtime ?        00:00:00 pr1
1 S  1000  2676  2675  0  80   0 -  1088 hrtime ?        00:00:00 pr1
4 S  1000  2752  1280  3  80   0 - 588068 poll_s ?       00:00:20 firefox
4 S  1000  2809  2752  6  80   0 - 497901 poll_s ?       00:00:37 Web Content
1 S     0  2897     2  0  80   0 -     0 -       ?        00:00:00 kworker/2:0
1 S  1000  3022  1280  0  80   0 -  1088 hrtime ?        00:00:00 pr
1 S     0  3085     2  0  80   0 -     0 -       ?        00:00:00 kworker/1:0
1 S     0  3169     2  0  80   0 -     0 -       ?        00:00:00 kworker/3:0
1 S     0  3172     2  0  80   0 -     0 -       ?        00:00:00 kworker/0:0
4 S  1000  3175  2752  0  80   0 - 396665 poll_s ?       00:00:00 Web Content
1 S  1000  3288  1280  0  80   0 -  1088 hrtime ?        00:00:00 pr
1 S     0  3307     2  0  80   0 -     0 -       ?        00:00:00 kworker/u16:0
1 S     0  3353     2  0  80   0 -     0 -       ?        00:00:00 kworker/2:2
0 R  1000  3443  1280  0  80   0 - 167588 poll_s ?       00:00:00 gnome-terminal
0 S  1000  3450  3443  0  80   0 -  7439 wait   pts/20   00:00:00 bash
4 S     0  3483     1  0  80   0 -  3918 -       ?        00:00:00 systemd-hostna
0 S  1000  3490  3450  0  80   0 -  1088 hrtime pts/20   00:00:00 pr1
1 S  1000  3491  3490  0  80   0 -  1088 hrtime pts/20   00:00:00 pr1
1 S  1000  3492  3491  0  80   0 -  1088 hrtime pts/20   00:00:00 pr1
1 S  1000  3493  3492  0  80   0 -  1088 hrtime pts/20   00:00:00 pr1
0 S  1000  3520  3443  1  80   0 -  7432 wait   pts/21   00:00:00 bash
4 R  1000  3530  3520  0  80   0 -  8996 -       pts/21   00:00:00 ps
drashti@ThinkPad-Edge-E440:~$
```

Now as soon as you fire kill command for parent process,child1 process will becomes orphan and is handed to any other process, while child2 and child3 remain unaffected.





Experiment No-6

Aim: File descriptor and File operations in Linux

Theory:

File descriptor is integer that uniquely identifies an open file of the process. The value returned by an open call is termed a file descriptor and is essentially an index into an array of open files kept by the kernel. The array of open files kept by the kernel is called a file descriptor table. One unique file descriptors table is provided in the operating system for each process.

When any process starts, then the file descriptor table fd (file descriptor) 0,1,2 open's automatically, By default each of these 3 fd references a file table entry for a file named **/dev/tty**

Descriptive Name	Short Name	File descriptor no.	Description
Standard In	Stdin	0	Input from the keyboard
Standard Out	Stdout	1	Output to the console
Standard Error	Stderr	2	Error output to the console

Read from stdin => read from fd 0: Whenever we write any character from keyboard, it read from stdin through fd 0 and save to file named /dev/tty.
Write to stdout => write to fd 1: Whenever we see any output on the Console screen, it's from the file named /dev/tty and written to stdout onto the screen through fd 1.
Write to stderr => write to fd 2: Whenever we see any error on the console screen, it is written from the file stderr onto the screen through fd 2.

List of System calls:

fd=open("sample.txt", O_CREAT | O_WRONLY:);	This system call would create a file named "Sample.txt" and will return a unique file descriptor number into the variable "fd". The file descriptor number is always a **unique integer.** A file can be opened in the following modes • **O_RDONLY**: read only • **O_WRONLY**: write only • **O_RDWR**: read and write • **O_CREAT**: create file if it doesn't exist • **O_EXCL**: prevent creation if it already exists
ssize_t read (int fd, void *buf, size_t count)	This system call is to read from the specified file with arguments of file descriptor fd, proper buffer with allocated memory (either static or dynamic) and the size of buffer. This call would return the number of bytes read (or zero in case of encountering the end of the file) on success and -1 in case of failure. The return bytes can be smaller than the number of bytes requested, just in case no data is available or file is closed.

ssize_t write (int fd, void *buf, size_t count)	This system call is to write to the specified file with arguments of the file descriptor fd, a proper buffer with allocated memory (either static or dynamic) and the size of buffer. This call would return the number of bytes written (or zero in case nothing is written) on success and -1 in case of failure. Proper error number is set in case of failure.
Close(fd)	This system call will close the process associated with the File descriptor number "fd". After the process is closed the "fd" which was assigned is removed from the file descriptor table and is free to be used by any other process

Example Programs:

```
/* ------------------------------------------------------------------------------------
File Name: open.c
Description: This example program demonstrates how to open and close a
             file using the open and close system call and also how a file
             descriptor number is assigned to a process.
   Author: Embedded Linux Team, EC Department, CSPIT, CHARUSAT
   Date & Time: 5th April, 2019 & 5:30 pm.
------------------------------------------------------------------------------------*/

#include<stdio.h>
#include<fcntl.h>
#include<sys/types.h>
#include<sys/stat.h>

void main()
{
     int fd,fd1,num;
```

```
	char str[30];
	close(0); //will close the process associated with file
descriptor number 0
	fd=open("data",O_RDONLY|O_CREAT);
	fd1=open("data",O_RDONLY|O_CREAT);
	perror("open");
	printf("fd=%d\n",fd);
	printf("fd1=%d\n",fd1);
}
```

Execution Steps:

```
jarvis@Jarvis:~/Desktop/FileDescriptor$ gcc -o open open.c
open.c: In function 'main':
open.c:10:2: warning: implicit declaration of function 'close' [-Wimplicit-funct
ion-declaration]
  close(0);
  ^
jarvis@Jarvis:~/Desktop/FileDescriptor$ ls
open  open.c  read.c  write.c
```

Result:

```
jarvis@Jarvis:~/Desktop/FileDescriptor$ ./open
open: Success
fd=0
fd1=3
```

```
/* ------------------------------------------------------------------------------------
File Name: STDIN_BLOCKED.c
Description: This example program demonstrates the blocking of the scanf
                statement on closing the file descriptor no "0".
   Author: Embedded Linux Team, EC Department, CSPIT, CHARUSAT
   Date & Time: 5th April, 2019 & 5:30 pm.
------------------------------------------------------------------------------------*/

#include<stdio.h>
#include<fcntl.h>
#include<sys/types.h>
#include<sys/stat.h>

void main()
{
	int fd,fd1,num;
	char str[30];
	close(0);
	fd=open("data.txt",O_RDONLY|O_CREAT);
	fd1=open("data1.txt",O_RDONLY|O_CREAT);
	printf("fd=%d\n",fd);
	printf("fd1=%d\n",fd1);
```

```
    scanf("enter data:%s\n",&str); //this statement will not
execute
}
```

Execution Steps:

```
jarvis@Jarvis:~/Desktop/FileDescriptor$ gcc -o STDIN STDIN_BLOCKED.c
STDIN_BLOCKED.c: In function 'main':
STDIN_BLOCKED.c:10:2: warning: implicit declaration of function 'close' [-Wimpli
cit-function-declaration]
  close(0);
  ^
STDIN_BLOCKED.c:15:8: warning: format '%s' expects argument of type 'char *', bu
t argument 2 has type 'char (*)[30]' [-Wformat=]
  scanf("enter data:%s\n",&str);
        ^
jarvis@Jarvis:~/Desktop/FileDescriptor$ ls
data       data.txt  open.c  read.c  STDIN_BLOCKED.c  write.c
data1.txt  open      read    STDIN   write
```

Result:

```
jarvis@Jarvis:~/Desktop/FileDescriptor$ ./STDIN
fd=0
fd1=3
jarvis@Jarvis:~/Desktop/FileDescriptor$
```

```
/* ------------------------------------------------------------------------------------------
File Name: STDOUT_BLOCKED.c
Description: This example program demonstrates blocking of the printf
                statement when file descriptor "1" is closed
   Author: Embedded Linux Team, EC Department, CSPIT, CHARUSAT
   Date & Time: 5th April, 2019 & 5:30 pm.
------------------------------------------------------------------------------------------*/

#include<stdio.h>
#include<fcntl.h>
#include<sys/types.h>
#include<sys/stat.h>

void main()
{
      int fd,fd1;
      char str[30];
      close(1);
      fd=open("data.txt",O_RDONLY|O_CREAT);
      fd1=open("data1.txt",O_RDONLY|O_CREAT);
      printf("fd=%d\n",fd); //this statement will not execute
      printf("fd1=%d\n",fd1);//this statement will not execute
```

```
}
```

Execution Steps:

```
jarvis@Jarvis:~/Desktop/FileDescriptor$ gcc -o STDOUT STDOUT_BLOCKED.c
STDOUT_BLOCKED.c: In function 'main':
STDOUT_BLOCKED.c:10:2: warning: implicit declaration of function 'close' [-Wimpl
icit-function-declaration]
  close(1);
  ^
jarvis@Jarvis:~/Desktop/FileDescriptor$ ls
data       data.txt  open.c  read.c  STDIN_BLOCKED.c  STDOUT_BLOCKED.c  write.c
data1.txt  open      read    STDIN   STDOUT           write
```

Result:

```
jarvis@Jarvis:~/Desktop/FileDescriptor$ ./STDOUT
jarvis@Jarvis:~/Desktop/FileDescriptor$
```

```
/* ----------------------------------------------------------------------------------------
File Name: STDERR_BLOCKED.c
Description: This example program demonstrates the blocking of the error
                statements on closing the the file descriptor number "2" .
   Author: Embedded Linux Team, EC Department, CSPIT, CHARUSAT
   Date & Time: 5th April, 2019 & 5:30 pm.
----------------------------------------------------------------------------------------*/

#include<stdio.h>
#include<fcntl.h>
#include<sys/types.h>
#include<sys/stat.h>

void main()
{
      int fd,fd1,num;
      char str[30];
      close(2);
      fd=open("data.txt",O_RDONLY|O_CREAT);
      fd1=open("data1.txt",O_RDONLY|O_CREAT);
      printf("fd=%d\n",fd);
```

```
	printf("fd1=%d\n",fd1);
	perror("open");//this statement will not execute

}
```

Execution Steps:

```
jarvis@Jarvis:~/Desktop/FileDescriptor$ gcc -o STDERR STDERR_BLOCKED.c
STDERR_BLOCKED.c: In function 'main':
STDERR_BLOCKED.c:10:2: warning: implicit declaration of function 'close' [-Wimpl
icit-function-declaration]
  close(2);
  ^
jarvis@Jarvis:~/Desktop/FileDescriptor$ ls
data       open    read.c            STDIN            STDOUT_BLOCKED.c
data1.txt  open.c  STDERR            STDIN_BLOCKED.c  write
data.txt   read    STDERR_BLOCKED.c  STDOUT           write.c
```

Result:

```
jarvis@Jarvis:~/Desktop/FileDescriptor$ ./STDERR
fd=2
fd1=3
jarvis@Jarvis:~/Desktop/FileDescriptor$
```

/* ----------

File Name: write.c

Description: This example program demonstrates how to write into a file using the open function.

Author: Embedded Linux Team, EC Department, CSPIT, CHARUSAT

Date & Time: 5th April, 2019 & 6:10 pm.

----------*/

```
#include<stdio.h>
#include<fcntl.h>
#include<sys/types.h>
#include<sys/stat.h>

main()
{
	int sz;
	int fd = open("data.txt", O_WRONLY | O_CREAT | O_TRUNC,
0644);
	sz = write(fd,"hello world\n",strlen("hello world\n"));
	printf("write operation successful \n");
	close(fd);
```

}

Execution Steps:

```
jarvis@Jarvis:~/Desktop/FileDescriptor$ gcc -o write write.c
write.c: In function 'main':
write.c:13:8: warning: implicit declaration of function 'write' [-Wimplicit-func
tion-declaration]
   sz = write(fd,"hello world\n",strlen("hello world\n"));
        ^
write.c:17:3: warning: implicit declaration of function 'close' [-Wimplicit-func
tion-declaration]
   close(fd);
   ^
jarvis@Jarvis:~/Desktop/FileDescriptor$ ls
open  open.c  read.c  write  write.c
jarvis@Jarvis:~/Desktop/FileDescriptor$
```

Result:

```
jarvis@Jarvis:~$ cd Desktop/FileDescriptor/
jarvis@Jarvis:~/Desktop/FileDescriptor$ ./write
write operation successful
jarvis@Jarvis:~/Desktop/FileDescriptor$ gedit data.txt
```

File Edit View Search Tools Documents Help

Open Save

hello world

/* ----------

File Name: read.c

Description: This example program demonstrates how to read from a file using the read function.

Author: Embedded Linux Team, EC Department, CSPIT, CHARUSAT

Date & Time: 5th April, 2019 & 10:10 am.

----------*/

```
#include<stdio.h>
#include<fcntl.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<string.h>

int main()
{
    char c[20]="";
    int fd = open("data.txt", O_RDONLY, 0);
    read(fd,&c,20);
    printf("data read from file:\n%s",c);
```

```
}
```

Execution Steps:

```
jarvis@Jarvis:~$ cd Desktop/FileDescriptor/
jarvis@Jarvis:~/Desktop/FileDescriptor$ gcc -o read read.c
read.c: In function 'main':
read.c:11:5: warning: implicit declaration of function 'read' [-Wimplicit-functi
on-declaration]
     read(fd,&c,20);
     ^
jarvis@Jarvis:~/Desktop/FileDescriptor$ ls
open  open.c  read  read.c  write  write.c
jarvis@Jarvis:~/Desktop/FileDescriptor$
```

Result:

```
jarvis@Jarvis:~/Desktop/FileDescriptor$ ./read
data read from file:
hello world
jarvis@Jarvis:~/Desktop/FileDescriptor$
```

Experiment No-7

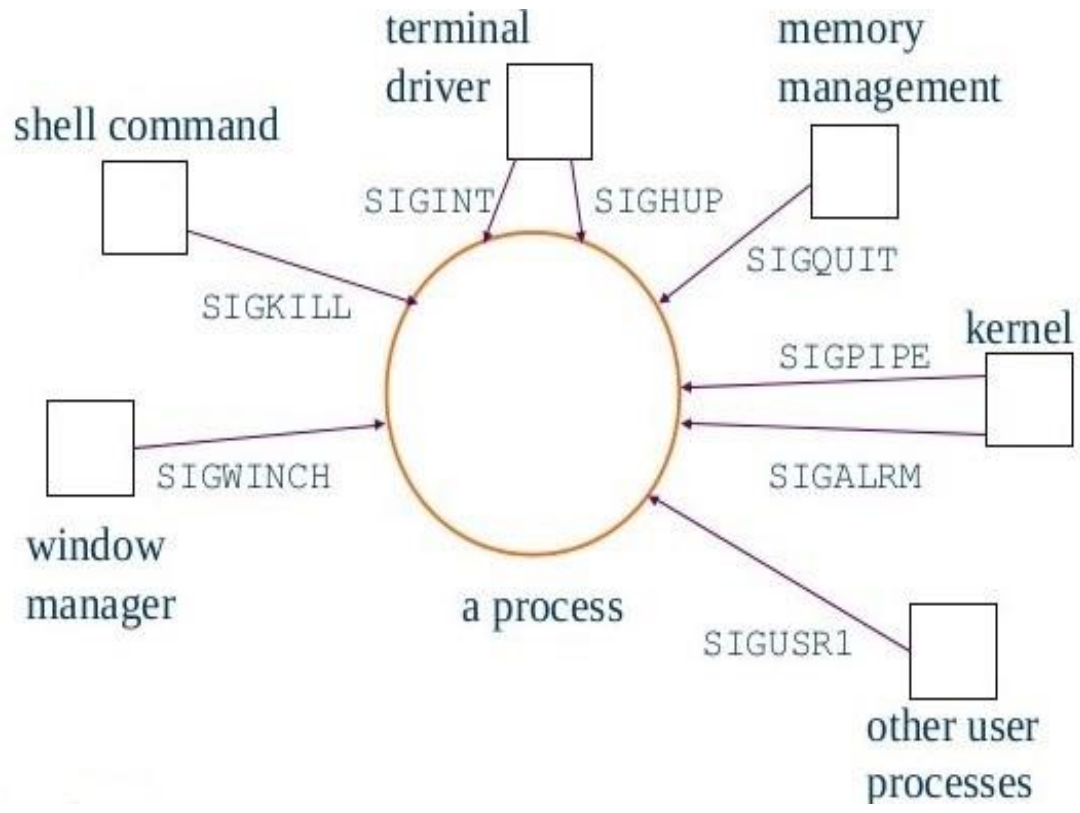
Aim: Implementation of Signal Handling in Linux.

Theory:

A signal is a software generated interrupt that is sent to a process by the OS because of when user press ctrl-c or another process tell something to this process. Each signal identified by a number, from 1 to 31. Signals don't carry any argument and their names are mostly self-explanatory. For instance SIGKILL or signal number 9 tells the program that someone tries to kill it.

Signal as Interrupt:

In addition to informative nature of signals, they also interrupt your program. I.e. to handle a signal, one of the threads in your program, stops its execution and temporarily switches to signal handler. Note that as in version 2.6 of Linux kernel, most of the signals interrupt only one thread and not the entire application as it used to be once. Moreover, signal handler itself can be interrupted by some other signal.



List of Signals:

There is an easy way to list down all the signals supported by your system. Just issue the **kill -l** command and it would display all the supported signals

Description of some Signals:

SIGINT	Issued if the user sends an interrupt signal (Ctrl + C)
SIGQUIT	Issued if the user sends a quit signal (Ctrl + D)
SIGKILL	If a process gets this signal it must quit immediately and will not perform any clean-up operations
SIGALRM	Alarm clock signal (used for timers)
SIGTERM	Software termination signal (sent by kill by default)

Default Actions:

Every signal has a default action associated with it. The default action for a signal is the action that a script or program performs when it receives a signal. Some of the possible default actions are −

- Terminate the process.
- Ignore the signal.
- Stop the process.
- Continue a stopped process.

Advantages:

- Unless Semaphore it takes the shortest possible CPU time. The signals are the flag or one or two byte message used as the IPC functions for synchronizing the concurrent processing of the tasks.
- A signal is identical to setting a flag that is shared and used by another interrupt servicing process.

Disadvantages:

- Signal is handled by only very high priority process. That may disrupt the usual schedule and usual priority inheritance mechanism.

Example Program:

```
/* ---------------------------------------------------------------------------------------
File Name: sig_handler.c
Description: This example program will print hello...pid. After pressing
             Ctrl+C five times the program will get terminated. If user
             presses ctrl-c to terminate the process because
             of SIGINT signal sent and its default handler to terminate
             the process.
Author: Embedded Linux Team, EC Department, CSPIT, CHARUSAT
Date & Time: 5th April, 2019 & 9:10 am
---------------------------------------------------------------------------------------*/
#include<stdio.h>

#include<signal.h>

void func(int n)

{

     static int count;

     count++;
```

```
        if(count<5)
                printf("process still running..\n");
        else
                signal(2,SIG_DFL);
}
void main()
{
        printf("hello...pid=%d\n",getpid());
        signal(2,func);
        while(1);
}
```

Execution Steps:

```
neel@neel-HP-280-G1-MT:~/Desktop/2_Signal$ gcc -o s2 sig_handler.c
sig_handler.c: In function ‘main’:
sig_handler.c:14:28: warning: implicit declaration of function ‘getpid’ [-Wimpli
cit-function-declaration]
  printf("hello...pid=%d\n",getpid());
                            ^
neel@neel-HP-280-G1-MT:~/Desktop/2_Signal$ ./s2
```

Result:

```
neel@neel-HP-280-G1-MT:~/Desktop/2_Signal$ ./s2
hello...pid=2079
^Cprocess still running..
^Cprocess still running..
^Cprocess still running..
^Cprocess still running..
^C^C
neel@neel-HP-280-G1-MT:~/Desktop/2_Signal$
```

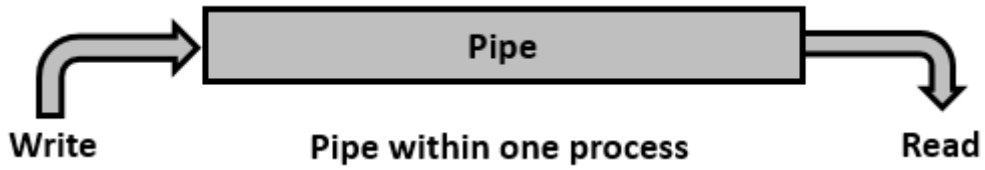
Experiment No. 8

Aim: Implementation of two way communication using Linux Pipe.

Theory:

Pipe is a communication medium between two or more related or interrelated processes. It can be either within one process or a communication between the child and the parent processes. Communication can also be multi-level such as communication between the parent, the child and the grand-child, etc. Communication is achieved by one process writing into the pipe and other reading from the pipe. To achieve the pipe system call, create two files, one to write into the file and another to read from the file.

Pipe mechanism can be viewed with a real-time scenario such as filling water with the pipe into some container, say a bucket, and someone retrieving it, say with a mug. The filling process is nothing but writing into the pipe and the reading process is nothing but retrieving from the pipe. This implies that one output (water) is input for the other (bucket).



Advantages:

- It is used for communication between related processes.

Disadvantages:

- Pipe is unidirectional.
- Failed for communication between unrelated processes.

List of System calls:

int pipe (int pipedes[2])	This system call would create a pipe for one-way communication i.e., it creates two descriptors, first one is connected to read from the pipe and other one is connected to write into

	the pipe. Descriptor pipedes[0] is for reading and pipedes[1] is for writing. Whatever is written into pipedes[1] can be read from pipedes[0]. This call would return zero on success and -1 in case of failure.
ssize_t read (int fd, void *buf, size_t count)	This system call is to read from the specified file with arguments of file descriptor fd, proper buffer with allocated memory (either static or dynamic) and the size of buffer. This call would return the number of bytes read (or zero in case of encountering the end of the file) on success and -1 in case of failure. The return bytes can be smaller than the number of bytes requested, just in case no data is available or file is closed.
ssize_t write (int fd, void *buf, size_t count)	The above system call is to write to the specified file with arguments of the file descriptor fd, a proper buffer with allocated memory (either static or dynamic) and the size of buffer. This call would return the number of bytes written (or zero in case nothing is written) on success and -1 in case of failure. Proper error number is set in case of failure.

Example Program:

```
/* ----------------------------------------------------------------------------------------
File Name: pipe_1.c
Description: This example program demonstrate two way communication
             between two process as parent process & child process by
             creating individual pipe p and q respectively. Here child
             process print the message which is sent by the parent
             process and parent process print the message which is sent
             by the child process.
        Author: Embedded Linux Team, EC Department, CSPIT, CHARUSAT
  Date & Time: 20th March, 2019 & 10:10 am.
----------------------------------------------------------------------------------------*/

#include<stdio.h>
#include<fcntl.h>
#include<unistd.h>
#include<string.h>
#include<sys/stat.h>
#include<sys/types.h>
void main()
{
        int p[2],q[2];

        /* Defining Pipe p*/
        pipe(p);

        /* Defining Pipe q*/
        pipe(q);

    /* fork()system call is used to create parent and child
process respectively */

        if(fork())
        {
         /* This is parent process

                char s1[10],s2[10];
                printf("String entered from parent\n");
                scanf("%s",s1);

                /* writing to pipe p*/
                write(p[1],s1,strlen(s1)+1);

                /* reading from pipe q*/
                read(q[0],s2,sizeof(s2));
```

```
		printf("string received from child=%s\n",s2);
	}
	else
	{
		/* This is child process*/
		char str1[10],str2[10];

		/* reading from pipe p*/
		read(p[0],str2,sizeof(str2));

		printf("String entered from child\n");
		scanf("%s",str1);

		/*writing to pipe q*/
		write(q[1],str1,strlen(str1)+1);

		printf("string received from parent=%s\n",str2);
	}
}
```

Execution Steps:

```
miral@ubuntu:~/Desktop/Linux_RTOS_Program/4_Pipe$ gcc -o pipe_1 pipe_1.c
miral@ubuntu:~/Desktop/Linux_RTOS_Program/4_Pipe$ ls
pipe_1  pipe_1.c  pipe_1.c~
miral@ubuntu:~/Desktop/Linux_RTOS_Program/4_Pipe$
```

Result:

```
miral@ubuntu:~/Desktop/Linux_RTOS_Program/4_Pipe$ ./pipe_1
String entered from parent
hello
String entered from child
hi
string received from parent=hello
string received from child=hi
miral@ubuntu:~/Desktop/Linux_RTOS_Program/4_Pipe$
```

Do it yourself (DIY):

1) Write a program to read and write 2 messages using single pipe.
   Hint:
   1) Create Pipe
   2) Send 1st message to the pipe
   3) Retrieve message from the pipe and write it to standard output
   4) Repeat step 2 and 3 for 2nd message

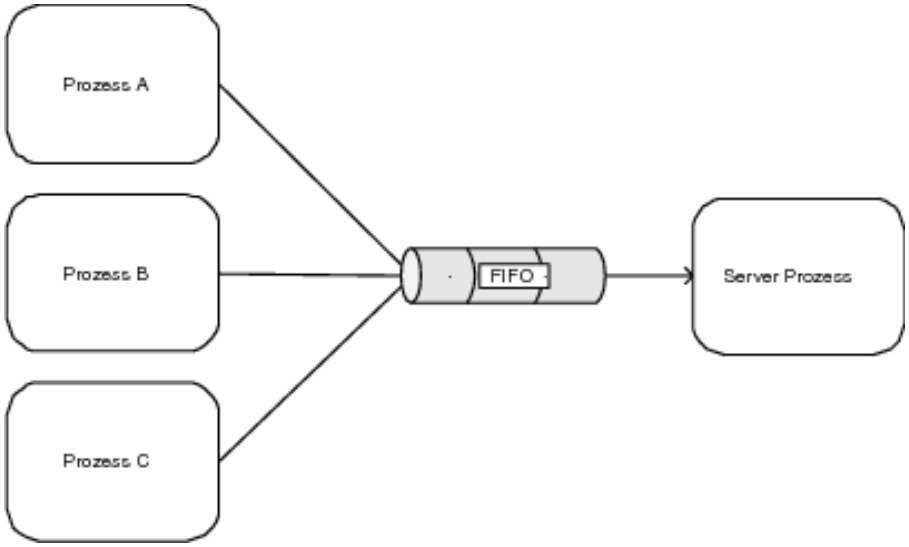
Experiment No.9

Aim: Implementation of Inter-process communication: FIFO

Theory:

A FIFO (First In- First Out) special file (a named pipe) is similar to a pipe, except that it is accessed as part of the file system. It can be opened by multiple processes for reading or writing. When processes are exchanging data via the FIFO, the kernel passes all data internally without writing it to the file system. Thus, the FIFO special file has no contents on the file system; the file system entry merely serves as a reference point so that processes can access the pipe using a name in the file system.

The kernel maintains exactly one pipe object for each FIFO special file that is opened by at least one process. The FIFO must be opened on both ends (reading and writing) before data can be passed. Normally, opening the FIFO blocks until the other end is opened also.



Advantages:

- A major advantage of using FIFO is that they provide a useful way to send one-line requests to an OpenEdge background session running a message handler procedure.
- Multiple users can send requests through the same named pipe and each request is removed from the pipe as it is received.

- In addition, the message handler procedure can loop indefinitely looking for input because it blocks (waits) until there is something to read.
- Finally, output through FIFO is more efficient than writing a complete response to an ordinary file, closing the file, and then informing the recipient that the results are available. The receiving process can read the result through a named pipe as soon as it is written

Disadvantages:

- A disadvantage of named pipes is that multiple processes cannot use a single named pipe to send or receive multi-line messages, unless you define a more complex protocol to control message interleaving.
- Also, although synchronizing named pipe input and output is helpful in some situations, it is a problem in others. For example, if the message handler procedure running in the background OpenEdge session starts returning results to an output named pipe, and for some reason the requestor is not ready to read the results, the message handler cannot move on to read the next request.

List of System calls:

mkfifo(myfifo, 0666)	Create the FIFO and gives the permission. mkfifo(<pathname>, <permission>)
fd = open(myfifo, O_WRONLY);	Write Hi to the FIFO. Open FIFO for write only
unlink(myfifo);	Remove FIFO
fd = open(myfifo, O_RDONLY);	open, read, and display the message from the FIFO
char buf[MAX_BUF];	Creates a buffer to store the Data send

	by the FIFO Sender.

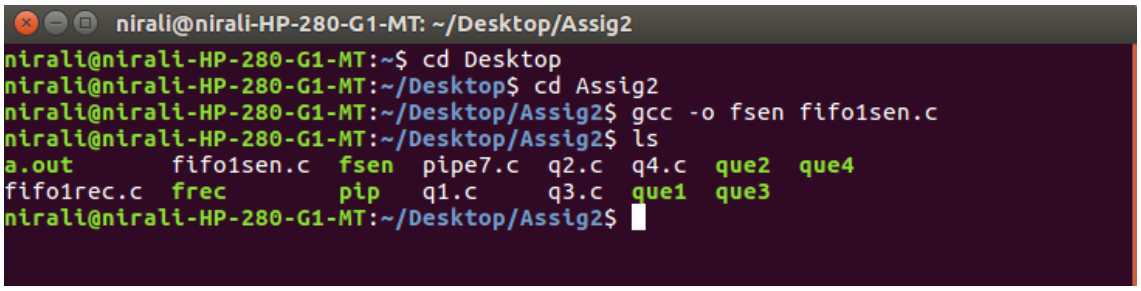
Program:

```
/* ------------------------------------------------------------------------------------
File Name: fifo1sen.c
Description: This example program demonstrate two way communication
             between Sender and Receiver . Here Sender process send the
             message “Hi” to receiver and receiver process gets the
             message which is sent by the sender process in different
             terminal.
Author: Embedded Linux Team, EC Department, CSPIT, CHARUSAT
------------------------------------------------------------------------------------*/
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
int fd;
char * myfifo = "/tmp/myfifo";
/* create the FIFO (named pipe) */
mkfifo(myfifo, 0666);
/* write "Hi" to the FIFO */
fd = open(myfifo, O_WRONLY);
write(fd, "Hi", sizeof("Hi"));
close(fd);
/* remove the FIFO */
unlink(myfifo);
return 0;
}
```

```
/* ---------------------------------------------------------------------------------------
File Name: fifo1rec.c
Description:    This example program demonstrate two way communication
                between Sender and Receiver . Here Receiver process receive
                the message “Hi” from the senderwhich is sent by the sender
                process from different terminal. The Receiver terminal will be
                get blocked if it does not get any input from sender terminal.
Author: Embedded Linux Team, EC Department, CSPIT, CHARUSAT
---------------------------------------------------------------------------------------*/
#include <fcntl.h>

#include <stdio.h>

#include <sys/stat.h>

#include <unistd.h>

#define MAX_BUF 1024

int main()

{

int fd;

char * myfifo = "/tmp/myfifo";

char buf[MAX_BUF];

/* open, read, and display the message from the FIFO */

fd = open(myfifo, O_RDONLY);

read(fd, buf, MAX_BUF);

printf("Received: %s\n", buf);

close(fd);

return 0;

}
```

Execution Steps:

Step1: From Sender Terminal



Step2: From Receiver Terminal

```
nirali@nirali-HP-280-G1-MT:~$ cd Desktop
nirali@nirali-HP-280-G1-MT:~/Desktop$ cd Assig2
nirali@nirali-HP-280-G1-MT:~/Desktop/Assig2$ gcc -o fsen fifo1sen.c
nirali@nirali-HP-280-G1-MT:~/Desktop/Assig2$ ls
a.out       fifo1sen.c  fsen  pipe7.c  q2.c  q4.c  que2  que4
fifo1rec.c  frec        pip   q1.c     q3.c  que1  que3
nirali@nirali-HP-280-G1-MT:~/Desktop/Assig2$ cc fifo1sen.c
nirali@nirali-HP-280-G1-MT:~/Desktop/Assig2$ ./fsen
```

Step3: From Sender terminal again.

```
nirali@nirali-HP-280-G1-MT: ~/Desktop/Assig2
nirali@nirali-HP-280-G1-MT:~$ cd Desktop
nirali@nirali-HP-280-G1-MT:~/Desktop$ cd Assig2
nirali@nirali-HP-280-G1-MT:~/Desktop/Assig2$ gcc -o frec fifo1rec.c
nirali@nirali-HP-280-G1-MT:~/Desktop/Assig2$ ls
a.out       fifo1sen.c  fsen  pipe7.c  q2.c  q4.c  que2  que4
fifo1rec.c  frec        pip   q1.c     q3.c  que1  que3
nirali@nirali-HP-280-G1-MT:~/Desktop/Assig2$ cc fifo1rec.c
nirali@nirali-HP-280-G1-MT:~/Desktop/Assig2$
```

Output:

From Receiver Terminal

```
fifo1rec.c  frec        pip   q1.c     q3.c  que1  que3
nirali@nirali-HP-280-G1-MT:~/Desktop/Assig2$ cc fifo1rec.c
nirali@nirali-HP-280-G1-MT:~/Desktop/Assig2$ ./frec
Received: Hi
nirali@nirali-HP-280-G1-MT:~/Desktop/Assig2$
```

Experiment No.10

Aim: Inter process communication:-share memory, message Queue, FCNTL

Theory:

SharedMemory:-

Shared memory is a concept where two or more process can access the common memory. And communication is done via this shared memory where changes made by one process can be viewed by anther process.

The problem with pipes, fifo and message queue – is that for two process to exchange information. The information has to go through the kernel.

- Server reads from the input file.
- The server writes this data in a message using either a pipe, fifo or message queue.
- The client reads the data from the IPC channel, again requiring the data to be copied from kernel's IPC buffer to the client's buffer.
- Finally the data is copied from the client's buffer.

A total of four copies of data are required (2 read and 2 write). So, shared memory provides a way by letting two or more processes share a memory segment. With Shared Memory the data is only copied twice – from input file into shared memory and from shared memory to the output file.

Advantages:

- The main advantage of shared memory is that copying of message data is eliminated .the usual mechanism for synchronizing shared memory access is semaphores.

List of System calls:

ftok():	is use to generate a unique key.
shmget():	int shmget(key_t,size_tsize,intshmflg); upon successful completion, shmget() returns an identifier for the shared memory segment.
shmdt():	When you're done with the shared memory segment, your program should detach itself from it using shmdt(). int shmdt(void *shmaddr);

shmctl():	when you detach from shared memory,it is not destroyed. So, to destroy shmctl() is used. shmctl(int shmid,IPC_RMID,NULL);
shmat():	Before you can use a shared memory Segment you have to attach yourself to is using shmat(). Void *shmat (int shmid, void*shmaddr,intshmfig); Shmid is shared memory id:shmaddr specifies specific address to use but we should set It is zero and os will automatically choose the address.

Example Program:

```
/* ------------------------------------------------------------------------------------
Description: In shared memory as an intance if we have two write
                exceutionsfiles .In this file we input our data
        Author: Embedded Linux Team, EC Department, CSPIT, CHARUSAT
   Date & Time: 20th March, 2019 & 10:10 am.
------------------------------------------------------------------------------------*/
```

.

File name:-Shared_mem_w_1

```
#include<stdio.h>
#include<fcntl.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<string.h>
#include<unistd.h>
#include<sys/ipc.h>
#include<sys/msg.h>
```

```
#include<sys/shm.h>
main()
{
    int count=20,i;
    char *p,alpha='A';
    int id;
    id=shmget(5,50,IPC_CREAT|0644);
    p=shmat(id,0,0);
    for(i=0;i<=count;alpha++,i++)
    {
        *p=alpha;
        p++;
        sleep(2);
    }
}
```

/* ----------
Description: In shared memory as an intance if we have two write exceutionsfiles .In this file we input our data.
Author: Embedded Linux Team, EC Department, CSPIT, CHARUSAT
Date & Time: 20th March, 2019 & 10:10 am.
----------*/

File name:-Shared_mem_w_2

```
 #include<stdio.h>
#include<fcntl.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<string.h>
#include<unistd.h>
#include<sys/ipc.h>
#include<sys/msg.h>
#include<sys/shm.h>
main()
{
    int count=20,i;
    char *p,alpha='a';
    int id;
    id=shmget(5,50,IPC_CREAT|0644);
    p=shmat(id,0,0);
    for(i=0;i<=count;alpha++,i++)
    {
        *p=alpha;
```

```
        p++;
        sleep(1);
    }
}
```

/* ----------
Description: In this file which write file execute first that file's data will be print.
Author: Embedded Linux Team, EC Department, CSPIT, CHARUSAT
Date & Time: 20th March, 2019 & 10:10 am.
----------*/

File Name: Shared_mem_r

```
#include<stdio.h>
#include<fcntl.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<string.h>
#include<unistd.h>
#include<sys/ipc.h>
#include<sys/msg.h>
#include<sys/shm.h>
main()
{
    char *read;
    int id,i;
    id=shmget(5,50,IPC_CREAT|0644);
    read=shmat(id,0,0);
    for(i=0;i<=40;i++,read++)
    {
        printf("%c\n",*read);
        sleep(1);
    }76
}
```

Execution Steps:

```
mayur@mayur:~/Downloads/Embedded_Linux_System_Programming/7_Shared_Memory/Multiple_Data_in_One_File$ gcc -o r1  shared_mem_r.
shared_mem_r.c:10:1: warning: return type defaults to ‘int’ [-Wimplicit-int]
 main()
 ^
```

Result:

```
mayur@mayur:~/Downloads/Embedded_Linux_System_Programming/7_Shared_Memory/Multiple_Data_in_One_File$ ./r1
abcdefghijklmnopqrstu
```

Message Queue:-

A message queue is a linked list of messages stored within the kernel and identified by a message queue identifier. A new queue is created or an existing queue opened by **msgget()**.

]New messages are added to the end of a queue by **msgsnd()**. Every message has a positive long integer type field, a non-negative length, and the actual data bytes (corresponding to the length), all of which are specified to msgsnd() when the message is added to a queue. Messages are fetched from a queue by **msgrcv()**. We don’t have to fetch the messages in a first-in, first-out order. Instead, we can fetch messages based on their type field.

All processes can exchange information through access to a common system message queue. The sending process places a message (via some (OS) message-passing module) onto a queue which can be read by another process. Each message is given an identification or type so that processes can select the appropriate message. Process must share a common key in order to gain access to the queue in the first place.

List of System calls:

ftok():	is use to generate a unique key.
msgget():	either returns the message queue identifier for a newly created message queue or returns the identifiers for a queue which exists with the same key value..
msgsnd():	Data is placed on to a message queue by calling msgsnd().
msgrcv():	messages are retrieved from a queue.
msgctl():	It performs various operations on a queue. Generally it is use to destroy message queue.

Example Program:

/* ----------
Description: This Program will send a message to specific ID which is created by kernel itself as per the Program. And Write a message.

Author: Embedded Linux Team, EC Department, CSPIT, CHARUSAT
Date & Time: 20th March, 2019 & 10:10 am.
----------*/

File Name-mesgq_s

```
#include<stdio.h>
#include<fcntl.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<string.h>
#include<unistd.h>
#include<sys/ipc.h>
#include<sys/msg.h>
struct msgbuf
    {
        long m_type;
        char msg_txt[10];
```

```
    };
void main()
{
    struct msgbuf v;
    int id;
    id=msgget(9,IPC_CREAT|0644);
    printf("%d\n",id);
    printf("enter your data to send..\n");
    scanf("%s",v.msg_txt);
    v.m_type=2;
    msgsnd(id,v.msg_txt,strlen(v.msg_txt)+1,0);
    perror("msgsnd");
}
```

/* ----------
Description:-In message queue receiver the program will receive the message sent by the sender from the communication ID is queue.
Author: Embedded Linux Team, EC Department, CSPIT, CHARUSAT
Date & Time: 20th March, 2019 & 10:10 am.
----------*/

File name:-queue_receiver

```
#include<stdio.h>
#include<fcntl.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<string.h>
#include<unistd.h>
#include<sys/ipc.h>
#include<sys/msg.h>
struct msgbuf
    {
        long m_type;
        char msg_txt[20];
    };
void main()
{
    struct msgbuf v;
    int id;
    id=msgget(9,0644);
    printf("%d\n",id);
    v.m_type=0;
    msgrcv(id,v.msg_txt,sizeof(v.msg_txt),v.m_type,0);
    perror("msgrcv");
    printf("%s\n",v.msg_txt);
}
```

Execution Steps:

```
mayur@mayur:~/Downloads/Embedded_Linux_System_Programming/6_Message_Queue$ gcc -o msg_rcv queue_receiver.c
mayur@mayur:~/Downloads/Embedded_Linux_System_Programming/6_Message_Queue$ ls
msgq_s.c  msg_rcv  msg_send  queue_receiver.c
```

Result:

```
mayur@mayur:~/Downloads/Embedded_Linux_System_Programming/6_Message_Queue$ sudo ./msg_rcv
0
msgrcv: Success
Raj
mayur@mayur:~/Downloads/Embedded_Linux_System_Programming/6_Message_Queue$
```

FCNTL:

The fcntl system call is the access point for several advanced operations on file descriptors. The first argument to fcntl is an open file descriptor, and the second is a value that indicates which operation is to be performed. For some operations, fcntl takes an additional argument.The fcntl system call allows a program to place a read lock or a write lock on a file.

Example Program:

```
/* ------------------------------------------------------------------------------------------
Description: The Program opens a file for writing whose name is provided on
             the command line and then places write lock on it .The
             Program waits for user to hit enter and then unlocks and
             closes the file.
      Author: Embedded Linux Team, EC Department, CSPIT, CHARUSAT
   Date & Time: 20th March, 2019 & 10:10 am.
------------------------------------------------------------------------------------------*/
```

File Name:-fcntl.c

```
#include<stdio.h>
#include<fcntl.h>
#include<sys/types.h>
#include<sys/stat.h>
```

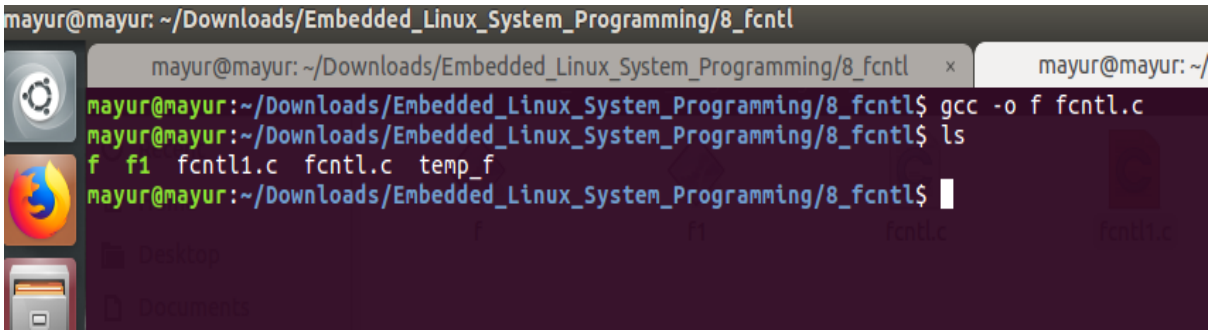
```
#include<string.h>
#include<unistd.h>
#include<sys/ipc.h>
#include<sys/msg.h>
void main()
{
    int fd,i;
    struct flock v;
    char str[]="123456789009876543210";
    fd=open("temp_f",O_CREAT|O_RDWR|O_APPEND,0644);
    v.l_type = F_WRLCK;
    v.l_whence=SEEK_SET;
    v.l_start=0;
    v.l_len=0;
    fcntl(fd,F_SETLKW,&v);
    if(fd<0)
    {
        perror("open");
        return;
    }
    for(i=0;str[i];i++)
    {
        write(fd,&str[i],1);
        sleep(1);
    }
    v.l_type=F_UNLCK;
    fcntl(fd,F_SETLKW,&v);
}
```

File name:-fcntl1.c

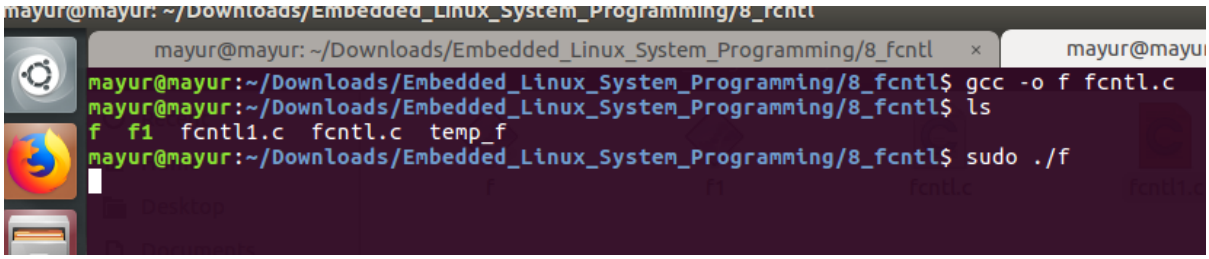
```
#include<stdio.h>
#include<fcntl.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<string.h>
#include<unistd.h>
#include<sys/ipc.h>
#include<sys/msg.h>
void main()
{
    int fd,i;
    struct flock v;
    char str[]="abcdefghijklmnopqrstuvwxyz";
    fd=open("temp_f",O_RDWR|O_APPEND,0644);
    v.l_type = F_WRLCK;
    v.l_whence=SEEK_SET;
    v.l_start=0;
    v.l_len=0;
```

```
    fcntl(fd,F_SETLKW,&v);
    if(fd<0)
    {
        perror("open");
        return;
    }
    for(i=0;str[i];i++)
    {
        write(fd,&str[i],1);
        sleep(1);
    }
    v.l_type=F_UNLCK;
    fcntl(fd,F_SETLKW,&v);
}
```

Execution Steps:



Result:



Experiment No. 11

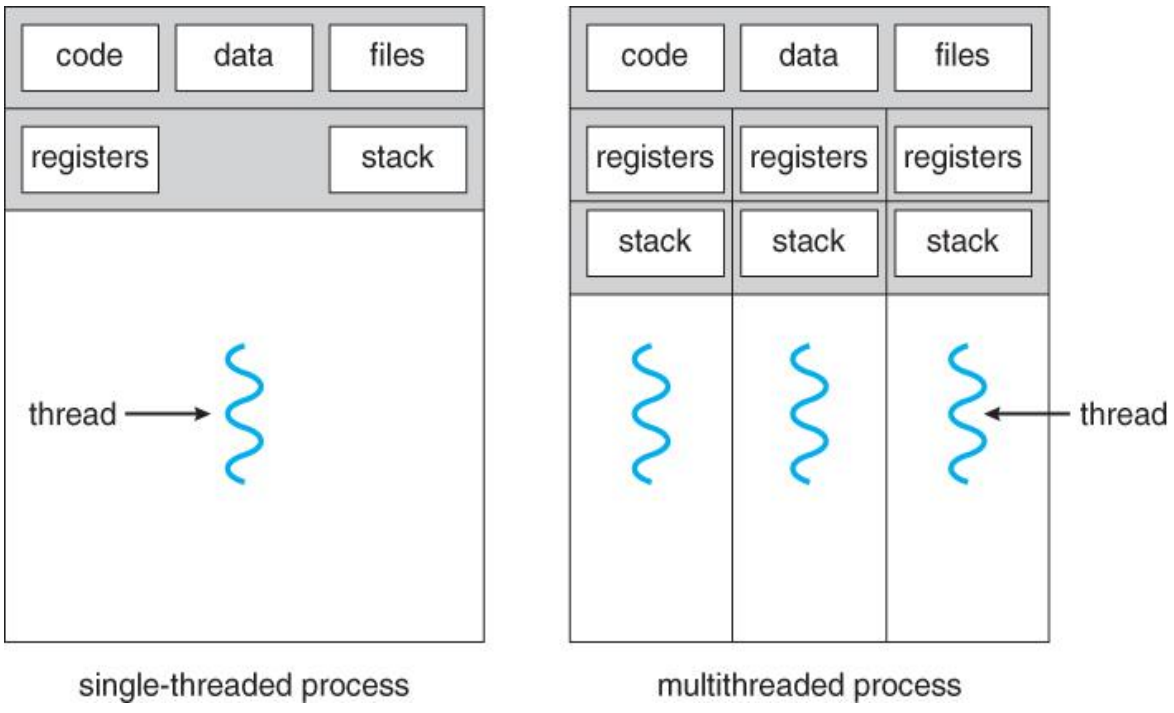
Aim: Inter-process Communication: Multithreading

Theory:

Thread: A thread is a path of execution within a process. A process can contain multiple threads.

Multithreading: A thread is also known as lightweight process. The idea is to achieve parallelism by dividing a process into multiple threads. For example, in a browser, multiple tabs can be different threads. MS Word uses multiple threads: one thread to format the text, another thread to process inputs, etc.

The primary difference between is that threads within the same process run in a shared memory space, while processes run in separate memory spaces. Threads are not independent of one another like processes are, and as a result threads share with other threads their code section, data section, and OS resources (like open files and signals). But, like process, a thread has its own program counter (PC), register set, and stack space.



single-threaded process multithreaded process

Advantages of Thread over Process:

1. Responsiveness: If the process is divided into multiple threads, if one thread completes its execution, then its output can be immediately returned.

2. Faster context switch: Context switch time between threads is lower compared to process context switch. Process context switching requires more overhead from the CPU.

3. Resource sharing: Resources like code, data, and files can be shared among all threads within a process.
Note: stack and registers can't be shared among the threads. Each thread has its own stack and registers.

4. Communication: Communication between multiple threads is easier, as the threads shares common address space. while in process we have to follow some specific communication technique for communication between two process.

6. Enhanced throughput of the system: If a process is divided into multiple threads, and each thread function is considered as one job, then the number of jobs completed per unit of time is increased, thus increasing the throughput of the system

List of System calls:

pthread_create () **Syntax:** int pthread_create(pthread_t * thread, constpthread_attr_t * attr, void * (*start_routine) (void *), void *arg);	The pthread_create() function isused to create a new thread, with attributes specified by attr, within aprocess. **Parameters:** **thread:** pointer to an unsigned integer value that returns the

	thread id of the thread created. **attr:** pointer to a structure that is used to define thread attributes like detached state, scheduling policy, stack address, etc. Set to NULL for default thread attributes. **start_routine:** pointer to a subroutine that is executed by the thread. The return type and parameter type of the subroutine must be of type void *. The function has a single attribute but if multiple values need to be passed to the function, a struct must be used. **arg:** pointer to void that contains the arguments to the function

	defined in the earlier argument
pthread_equal() **syntax:** intpthread_equal (pthread_t t1, pthread_t t2);	This compares two thread which is equal or not. This function compares two thread identifiers. It return '0' and non zero value. If it is equal then return non zero value else return 0.
pthread_self()	The pthread_self() function returns the IDof the thread inwhich it is invoked.
pthread_join() ***Syntax:*** *int pthread_join(pthread_tth, void **thread_return)*	Used to wait for the terminationof a thread. **Parameters:** **th:**thread id of the thread forwhich the current thread waits. **thread_return:** pointer to the thread waiting to join this thread may read the return status.

pthread_exit() ***syntax:*** *void pthread_exit(void *retval);*	Used to terminate the thread. Parameter: This method accepts a mandatory parameter **retval** which is the pointer to an integer that stores the return status of the thread terminated. The scope of this variable must be global so that any location where the exit status of the thread mentioned in th is stored.

Example Program 1:

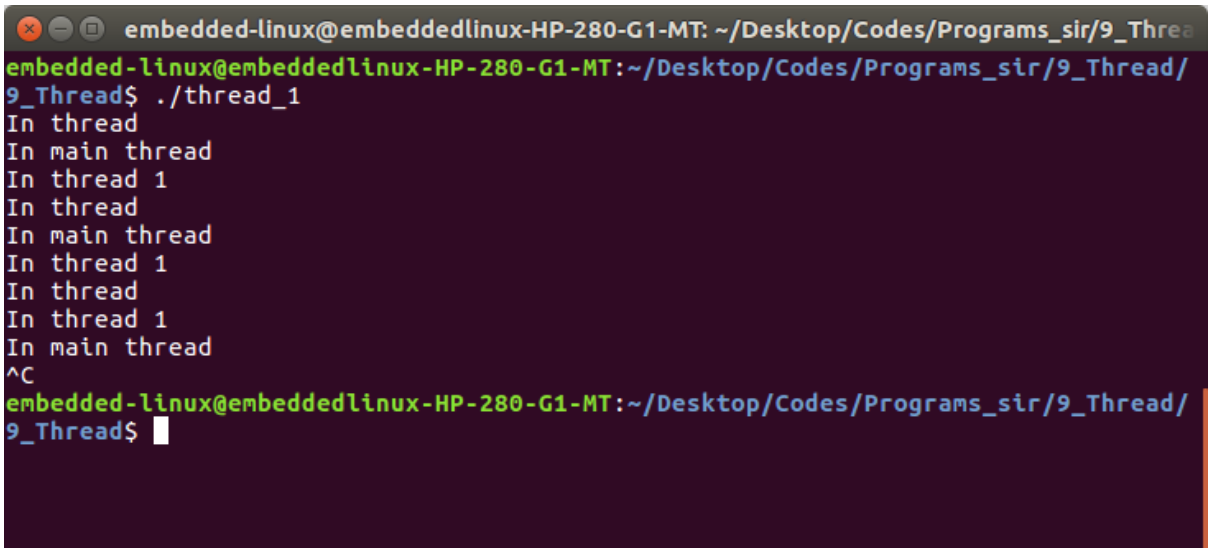
```
/* ------------------------------------------------------------------------------------------
File Name: thread_1.c
Description: This example program demonstrate the simultaneous execution
                of threads linked with fun() and thread_1() which are created
                by pthread_create() function into the main() function.
        Author: Embedded Linux Team, EC Department, CSPIT, CHARUSAT
   Date & Time: 05thApril, 2019 & 10:36 am.
------------------------------------------------------------------------------------------*/


#include<stdio.h>
#include<string.h>
#include<pthread.h> // library for thread functions
#include<stdlib.h>
#include<unistd.h>

void *fun(void *arg)
{
while(1)
        {
                printf("In thread\n");
                sleep(1);
        }
```

```
}
void *thread_1(void *arg)
{
        while(1)
        {
                printf("In thread 1\n");
                sleep(1);
        }
}
void main()
{
        pthread_ttid,tid1;
        pthread_create(&tid,NULL,fun,NULL); //first thread created
        pthread_create(&tid1,NULL,thread_1,NULL); //second thread created
        while(1)
        {
                printf("In main thread\n");
                sleep(1);
        }
}
```

EXECUTION STEPS: (Common for all 6 programs)

```
saumil@Patidar:~/Desktop/Embedded_Linux_System_Programming/9_Thread/9_Thread$ gc
c thread_1.c -o thread_1 -lpthread
saumil@Patidar:~/Desktop/Embedded_Linux_System_Programming/9_Thread/9_Thread$ ls
thread_1     thread_2     thread_3    thread_4.c  thread_6
thread_1.c   thread_2.c   thread_3.c  thread_5    thread_6.c
thread_1.c~  thread_2.c~  thread_4    thread_5.c  thread_6.c~
saumil@Patidar:~/Desktop/Embedded_Linux_System_Programming/9_Thread/9_Thread$
```

RESULT:

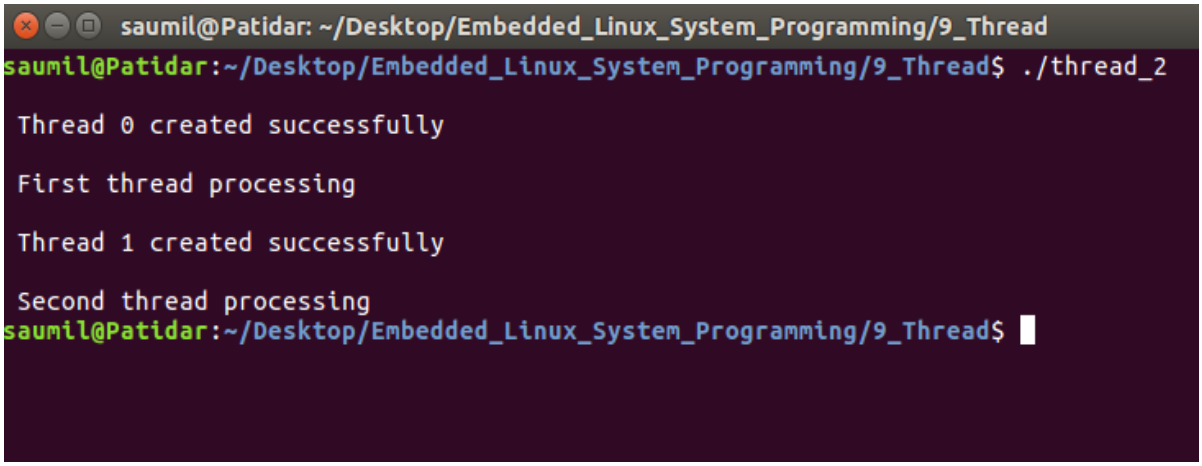


Example Program 2:

```
/* ---------------------------------------------------------------------------------------
File Name: thread_2.c
Description: This example program demonstratesthe execution of threads by
                order. In main function when thread is created, and fun() is
                invoked, the ids are compared through pthread_equal()
                function and respective branch is executed.
        Author: Embedded Linux Team, EC Department, CSPIT, CHARUSAT
   Date & Time: 05th April, 2019 & 10:36 am.
---------------------------------------------------------------------------------------*/


#include<stdio.h>
#include<fcntl.h>
#include<unistd.h>
#include<stdio.h>
#include<string.h>
#include<pthread.h>
#include<stdlib.h>
#include<unistd.h>

pthread_ttid[2];

void *fun(void *arg)
{
        unsigned long i=0;
        /*returns ID of thread in which it is invoked*/
```

```
        pthread_t id = pthread_self();
        /*Compare two threads, returns non-zero if equal else zero.*/
        if(pthread_equal(id,tid[0]))
        {
                printf("\n First thread processing\n");
        }
        else if (pthread_equal(id,tid[1]))
        {
                printf("\n Second thread processing\n");
        }
        for(i=0;i<(0xFFFFFFFF);i++);
        return NULL;
}
int main(void)
{
        inti=0;
        int err;
        while(i<2)
        {
                /*Returns non-zero value if thread is not created successfully*/
                err=pthread_create(&tid[i],NULL,&fun,NULL);
                if(err!=0)
                        printf("\ncan't create thread:[%s]",strerror(err));
                else
                        printf("\n Thread %d created successfully\n",i);
                i++;
        }
        sleep(5);
        return 0;
}
```

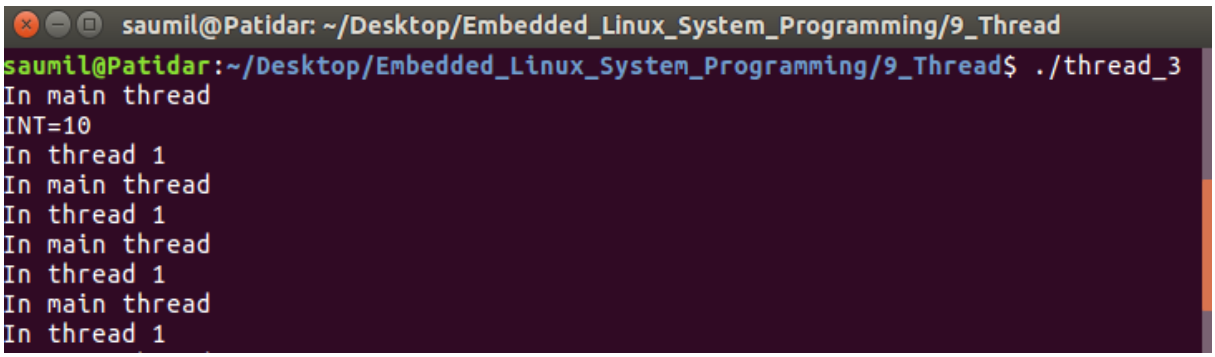
OUTPUT:



Example Program 3:

```
/* ----------------------------------------------------------------------------------------
File Name: thread_3.c
Description: In this program the pthread_create() takes an argument that is
                pointer to void which passes to start routine referred in
                pthread_create() function.
        Author: Embedded Linux Team, EC Department, CSPIT, CHARUSAT
   Date & Time: 05th April, 2019 & 10:36 am.
----------------------------------------------------------------------------------------*/



#include<stdio.h>
#include<string.h>
#include<pthread.h>
#include<stdlib.h>
#include<unistd.h>
void *thread_1(void *arg)
{
        printf("INT=%d\n",*(int*)arg);
        while(1)
        {  printf("In thread 1\n");
                sleep(1);
        }
}
void main()
{
        pthread_t tid1;
        int no=10;
        /* passing argument as pointer to void to function as thread_1*/
        pthread_create(&tid1,NULL,thread_1,&no);
        while(1)
        {
                printf("In main thread\n");
                sleep(1);
        }
}
```

RESULT:

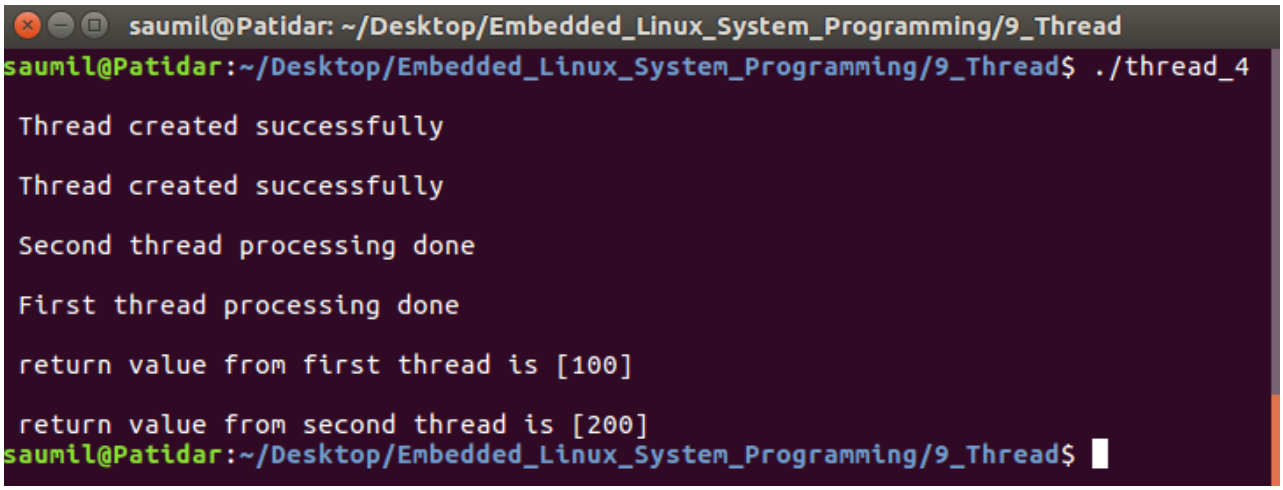


Example Program 4:

```
/* ------------------------------------------------------------------------------------------
File Name: thread_4.c
Description: In this program two threads are created, and after execution of
             each thread, they are terminated using pthread_exit() and
             return value to pointer passed through pthread_join().
        Author: Embedded Linux Team, EC Department, CSPIT, CHARUSAT
   Date & Time: 05th April, 2019 & 10:36 am.
------------------------------------------------------------------------------------------*/


#include<stdio.h>
#include<string.h>
#include<pthread.h>
#include<stdlib.h>
#include<unistd.h>

pthread_ttid[2];
int ret1,ret2;
void *fun(void *arg)
{
        unsigned long i=0;
        /*returns ID of thread in which it is invoked*/
        pthread_t id = pthread_self();
        for(i=0;i<(0xFFFFFFFF);i++);
        /*Compare two threads, returns non-zero if equal else zero.*/
        if(pthread_equal(id,tid[0]))
        {
```

```
		printf("\n First thread processing done\n");
		ret1=100;

		/*Terminates thread 1 and returns ret1*/
		pthread_exit(&ret1);
	}
	else
	{
		printf("\n Second thread processing done\n");
		ret2=200;
		/*Terminates thread 2 and returns ret2*/
		pthread_exit(&ret2);
	}
	return NULL;
}

int main(void)
{
	inti=0;
	int err;
	int *ptr[2];
	while(i<2)
	{
		err=pthread_create(&tid[i],NULL,&fun,NULL);
		/*Returns non-zero value if thread is not created successfully*/
		if(err!=0)
			printf("\ncan't create thread:[%s]",strerror(err));
		else
			printf("\n Thread created successfully\n");
		i++;
	}
	/*waits for termination of thread and returns value to respective
pointer*/
	pthread_join(tid[0],(void**)&(ptr[0]));
	pthread_join(tid[1],(void**)&(ptr[1]));

	printf("\n return value from first thread is [%d]\n",*ptr[0]);
	printf("\n return value from second thread is [%d]\n",*ptr[1]);
	return 0;
}
```

RESULT:

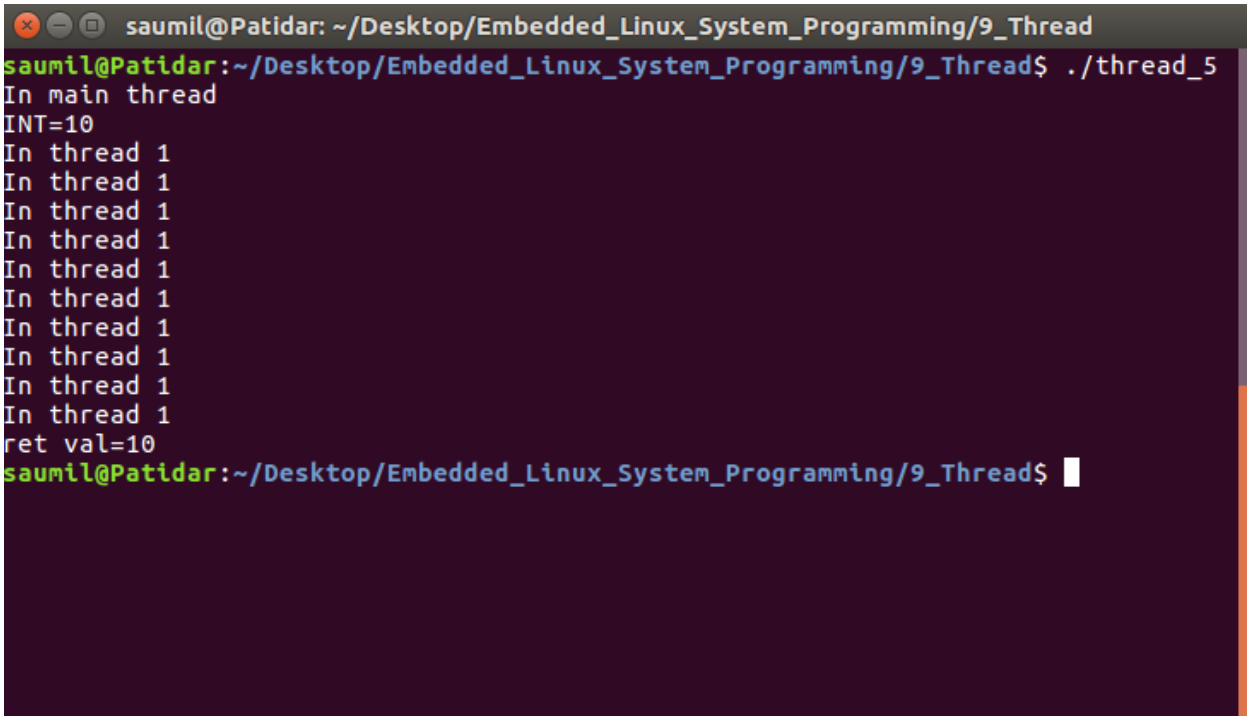


Example Program 5:

```
/* ------------------------------------------------------------------------------------
File Name: thread_5.c
Description: In this program we pass argument and iterates thread function
             on the basis of value of variable initialized globally and
             incremented in while() function. When condition is met
             thread is terminated using pthread_exit() and return value is
             obtained using pthread_join() .
        Author: Embedded Linux Team, EC Department, CSPIT, CHARUSAT
   Date & Time: 05th April, 2019 & 10:36 am.
------------------------------------------------------------------------------------*/


#include<stdio.h>
#include<string.h>
#include<pthread.h>
#include<stdlib.h>
#include<unistd.h>

int c=0;
void *thread_1(void *arg)
{
        printf("INT=%d\n",*(int*)arg);
        while(1)
        {
                c++;
                printf("In thread 1\n");
                sleep(1);
```

```
                if(c==10)
                {
                        break;
                }
        }
         /*Terminates thread and returns c*/
pthread_exit(&c);
}

void main()
{
        pthread_t tid1;
        inti=10;
        void *c;
        /*Creates thread tid1 passing i as an argument to thread_1()
function*/
        pthread_create(&tid1,NULL,thread_1,&i);
        printf("In main thread\n");
        /*waits for termination of thread and returns value c*/
        pthread_join(tid1,&c);
        printf("ret val=%d\n",*(int*)c);
}
```

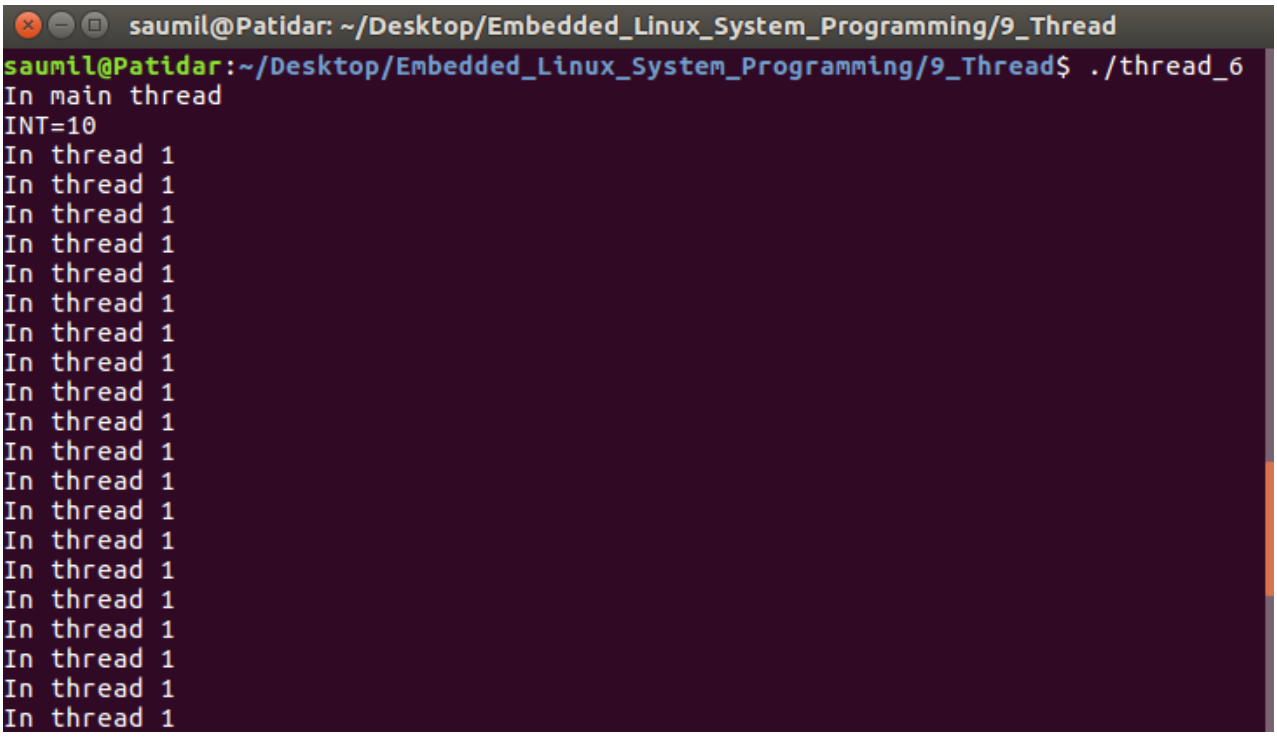
RESULT:



Example Program 6:

```
/* ---------------------------------------------------------------------------------------
File Name: thread_6.c
Description: In this program, the routine for thread is executed infinitely as
             pthread_exit() will not be able to get any argument to
             terminate the thread and pthread_join() will not be executed.
      Author: Embedded Linux Team, EC Department, CSPIT, CHARUSAT
   Date & Time: 05th April, 2019 & 10:36 am.
---------------------------------------------------------------------------------------*/


#include<stdio.h>
#include<string.h>
#include<pthread.h>
#include<stdlib.h>
#include<unistd.h>

int c=0;
void *thread_1(void *arg)
{
        printf("INT=%d\n",*(int*)arg);
        while(1)
        {
                c++;
                printf("In thread 1\n");
                sleep(1);
        }
        /*Terminates thread and returns c*/
        pthread_exit(&c);
}

void main()
{
        pthread_t tid1;
        inti=10;
        void *c;
        /* Creates thread with argument i*/
        pthread_create(&tid1,NULL,thread_1,&i);
        printf("In main thread\n");
        sleep(5);
        /*waits for termination of thread and returns value c*/
        pthread_join(tid1,&c);
        printf("ret val=%d\n",*(int*)c);
}
```

RESULT:


saumil@Patidar: ~/Desktop/Embedded_Linux_System_Programming/9_Thread
saumil@Patidar:~/Desktop/Embedded_Linux_System_Programming/9_Thread$ ./thread_6
In main thread
INT=10
In thread 1
In thread 1
In thread 1
In thread 1
In thread 1
In thread 1
In thread 1
In thread 1
In thread 1
In thread 1
In thread 1
In thread 1
In thread 1
In thread 1
In thread 1
In thread 1
In thread 1
In thread 1
In thread 1
In thread 1

Experiment No.12

Aim: Inter process communication (IPC) :Semaphore and Mutex.

Theory:

A thread is a single sequence stream within in a process.Threads are not independent of one other like processes as a result threads shares with other threads their code section, data section and OS resources like open files and signals. Threads are popular way to improve application through parallelism.

Semaphore :

Semaphores are a technique for coordinating activities in which multiple processes or threads compete for the same operating system resources. A semaphore is a value in a designated place in operating system (or kernel) storage that each process can check and then change. Depending on the value that is found, the process can use the resource or will find that it is already in use and must wait for some period before trying again. Typically, a process using semaphores checks the value and then, if it using the resource, changes the value to reflect this so that subsequent semaphore users will know to wait.Semaphores are one of the techniques for inter-process communication (IPC).

Mutex(Binary Semaphore) :

Mutex is a lock for Linux thread synchronization. Technically it is a mechanism which ensures that two or more concurrent processes or threads do not simultaneously execute some particular program segment known as critical section. Processes access to critical section is controlled by using mutex. When one thread starts executing the critical section,the other thread should wait until the first thread finishes. If proper synchronization techniques are not applied, it may cause a race condition where the values of variables may be unpredictable and vary depending on the timings of context switches of the processes or threads.

Difference between Semaphore and Mutex :

Semaphore	Mutex
Semaphore is signalling mechanism. It allows a number of thread to access shared resources.	Mutex is a object owned by thread,so there is ownership in mutex. Mutex allow only one thread to access resource.
It is a generalized mutex. In line of single buffer, we can split the 4 KB buffer into four 1 KB buffers (identical resources). A semaphore can be associated with these four buffers. The consumer and producer can work on different buffers at the same time.	It provides mutual exclusion, either producer or consumer can have the key (mutex) and proceed with their work. As long as the buffer is filled by producer, the consumer needs to wait, and vice versa.At any point of time, only one thread can work with the entire buffer.

Advantages:

- Better communication among threads.
- While working on critical sections of program, other threads are blocked.

Disadvantages:

- Code complexity increases.
- In semaphore while working on critical sections of program, other threads might be blocked, which can be resolved using mutex.

List of System calls used for mutex:

intpthread_mutex_init(pthread_mutex_t *restrict mutex, constpthread_mutexattr_t *restrict attr)	Creates a mutex, referenced by mutex, with attributes specified by attr. If attr is NULL, the default mutex attribute (NONRECURSIVE) is used.Successful, pthread_mutex_init() returns 0, and the state of the mutex becomes initialized and unlocked else returns -1 for unsuccessful ones
intpthread_mutex_lock(pthread_mutex_t *mutex)	Locks a mutex object, which identifies a mutex. If the mutex is already locked by another thread, the thread waits for the mutex to become available. The thread that has locked a mutex becomes its current owner and remains the owner until the same thread has unlocked it. Successful, pthread_mutex_lock() returns 0 else returns -1.
intpthread_mutex_unlock(pthread_mutex_t *mutex)	Releases a mutex object. If one or more threads are waiting to lock the mutex, pthread_mutex_unlock() causes one of those threads to return from pthread_mutex_lock() with

	the mutex object acquired. If no threads are waiting for the mutex, the mutex unlocks with no current owner.Successful, pthread_mutex_unlock() returns 0 else returns -1.
PTHREAD_MUTEX_INITIALIZER	The pthread_mutex_init() function shall initialize the mutex referenced by mutex with attributes specified by attr.

Example Program 1:

```
/* ---------------------------------------------------------------------------------------
File Name: mutex_1.c
Description: This example program demonstrate use of mutex in
             multithreading to create a definite sequential outptut where
             main loop, thread 2 and thread 1 are executed in fixed
             sequential manner with the use of locks and unlocks.
Author: Embedded Linux Team, EC Department, CSPIT, CHARUSAT
Date & Time: 2nd April, 2019 & 11.30 pm.
---------------------------------------------------------------------------------------*/

#include<stdio.h>
#include<string.h>
#include<pthread.h>
#include<stdlib.h>
#include<unistd.h>

pthread_mutex_tmlock=PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t mlock1=PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t mlock2=PTHREAD_MUTEX_INITIALIZER;

void *thread_1(void *arg)
{
        charch;
        for(ch='a';ch<='z';ch++)
        {
//Locks mutex object ‘mlock1’
```

```
                pthread_mutex_lock(&mlock1);
                printf("%c\n",ch);
                sleep(1);
//Unlocks mutex object 'mlock1' which creates a scope for 'for loop' in
main() to execute
                pthread_mutex_unlock(&mlock);
        }
}

void *thread_2(void *arg)
{
        inti;
        for(i=0;i<26;i++)
        {
//Locks mutex object 'mlock2'
                pthread_mutex_lock(&mlock2);
                printf("%d\n",i);
                sleep(1);
//Unlocks mutex object 'mlock1' which creates a scope for thread 1 to
execute
                pthread_mutex_unlock(&mlock1);
        }
}

void main()
{
        pthread_ttid1[2];
        charch;

//locks both mutex object
        pthread_mutex_lock(&mlock2);
        pthread_mutex_lock(&mlock1);

        pthread_create(&tid1[0],NULL,thread_1,NULL);
        pthread_create(&tid1[1],NULL,thread_2,NULL);


        for(ch='A';ch<='Z';ch++)
        {
//locks mutex object 'mlock'
                pthread_mutex_lock(&mlock);
                printf("%c\n",ch);
                sleep(1);
//unlocks mutex object 'mlock2' which creates a scope for thread 2 to
execute
                pthread_mutex_unlock(&mlock2);
```

```
        }
}
```

Execution Steps:

```
soham@soham-HP-Notebook:~/10_Mutex$ ls
mutex_1.c  mutex_2.c
soham@soham-HP-Notebook:~/10_Mutex$ gcc -o mutex_1 mutex_1.c -lpthread
soham@soham-HP-Notebook:~/10_Mutex$ ls
mutex_1  mutex_1.c  mutex_2.c
soham@soham-HP-Notebook:~/10_Mutex$ ./mutex_1
```

Result:

```
soham@soham-HP-Notebook:~/10_Mutex$ ls
mutex_1.c  mutex_2.c
soham@soham-HP-Notebook:~/10_Mutex$ gcc -o mutex_1 mutex_1.c -lpthread
soham@soham-HP-Notebook:~/10_Mutex$ ls
mutex_1  mutex_1.c  mutex_2.c
soham@soham-HP-Notebook:~/10_Mutex$ ./mutex_1
A
0
a
B
1
b
C
2
c
D
3
d
E
4
e
F
5
f
G
6
g
H
7
h
I
8
i
J
9
j
K
10
```

Example Program 2:

```
/* ---------------------------------------------------------------------------------------
File Name: mutex_2.c
Description: This example program demonstrates use of mutex to perform 2
             tasks in alternative manner. Here 1 letter is printed on
             screen then 1 letter is written in file 'data' created by
             program. This procedure continues in alternating manner
             with the use of locks and unlocks until all letters are
             printed.
Author: Embedded Linux Team, EC Department, CSPIT, CHARUSAT
Date & Time: 2nd April, 2019 & 11:35 pm.
---------------------------------------------------------------------------------------*/

#include<stdio.h>
#include<string.h>
#include<pthread.h>
#include<stdlib.h>
#include<unistd.h>

pthread_mutex_tmlock,mlock1;

FILE *fp;

void *thread_1(void *arg)
{
        charch;
        for(ch='a';ch<='z';ch++)
        {

//locks mutex object 'mlock1'
                pthread_mutex_lock(&mlock1);
                fprintf(fp,"%c",ch);

//Unlocks mutex object 'mlock' which creates a scope for 'for loop' in main()
to execute
                pthread_mutex_unlock(&mlock);
        }
}

void main()
{
        pthread_t tid1;
        charch;
```

```
//creates mutex objects
        pthread_mutex_init(&mlock1,NULL);
        pthread_mutex_init(&mlock,NULL);

//locks mutex object 'mlock1'
        pthread_mutex_lock(&mlock1);

        pthread_create(&tid1,NULL,thread_1,NULL);

//creating a file name 'data' in write mode
        fp=fopen("data","w");

        for(ch='A';ch<='Z';ch++)
        {

//locks mutex object 'mlock'
                pthread_mutex_lock(&mlock);
                printf("%c",ch);

//unlocks mutex object 'mlock1' which creates a scope for thread 1 to
execute
                pthread_mutex_unlock(&mlock1);
        }
}
```

Execution Steps:

```
soham@soham-HP-Notebook:~/10_Mutex$ ls
mutex_1  mutex_1.c  mutex_2.c
soham@soham-HP-Notebook:~/10_Mutex$ gcc -o mutex_2 mutex_2.c -lpthread
soham@soham-HP-Notebook:~/10_Mutex$ ls
mutex_1  mutex_1.c  mutex_2  mutex_2.c
soham@soham-HP-Notebook:~/10_Mutex$ ./mutex_2
```

Result:

```
soham@soham-HP-Notebook:~/10_Mutex$ ls
mutex_1  mutex_1.c  mutex_2.c
soham@soham-HP-Notebook:~/10_Mutex$ gcc -o mutex_2 mutex_2.c -lpthread
soham@soham-HP-Notebook:~/10_Mutex$ ls
mutex_1  mutex_1.c  mutex_2  mutex_2.c
soham@soham-HP-Notebook:~/10_Mutex$ ./mutex_2
ABCDEFGHIJKLMNOPQRSTUVWXYZsoham@soham-HP-Notebook:~/10_Mutex$ ls
data  mutex_1  mutex_1.c  mutex_2  mutex_2.c
soham@soham-HP-Notebook:~/10_Mutex$ cat data
abcdefghijklmnopqrstuvwxyzsoham@soham-HP-Notebook:~/10_Mutex$
```

Experiment No. 13

Aim: Implementation of Linux Kernel Moduling.

A) Linux kernel moduling: helloworld.

Theory: Kernel: A kernel is actually a large block of code which keeps the system up and running from the time of booting, till shutdown. Kernel is that part of an OS which directly communicates with the hardware of the machine in which it runs and also with external hardware devices.The kernel can be viewed as resource managing code of an OS. It is responsible for managing and allocating resources like memory, processors etc. It also allows OS to communicate and control various external hardware devices like pendrive, memory card, keyboard etc.

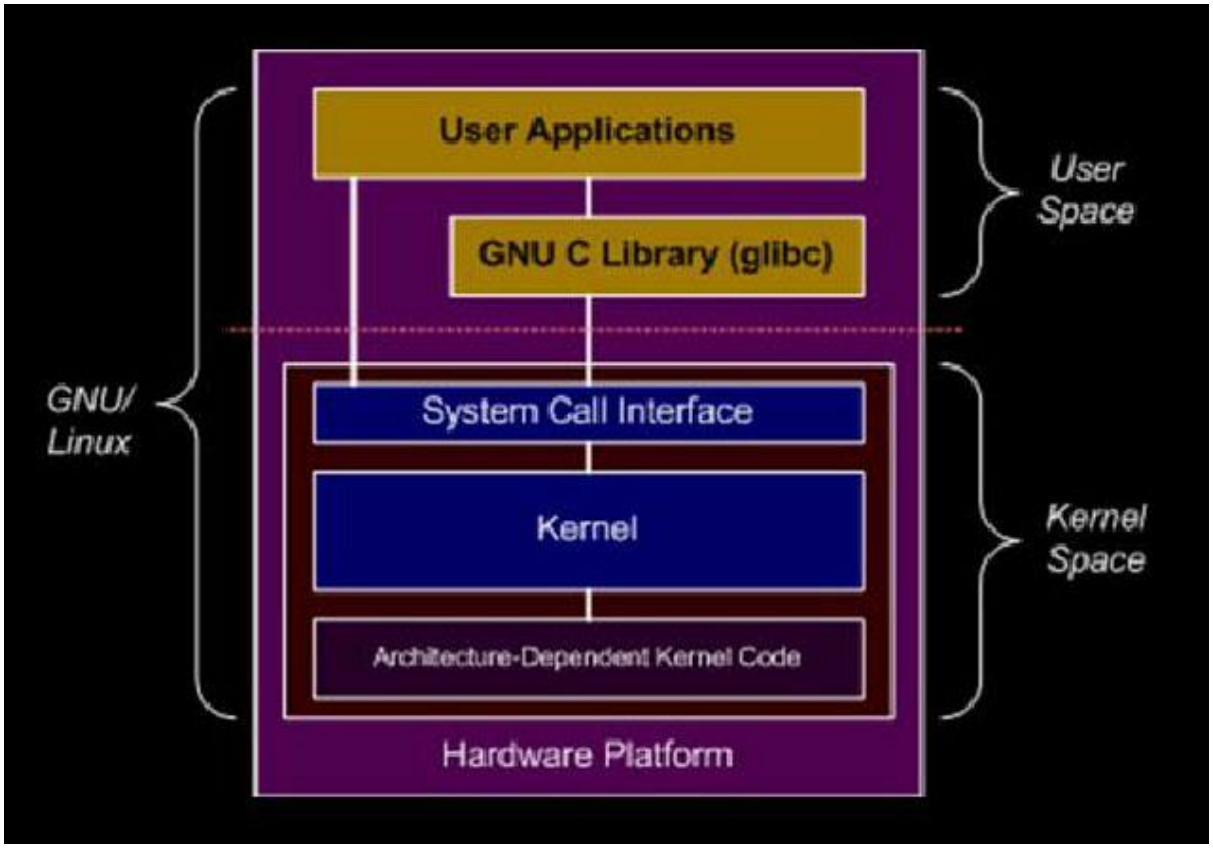


Fig. 1

- Systemmemory inLinux can be divided into two distinct regions: kernel space anduser space. Kernel space is where thekernel (i.e., the core of the operating system) executes (i.e., runs) and provides its services.

- User space is that set of memory locations in which user processes (i.e., everything other than the kernel) run. Aprocess is an executing instance of a program. One of the roles of the kernel is to manage individual user processes within this space and to prevent them from interfering with each other.

- Kernel modules run in kernel space and applications run in user space, as illustrated in Figure 1. Both kernel space and user space have their own unique memory address spaces that do not overlap. This approach ensures that applications running in user space have a consistent view of the hardware, regardless of the hardware platform. The kernel services are then made available to the user space in a controlled way through the use of system calls. The kernel also prevents individual user-space applications from conflicting with each other or from accessing restricted resources through the use of protection levels (e.g., superuser versus regular user permissions).

- Kernel modules are pieces of code that can be loaded and unloaded into the kernel upon demand. They extend the functionality of the kernel without the need of rebooting the system.

Custom codes can be added to Linux kernels via two methods :

- The basic way is to add the code to the kernel source tree and recompile the kernel. A more efficient way is to do this is by adding code to the kernel while it is running. This process is called loading the module, where module refers to the code that we want to add to the kernel.

- Since we are loading these codes at runtime and they are not part of the official Linux kernel, these are called loadable kernel module(LKM), which is different from the “base kernel”. Base kernel is

located in /boot directory and is always loaded when we boot our machine whereas LKMs are loaded after the base kernel is already loaded. Nevertheless, these LKM are very important part of our kernel and they communicate with base kernel to complete their functions.

LKMs can perform a variety of task, but basically they come under three main categories,

- device driver
- file system driver
- System calls

Advantages:

- One major advantage they have is that we don't need to keep rebuilding the kernel every time we add a new device or if we upgrade an old device.
- This saves time and also helps in keeping our base kernel error free. A useful rule of thumb is that we should not change our base kernel once we have a working base kernel.
- LKMs are very flexible, in the sense that they can be loaded and unloaded with a single line of command.

Disadvantages:

- This has the disadvantage of requiring us to rebuild and reboot the kernel every time we want new functionality.

List of Macros:

module_init ()	The __init macro causes the init function to be discarded and its memory freed once the init function finishes for built–in drivers, but not loadable modules. If you think about when the init function is invoked, this makes perfect sense. There is also an __initdata which works similarly to __init but for init variables rather than functions.

module_exit()	The __exit macro causes the omission of the function when the module is built into the kernel, and like __exit, has no effect for loadable modules. Again, if you consider when the cleanup function runs, this makes complete sense; built-in drivers don't need a cleanup function, while loadable modules do.

Example Program:

```
/* ----------------------------------------------------------------------------------------
File Name: ex01_simple_module.c
Description: This example program demonstrate linux kernel moduling hello
                world!!. Here, when the module inserted the hello world
                message is displayed and when module is removed the good
                bye message is displayed.
          Author: Embedded Linux Team, EC Department, CSPIT, CHARUSAT
         Date& Time :20th March,2019 & 10:10 a.m.
----------------------------------------------------------------------------------------*/
#include<linux/init.h>

#include<linux/module.h>

MODULE_LICENSE('GPL');

int ex01_simple_module_init(void)

{

printk(KERN_ALERT"Hello wold\n");

return 0;

}

void ex01_simple_module_exit(void)

{

printk(KERN_ALERT"Good bye!!\n");

        return 0;
```

```
}
module_init(ex01_simple_module_init);
module_exit(ex01_simple_module_exit);
```

Make file

```
Obj –m := helloworld.o
```

Execution Steps

Step 1: for inserting the module in kernel space

```
tej@bapalal:~/Desktop/new$ make -C /lib/modules/$(uname -r)/build M=$PWD cleanmake: Entering directory '/usr/src/linux-headers-4.8.0-36-generic
'
  CLEAN   /home/tej/Desktop/new/.tmp_versions
  CLEAN   /home/tej/Desktop/new/Module.symvers
make: Leaving directory '/usr/src/linux-headers-4.8.0-36-generic'
tej@bapalal:~/Desktop/new$ make -C /lib/modules/$(uname -r)/build M=$PWD modulesmake: Entering directory '/usr/src/linux-headers-4.8.0-36-gener
ic'
  CC [M]  /home/tej/Desktop/new/ex01_simple_module.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/tej/Desktop/new/ex01_simple_module.mod.o
  LD [M]  /home/tej/Desktop/new/ex01_simple_module.ko
make: Leaving directory '/usr/src/linux-headers-4.8.0-36-generic'
tej@bapalal:~/Desktop/new$ sudo insmod ex01_simple_module.ko
[sudo] password for tej:
tej@bapalal:~/Desktop/new$ dmesg
```

Step 2: for removing the module from the kernel space

```
tej@bapalal:~/Desktop/new$ sudo rmmod ex01_simple_module.ko
tej@bapalal:~/Desktop/new$ dmesg
```

Result:

```
[  268.930976] hello world
[  288.037498] good bye
[  332.755757] hello world
[  374.565911] good bye
tej@bapalal:~/Desktop/new$
```

B)Linux kernel module: Parameter Passing

Theory:

A kernel module can accept arguments from the command line. This allows dynamically changing the behaviour of the module according to the given parameters, and can avoid the developer having to indefinitely change/compile the module during a test/debug session. In order to set this up, you should first declare the variables that will hold the values of command line arguments, and use the module_param() macro on each of these. The macro is defined in include/Linux/moduleparam.h

Macro:

module_Init()	The __init macro causes the init function to be discarded and its memory freed once the init function finishes for built–in drivers, but not loadable modules.There is also an __initdata which works similarly to __init but for init variables rather than functions. The return value for i.
module_exit()	The __exit macro causes the omission of the function when the module is built into the kernel, and like __exit, has no effect for loadable modules. Again, if you consider when the cleanup function runs, this makes complete sense; built–in drivers don't need a cleanup function, while loadable modules do.
module_param()	module_param (name, type, perm); In this macro the 1st parameter is the name of the variable which is already defined, 2nd parameter is the type of that variable and The perm parameter was set aside for the

	sys/module representation of this parameter and we have to give the permissions (writable, readable & executable) to this parameter.
MODULE_LICENSE('GPL')	The mechanism was devised to identify code licensed under the GPL (and friends) so people can be warned that the code is non open-source. This is accomplished by the MODULE_LICENSE() macro. By setting the license to GPL, you can keep the warning from being printed. This license mechanism is defined and documented in linux/module.h.

Example Program:

```
/* ----------------------------------------------------------------------------------------

File Name: Ex06_module_Param.c

Description: This example program demonstratedLinux kernel module
             programming to pass the parameter. In this program we
             passing the parameters to a loadable kernel module and
             modifying the module parameter after module is loaded.

Author: Embedded Linux Team, EC Department, CSPIT, CHARUSAT

Date & Time:20th March,2019 & 10:10 a.m.

----------------------------------------------------------------------------------------*/

#include<linux/init.h>

#include<linux/module.h>

#include<linux/moduleparam.h>

MODULE_LICENSE("GPL");

int count=1;
```

```
//We can pass parameters in thismodule.
module_param(count,int,0644);
int ex06_simple_module_init(void)
{
   int index;
printk(KERN_ALERT "Inside %s function",__FUNCTION__);
   for(index=0;index<count;index++)
    {
        printk(KERN_ALERT"HELLO World: index=%d\n",index);
    }
   return 0;
}
void ex06_simple_module_exit (void)
{
   int index;
printk(KERN_ALERT "Inside %s function",__FUNCTION__);
   for(index=0;index<count;index++)
    {
        printk(KERN_ALERT"Good Bye!!: index=%d\n",index);
    }
}
module_init(ex06_simple_module_init);
module_exit(ex06_simple_module_exit);
```

Make file

Obj –m := Ex06_module_Param.o

Execution Steps:

Terminal 1:

```
vidhi@vidhi-HP-280-G1-MT:~$ cd Desktop
vidhi@vidhi-HP-280-G1-MT:~/Desktop$ cd ex06
vidhi@vidhi-HP-280-G1-MT:~/Desktop/ex06$ gedit Ex06_module_Param.ko
^C
vidhi@vidhi-HP-280-G1-MT:~/Desktop/ex06$ gedit Makefile^C
vidhi@vidhi-HP-280-G1-MT:~/Desktop/ex06$ make -C /lib/modules/$(uname -r)/build M=$PWD clean
make: Entering directory '/usr/src/linux-headers-4.8.0-36-generic'
  CLEAN   /home/vidhi/Desktop/ex06/.tmp_versions
  CLEAN   /home/vidhi/Desktop/ex06/Module.symvers
make: Leaving directory '/usr/src/linux-headers-4.8.0-36-generic'
vidhi@vidhi-HP-280-G1-MT:~/Desktop/ex06$ make -C /lib/modules/$(uname -r)/build M=$PWD modules
make: Entering directory '/usr/src/linux-headers-4.8.0-36-generic'
  CC [M]  /home/vidhi/Desktop/ex06/Ex06_module_Param.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/vidhi/Desktop/ex06/Ex06_module_Param.mod.o
  LD [M]  /home/vidhi/Desktop/ex06/Ex06_module_Param.ko
make: Leaving directory '/usr/src/linux-headers-4.8.0-36-generic'
vidhi@vidhi-HP-280-G1-MT:~/Desktop/ex06$ ls *.ko
Ex06_module_Param.ko
vidhi@vidhi-HP-280-G1-MT:~/Desktop/ex06$ sudo tail -f /var/log/syslog
[sudo] password for vidhi:
```

Terminal 2:

```
vidhi@vidhi-HP-280-G1-MT:~$ cd Desktop
vidhi@vidhi-HP-280-G1-MT:~/Desktop$ cd ex06
vidhi@vidhi-HP-280-G1-MT:~/Desktop/ex06$ sudo insmod Ex06_module_Param.ko
vidhi@vidhi-HP-280-G1-MT:~/Desktop/ex06$ sudo rmmod Ex06_module_Param.ko
vidhi@vidhi-HP-280-G1-MT:~/Desktop/ex06$ sudo insmod Ex06_module_Param.ko count=5
vidhi@vidhi-HP-280-G1-MT:~/Desktop/ex06$ sudo rmmod Ex06_module_Param.ko
vidhi@vidhi-HP-280-G1-MT:~/Desktop/ex06$
```

Result:

```
Mar 31 19:31:30 vidhi-HP-280-G1-MT kernel: [  819.532708] HELLO World: index=0
Mar 31 19:31:39 vidhi-HP-280-G1-MT kernel: [  827.998249] Inside ex06_simple_module_exit function
Mar 31 19:31:39 vidhi-HP-280-G1-MT kernel: [  827.998252] Good Bye!!: index=0
Mar 31 19:31:46 vidhi-HP-280-G1-MT kernel: [  835.751046] Inside ex06_simple_module_init function
Mar 31 19:31:46 vidhi-HP-280-G1-MT kernel: [  835.751049] HELLO World: index=0
Mar 31 19:31:46 vidhi-HP-280-G1-MT kernel: [  835.751051] HELLO World: index=1
Mar 31 19:31:46 vidhi-HP-280-G1-MT kernel: [  835.751053] HELLO World: index=2
Mar 31 19:31:46 vidhi-HP-280-G1-MT kernel: [  835.751054] HELLO World: index=3
Mar 31 19:31:46 vidhi-HP-280-G1-MT kernel: [  835.751054] HELLO World: index=4
Mar 31 19:31:49 vidhi-HP-280-G1-MT kernel: [  838.690240] Inside ex06_simple_module_exit function
Mar 31 19:31:49 vidhi-HP-280-G1-MT kernel: [  838.690244] Good Bye!!: index=0
Mar 31 19:31:49 vidhi-HP-280-G1-MT kernel: [  838.690246] Good Bye!!: index=1
Mar 31 19:31:49 vidhi-HP-280-G1-MT kernel: [  838.690247] Good Bye!!: index=2
Mar 31 19:31:49 vidhi-HP-280-G1-MT kernel: [  838.690248] Good Bye!!: index=3
Mar 31 19:31:49 vidhi-HP-280-G1-MT kernel: [  838.690249] Good Bye!!: index=4
Mar 31 19:31:58 vidhi-HP-280-G1-MT dbus[917]: [system] Activating via systemd: service name='org.freedesktop.hostname1' unit='dbus-org.freedesktop.hostname1.service'
Mar 31 19:31:58 vidhi-HP-280-G1-MT systemd[1]: Starting Hostname Service...
Mar 31 19:31:58 vidhi-HP-280-G1-MT dbus[917]: [system] Successfully activated service 'org.freedesktop.hostname1'
Mar 31 19:31:58 vidhi-HP-280-G1-MT systemd[1]: Started Hostname Service.
```

C)Linux Kernel Moduling- Exporting symbols.

Theory:

The kernel's loadable module mechanism does not give modules access to all parts of the kernel. Instead, any kernel symbol which is intended to be usable by loadable modules, must be explicitly exported to them via one of the variants of the EXPORT_SYMBOL() macro.

Linux kernel allows modules stacking, which basically means one module can use the symbols defined in other modules. But this is possible only when the symbols are exported.

A symbol can be exported by a module using the macro EXPORT_SYMBOL().

List of Macros:

module_init ()	The __init macro causes the init function to be discarded and its memory freed once the init function finishes for built−in drivers, but not loadable modules. If you think about when the init function is invoked, this makes perfect sense. There is also an __initdata which works similarly to __init but for init variables rather than functions.
module_exit()	The __exit macro causes the omission of the function when the module is built into the kernel, and like __exit, has no effect for loadable modules. Again, if you consider when the cleanup function runs, this makes complete sense; built−in drivers don't need a cleanup function, while loadable modules do.
EXPORT_SYMBOL()	EXPORT_SYMBOL() macro is used to make kernel symbols available to loadable modules; only the symbols that have been explicitly exported

	can by used by modules.

Example Program:

```
/* ---------------------------------------------------------------------------------------
File Name: ex05_exporting_symbol.c
Description: This program file is to create a exporting module which is going
             to export a function. And allow function to pass function
             name and variable in kernel module.
Author: Embedded Linux Team, EC Department, CSPIT, CHARUSAT
Date & Time: 20th March, 2019 & 10:10 am.
---------------------------------------------------------------------------------------*/

#include<linux/init.h>
#include<linux/module.h>
//To export symbol in kernel module
int ex05_simple_module_function(void)
{0
        printk(KERN_ALERT "Inside %s function",__FUNCTION__);
        return 0;
}
//Function defined to insert module
int ex05_simple_module_init(void)
{
        printk(KERN_ALERT "Inside %s function",__FUNCTION__);
        return 0;
}
//Fuction defined to exit module
void ex05_simple_module_exit(void)
{
        printk(KERN_ALERT "Inside %s function",__FUNCTION__);
}
//Macro to export symbols
EXPORT_SYMBOL(ex05_simple_module_function);
//Inserting module in kernel space
module_init(ex05_simple_module_init);
//Removing the module from kernel space
module_exit(ex05_simple_module_exit);
```

```
/* ------------------------------------------------------------------------------------
File Name: ex05_using_symbol.c
Description: This Program file creates a function used to pass the function
                name from previous exporting module.
Author: Embedded Linux Team, EC Department, CSPIT, CHARUSAT
Date & Time: 20th March, 2019 & 10:10 am.
------------------------------------------------------------------------------------*/

#include<linux/init.h>
#include<linux/module.h>
//using the exported funtion

int ex05_simple_module_function(void);
int ex05_simple_module_init(void)
{
        printk(KERN_ALERT "Inside %s function",__FUNCTION__);
        ex05_simple_module_function();
        return 0;
}
void ex05_simple_module_exit(void)
{
        printk(KERN_ALERT "Inside %s function",__FUNCTION__);
}
module_init(ex05_simple_module_init);
module_exit(ex05_simple_module_exit);

/* ------------------------------------------------------------------------------------
File Name: Makefile
Description: Makefile is use to compile a module and create .o file of
                modules and .ko file from .o file.
Author: Embedded Linux Team, EC Department, CSPIT, CHARUSAT
Date & Time: 20th March, 2019 & 10:10 am.
------------------------------------------------------------------------------------*/

obj-m := ex05_exporting_symbol.o
obj-m += ex05_using_symbol.o
```

Execution Steps:

```
vraj@vraj:~$ cd Desktop
vraj@vraj:~/Desktop$ cd new/
vraj@vraj:~/Desktop/new$ gedit ex05_exporting_symbol.c
vraj@vraj:~/Desktop/new$ gedit ex05_using_symbol.c
vraj@vraj:~/Desktop/new$ gedit Makefile
vraj@vraj:~/Desktop/new$ make -C /lib/modules/$(uname -r)/build M=$PWD clean
make: Entering directory '/usr/src/linux-headers-4.8.0-36-generic'
  CLEAN   /home/vraj/Desktop/new/.tmp_versions
  CLEAN   /home/vraj/Desktop/new/Module.symvers
make: Leaving directory '/usr/src/linux-headers-4.8.0-36-generic'
vraj@vraj:~/Desktop/new$ make -C /lib/modules/$(uname -r)/build M=$PWD modules
make: Entering directory '/usr/src/linux-headers-4.8.0-36-generic'
  CC [M]  /home/vraj/Desktop/new/ex05_exporting_symbol.o
  CC [M]  /home/vraj/Desktop/new/ex05_using_symbol.o
  Building modules, stage 2.
  MODPOST 2 modules
  CC      /home/vraj/Desktop/new/ex05_exporting_symbol.mod.o
  LD [M]  /home/vraj/Desktop/new/ex05_exporting_symbol.ko
  CC      /home/vraj/Desktop/new/ex05_using_symbol.mod.o
  LD [M]  /home/vraj/Desktop/new/ex05_using_symbol.ko
make: Leaving directory '/usr/src/linux-headers-4.8.0-36-generic'
```

Result:

```
vraj@vraj:~/Desktop/new$ ls *.ko
ex05_exporting_symbol.ko  ex05_using_symbol.ko
vraj@vraj:~/Desktop/new$ sudo insmod ./ex05_exporting_symbol.ko
[sudo] password for vraj:
vraj@vraj:~/Desktop/new$ sudo insmod ./ex05_using_symbol.ko
vraj@vraj:~/Desktop/new$ lsmod
Module                  Size  Used by
ex05_using_symbol      16384  0
ex05_exporting_symbol    16384  1 ex05_using_symbol
ccm                    20480  1
rfcomm                 77824  0
bnep                   20480  2
joydev                 20480  0
asus_nb_wmi            24576  0
asus_wmi               28672  1 asus_nb_wmi
sparse_keymap          16384  1 asus_wmi
intel_rapl             20480  0
x86_pkg_temp_thermal    16384  0
coretemp               16384  0
kvm_intel             192512  0
```

Exported symbol is used by using module.

```
[  182.143212] sd 3:0:0:0: [sdc] No Caching mode page found
[  182.143218] sd 3:0:0:0: [sdc] Assuming drive cache: write through
[  182.143220] sd 3:0:0:0: Attached scsi generic sg3 type 0
[  182.332356]  sdc: sdc1
[  182.333271] sd 3:0:0:0: [sdc] Attached SCSI removable disk
[  182.790086] FAT-fs (sdc1): Volume was not properly unmounted. Some data may b
e corrupt. Please run fsck.
[  380.161473] Inside ex05_simple_module_exit function
[  382.534573] Inside ex05_simple_module_exit function
[  409.614672] Inside ex05_simple_module_init function
[  417.748136] Inside ex05_simple_module_init function
[  417.748142] Inside ex05_simple_module_function function
[  434.083653] Inside ex05_simple_module_exit function
```

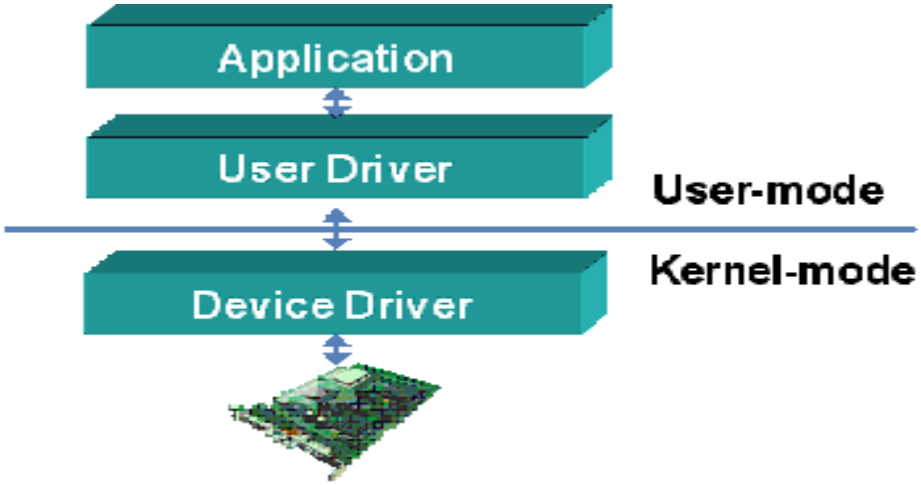
Experiment No.14

Aim: Implementation of character device driver in Linux.

Theory:

Device Drivers are the software through which, the kernel of a computer communicates with different hardware, without having to go into the details of how the hardware works. It is a software that controls a hardware part attached to a computer and allows the computer to use the hardware by providing a suitable interface. This means that the operating system need not go into the details about how the hardware part works. It also provides a common interface so that the operating system or the kernel can communicate with the hardware.

Thus, the purpose of device drivers is to allow smooth functioning of the hardware for which it is created and to allow it to be used with different operating systems.



There are two types of device driver:

(1) Character device driver :

Examples : serial ports, parallel ports, sounds cards.

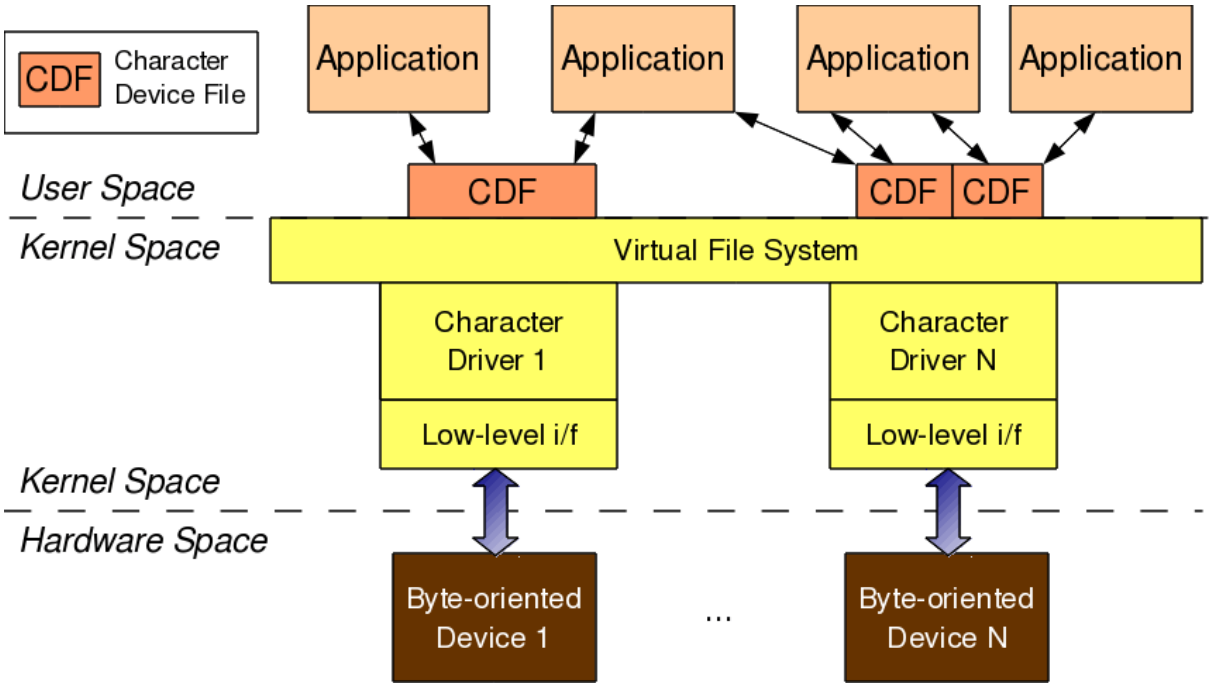
(2) Block device driver:
Examples : hard disks, USB cameras, Disk-On-Key.

Character device Driver :

We already know what drivers are, and why we need them. What is so special about character drivers? If we write drivers for byte-oriented operations (or, in C lingo, character-oriented operations), then we refer to them as character drivers. Since the majority of devices are byte-oriented, the majority of device drivers are character device drivers.

Take, for example, serial drivers, audio drivers, video drivers, camera drivers, and basic I/O drivers. In fact, all device drivers that are neither Storage nor network device drivers are some type of a character driver.

Character device driver Overview :



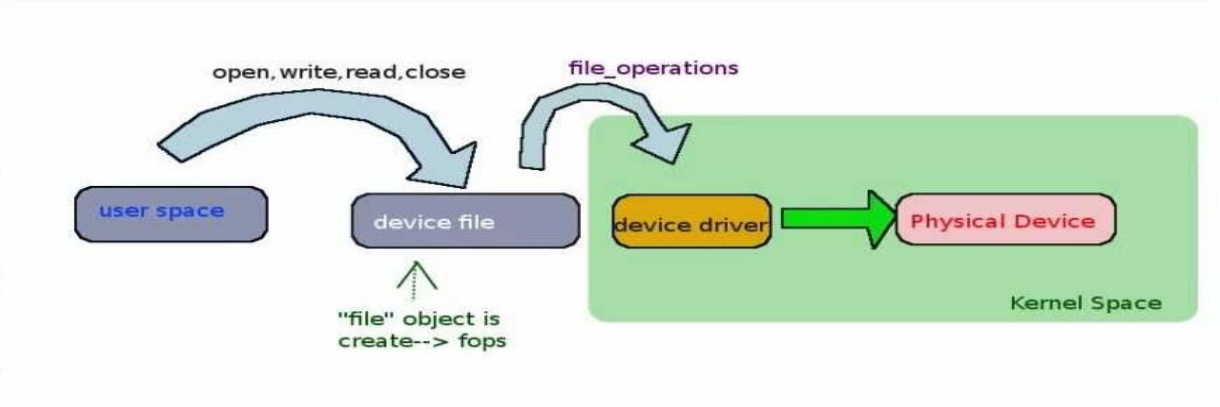
As shown in above Figure a, for any user-space application to operate on a byte-oriented device (in hardware space), it should use the corresponding character device driver (in kernel space). Character driver usage is done through the corresponding character device file(s), linked to it through the virtual file system (VFS). What this means is that an application does the usual file operations on the character device file. Those operations are translated to the corresponding functions in the linked character device driver by the VFS. Those functions then do the final low-level access to the actual device to achieve the desired results.

Note that though the application does the usual file operations, their outcome may not be the usual ones. Rather, they would be as driven by the corresponding functions in the device driver. For example, a write followed

by a read may not fetch what has just been written to the character device file, unlike for regular files. Remember that this is the usual expected behaviour for device files. Let's take an audio device file as an example. What we write into it is the audio data we want to play back, say through a speaker. However, the read would get us audio data that we are recording, say through a microphone. The recorded data need not be the played-back data.

In this complete connection from the application to the device, there are four major entities involved:

1. Application(user space)
2. Character device file
3. Character device driver
4. Character device(Physical device)



Major and Minor Number :

The connection between the application and the device file is based on the name of the device file. However, the connection between the device file and the device driver is based on the number of the device file, not the name. This allows a user-space application to have any name for the device file, and enables the kernel-space to have a trivial index-based linkage between the device file and the device driver. This device file number is more commonly referred to as the <major, minor> pair, or the major and minor numbers of the device file.

Earlier (till kernel 2.4), one major number was for one driver, and the minor number used to represent the sub-functionalities of the driver. With kernel 2.6, this distinction is no longer mandatory; there could be multiple drivers under the same major number, but obviously, with different minor number ranges.

However, this is more common with the non-reserved major numbers, and standard major numbers are typically preserved for single drivers. For example, 4 for serial Interfaces, 13 for mice, 14 for audio devices, and so on.

The following command would list the various character device files on yoursystem:

```
$ ls -l /dev/ | grep "^c"
```

List of System calls:

ChDrv_open(structinode *inode, struct file *filp)	The open system call performs the initialization of a device. In most cases, these operations refer to initializing the device and filling in specific data (if it is the first open call).where ,inode parameter is inode number with pointer and file will be point to the file to be opened. This call would return zero on success and -1 in case of failure.
intChDrv_release(structinode *inode, struct file *filp)	The release function is about releasing device-specific resources: unlocking specific data and closing the device if the last call is close. This call would return zero on success and -1 in case of failure.
ssize_tchar_read(struct file *, char *, size_t, loff_t *)	This system call transfer data between the device and the user-space: the read system call reads the data from the device and transfers it to the user-space. Parameter char is the character ,size_t is size of character ,loff_t is offset to determine to start from to the pointed file by file *.
access_ok(VERIFY_WRITE, buf, count))	To check the wheather the previous system call giving true output i.e 1.

ssize_tchar_write (struct file *, const char *, size_t, loff_t *)	Write reads the user-space data and writes it to the device.
copy_from_user(str,buf, count)	This system call performs the task of copying th data from user to the kernel space.
class_create(THIS_MODULE, CLASS_NAME);	This system call create virtual class to access device node in /dev file.
device_create(mycharClass, NULL, MKDEV(majorNumber, 0), NULL, DEVICE_NAME);	It will create device file in /devfile .

Examples:

```
/* ---------------------------------------------------------------------------------------
File Name:  chardriver.c
Description: This example program demonstrate simple opening and closed
                of kernel module with the open and release system calls.

Author: Embedded Linux Team, EC Department, CSPIT, CHARUSAT
Date & Time: 5th April, 2019 & 9:10 am
---------------------------------------------------------------------------------------*/
```

Chardriver1.c:

```
#include <linux/module.h>     /* Needed by all modules */
#include <linux/kernel.h>     /* Needed for KERN_INFO */
#include <linux/init.h>       /* Needed for the macros */

#include <linux/errno.h>      /* All the Linux/C error codes are listed here*/
#include <linux/fs.h>         /* Needed for file_operations structure */

#define DEVICE_NAME "CharDrv1"
#define MAJOR_NUM 55

#define SUCCESS 0;

staticintDevice_open = 0;
```

```
staticintChDrv_open(structinode *inode, struct file *filp)
{
        if(Device_open)
                return -EBUSY;

        printk(KERN_INFO "\n Kernel Module Opened; \n ChDrv_open: \n");

        Device_open++;

        try_module_get(THIS_MODULE);
        return SUCCESS
}

staticintChDrv_release(structinode *inode, struct file *filp)
{

        printk(KERN_INFO "\n Kernel Module Closed; \n ChDrv_release: \n");
        Device_open--;
       /* Decrement the usage count,
        * or else once you opened the file, you'll never get rid of the module.
        */
        module_put(THIS_MODULE);
        return SUCCESS;
}

structfile_operationsChDrv_fops =
{
       .owner                  = THIS_MODULE,
       .open           = ChDrv_open,
       .release        = ChDrv_release,
};

staticint __initChDrv_init(void)
{
        intret_val;
        ret_val = register_chrdev(MAJOR_NUM,DEVICE_NAME,&ChDrv_fops);
        if(ret_val< 0)
        {
                printk(KERN_ALERT "%s failed with %d\n",
                        "Sorry, registering the character device", ret_val);
                returnret_val;
        }
        printk(KERN_INFO "\n Inserted Kernel Module; \n Register_chrdev:
\n");
        return 0;
}
```

```
static void __exit ChDrv_exit(void)
{
        //int ret;
        //ret=unregister_chrdev(MAJOR_NUM,DEVICE_NAME);
        unregister_chrdev(MAJOR_NUM,DEVICE_NAME);
                printk(KERN_ALERT "Error: unregister_chrdev: \n");

        printk(KERN_INFO "\n Removed Kernel Module; \n
unregister_chrdev:\n");
}

module_init(ChDrv_init);
module_exit(ChDrv_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Ketan Patel");
MODULE_DESCRIPTION("A sample device driver");
```

EXECUTION STEPS:

```
el@nihal:~/Desktop$ cd CharDriver/
el@nihal:~/Desktop/CharDriver$ ls
2_CharDriver-0  2_char_driver-1  2_char_driver-2  2_char_driver-3
el@nihal:~/Desktop/CharDriver$ cd 2_CharDriver-0
el@nihal:~/Desktop/CharDriver/2_CharDriver-0$ ls
Driver
el@nihal:~/Desktop/CharDriver/2_CharDriver-0$ cd Driver/
el@nihal:~/Desktop/CharDriver/2_CharDriver-0/Driver$ ls
CharDriver1.c  Makefile
el@nihal:~/Desktop/CharDriver/2_CharDriver-0/Driver$ make
make -C /lib/modules/4.8.0-36-generic/build M=/home/el/Desktop/CharDriver/2_Char
Driver-0/Driver modules
make[1]: Entering directory '/usr/src/linux-headers-4.8.0-36-generic'
  CC [M]  /home/el/Desktop/CharDriver/2_CharDriver-0/Driver/CharDriver1.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/el/Desktop/CharDriver/2_CharDriver-0/Driver/CharDriver1.mod.o
  LD [M]  /home/el/Desktop/CharDriver/2_CharDriver-0/Driver/CharDriver1.ko
make[1]: Leaving directory '/usr/src/linux-headers-4.8.0-36-generic'
el@nihal:~/Desktop/CharDriver/2_CharDriver-0/Driver$ ls
CharDriver1.c   CharDriver1.mod.c  CharDriver1.o  modules.order
CharDriver1.ko  CharDriver1.mod.o  Makefile       Module.symvers
el@nihal:~/Desktop/CharDriver/2_CharDriver-0/Driver$
```

```
el@nihal:~/Desktop/CharDriver/2_CharDriver-0/Driver$ sudo mknod /dev/CharDrv1 c
55 0
sudo: unable to resolve host nihal
```

```
hal:~$ sudo su
 unable to resolve host nihal
nihal:/home/el# echo "hi" > /dev/driver1
 echo: write error: Invalid argument
nihal:/home/el# cat /dev/driver1
 /dev/driver1: Invalid argument
nihal:/home/el#
```

OUTPUT:

For output open another terminal and write dmesg command

```
[  298.939138]
               Inserted Kernel Module;
               Register_chrdev:
[ 1573.295767]
               Kernel Module Opened;
               ChDrv_open:
[ 1573.295848]
               Kernel Module Closed;
               ChDrv_release:
[ 1618.718790]
               Kernel Module Opened;
               ChDrv_open:
[ 1618.718851]
               Kernel Module Closed;
               ChDrv_release:
el@nihal:~$
```

EXAMPLE PROGRAM-2:

/* ----------
File Name: chardriver.c
Description: This example program demonstrate simple opening and closed of kernel module along with its also peforms reading and writing operation from kernel to user and vice-versa.

Author: Embedded Linux Team, EC Department, CSPIT, CHARUSAT
Date & Time: 5th April, 2019 & 9:10 am
----------*/

SOURCE CODE:

Char.c:

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/fs.h>
#include <linux/uaccess.h>
#include <linux/kdev_t.h>
```

```
MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("Character Device Example");
MODULE_AUTHOR("Ashutosh Bhatt");

// All the file operations which the driver will be able to handle
ssize_tchar_read(struct file *, char *, size_t, loff_t *);
ssize_tchar_write(struct file *, const char *, size_t, loff_t *);
intchar_open(structinode *, struct file *);
intchar_release(structinode *, struct file *);

charstr[100];

// Defining the structure containing all file operations for the driver
staticstructfile_operations fops = { .owner = THIS_MODULE, .read = char_read,
        .write = char_write, .open = char_open, .release = char_release,
};

// Assigning major number for the driver dynamically by the kernel
staticint major = 0;

// Name/Entry of the kernel module in /proc/devices
staticconst char dev_name[] = "my_test_char_driver";

// Registering the device
staticintregister_device(void) {
        int res;
        printk(KERN_NOTICE "CharDriver: register_device() initiated\n");

        res = register_chrdev(0, dev_name, &fops);
        if (res < 0) {
                printk(KERN_ALERT "CharDriver: device registration failed!\n");
                return res;
        }

        major = res; //Returning the major number assigned by the kernel
        printk(KERN_NOTICE "CharDriver: registered as a char device with major number #%d\n", major);

        return 0;
}

// Unregistering the device
voidunregister_device(void) {
        printk(KERN_NOTICE "CharDriver: unregister_device() initiated\n");
```

```
        if (major != 0)
                unregister_chrdev(major, dev_name);
}

// Reading operation of the driver
ssize_tchar_read(struct file *fp, char __user *buf, size_t count, loff_t *fpos)
{
        intlen=10;
        char* mystr;
printk(KERN_NOTICE "CharDriver: char_read() initiated\n");
        printk(KERN_NOTICE " %s \n",str);

        if (count <len)
        return -EINVAL;

        //if (*fpos != 0)
        if (*fpos>len)
        return -EINVAL;

        if (!access_ok(VERIFY_WRITE, buf, count))
        return -EFAULT;

        //*fpos += 10;
        mystr = str + *fpos;
        if (copy_to_user(buf, mystr, len))
        return -EINVAL;

        *fpos += len;
        returnlen;
}

// Writing operation of the driver
ssize_tchar_write(struct file *fp, const char __user *buf, size_t count, loff_t
*fpos)
{
        intlen = 100;
        printk(KERN_INFO "CharDriver: write called..\n");

        if (count >len)
        return -EINVAL;

        if (!access_ok(VERIFY_READ, buf, count))
        return -EFAULT;

        if (copy_from_user(str,buf, count))
        return -EINVAL;
```

```
//      str[count]='\0';

        printk(KERN_INFO "CharDriver: value written into device is: %s", str);

        return count;
}

// Open operation of the driver
intchar_open(structinode *node, struct file *fp) {
        printk(KERN_NOTICE "CharDriver: open is called\n");
        return 0;
}

// Close/Release operation of the driver
intchar_release(structinode *node, struct file *fp) {
        printk(KERN_NOTICE "CharDriver: release is called\n");
        return 0;
}

// Initialization function of kernel module i.e. this char driver
staticintchar_init(void) {
        printk(KERN_ALERT "CharDriver: Init\n");
        returnregister_device(); //Registering region with major no & minor no
}

// Exit function of kernel module i.e. this char driver
static void char_exit(void) {
        unregister_device();  //Unregistering the allocation
printk(KERN_ALERT "CharDriver: Bye\n");
}

module_init(char_init);
module_exit(char_exit);
```

Testchar.c:

```
/* ------------------------------------------------------------------------------------
File Name:  Testchar.c
Description: This example program demonstrating the simple code to test
                the character device driver.
          Author: Embedded Linux Team, EC Department, CSPIT, CHARUSAT
Date & Time: 5th April, 2019 & 9:10 am
.
------------------------------------------------------------------------------------*/

#include<stdio.h>
```

```
#include<stdlib.h>
#include<errno.h>
#include<fcntl.h>
#include<string.h>
#include<unistd.h>

#define BUFFER_LENGTH 256               // The buffer length
static char receive[BUFFER_LENGTH];     // The receive buffer

int main() {
        int ret, fd;
        charstringToSend[BUFFER_LENGTH];
        printf("Starting device test code example...\n");
        fd = open("/dev/ashu", O_RDWR);         // Open the device with
read/write access
        if (fd< 0) {
                perror("Failed to open the device...\n");
                returnerrno;
        }
        printf("Type in a short string to send to the kernel module:\n");
        scanf("%[^\n]c", stringToSend);                 // Read in a string (with
spaces)
        printf("Writing message to the device [%s].\n", stringToSend);
        ret = write(fd, stringToSend, strlen(stringToSend)); // Send the string
to the LKM
        if (ret < 0) {
                perror("Failed to write the message to the device.\n");
                returnerrno;
        }

        printf("Press ENTER key to read back from the device...\n");
        getchar();

        printf("Reading from the device...\n");
        unsignedintlen = 0;
        unsignedint offset = 0;
        len = pread(fd, receive, BUFFER_LENGTH, offset);
        printf("Offset : %ul\n", offset);

//    ret = read(fd, receive, BUFFER_LENGTH);         // Read the response
from the LKM
        if (len< 0) {
                perror("Failed to read the message from the device.\n");
                returnerrno;
        }
        printf("Read data of length: %ul\n", len);
```

```
        printf("The received message is: [%s]\n", receive);
    //offset = offset + len;

printf("Reading from the device...\n");
//    ret = read(fd, receive, BUFFER_LENGTH);        // Read the response
from the LKM
        len = pread(fd, receive, BUFFER_LENGTH, offset);
        printf("Offset : %ul", offset);

        if (len< 0) {
                perror("Failed to read the message from the device.\n");
                returnerrno;
        }
        printf("Read data of length: %ul\n", len);
        printf("The received message is: [%s]\n", receive);
        printf("End of the program\n");
        return 0;
}
```

EXECUTION STEPS:

```
el@nihal:~/Desktop$ cd CharDriver/
el@nihal:~/Desktop/CharDriver$ ls
2_CharDriver-0  2_char_driver-1  2_char_driver-2  2_char_driver-3
el@nihal:~/Desktop/CharDriver$ cd 2_char_driver-1
el@nihal:~/Desktop/CharDriver/2_char_driver-1$ ls
char.c   char.mod.c  char.o    modules.order   README    testchar.c
char.ko  char.mod.o  Makefile  Module.symvers  testchar  testchar.o
el@nihal:~/Desktop/CharDriver/2_char_driver-1$ sudo insmod char.ko
sudo: unable to resolve host nihal
[sudo] password for el:
el@nihal:~/Desktop/CharDriver/2_char_driver-1$ lsmod
Module                  Size  Used by
char                   16384  0
```

```
el@nihal:~/Desktop/CharDriver/2_char_driver-1$ sudo mknod /dev/ashu c 244 0
sudo: unable to resolve host nihal
el@nihal:~/Desktop/CharDriver/2_char_driver-1$ sudo chmod 777 /dev/ashu
sudo: unable to resolve host nihal
el@nihal:~/Desktop/CharDriver/2_char_driver-1$ echo "Hello Driver" > /dev/ashu
el@nihal:~/Desktop/CharDriver/2_char_driver-1$ cat /dev/ashu
Hello Driver
el@nihal:~/Desktop/CharDriver/2_char_driver-1$ ./testchar
```

OUTPUT:

```
Starting device test code example...
Type in a short string to send to the kernel module:
hey there
Writing message to the device [hey there].
Press ENTER key to read back from the device...
Reading from the device...
Offset : 0l
Read data of length: 10l
The received message is: [hey therev]
Reading from the device...
Offset : 0lRead data of length: 10l
The received message is: [hey therev]
End of the program
```

EXAMPLE PROGRAM -3

/* ----------
File Name: mychar.c
Description: This example program demonstrate simple opening and closed of kernel module, reading and writing operation from kernel to user and vice-versa along with assigning of major number will be performed automatically in this program .

Author: Embedded Linux Team, EC Department, CSPIT, CHARUSAT
Date & Time: 5th April, 2019 & 9:10 am
.
----------*/

SOURCE CODE:

Mychar.c:

```
#include <linux/init.h>             // Macros used to mark up functions e.g. __init __exit
#include <linux/module.h>           // Core header for loading LKMs into the kernel
#include <linux/device.h>           // Header to support the kernel Driver Model
#include <linux/kernel.h>           // Contains types, macros, functions for the kernel
#include <linux/fs.h>               // Header for the Linux file system support
#include <linux/uaccess.h>          // Required for the copy to user function
#define  DEVICE_NAME "mychar"       ///< The device will appear at /dev/mychar using this value
#define  CLASS_NAME  "my"           ///< The device class -- this is a character device driver
```

```
MODULE_LICENSE("GPL");              ///< The license type -- this affects available functionality
MODULE_AUTHOR("Ashutosh Bhatt");    ///< The author -- visible when you use modinfo
MODULE_DESCRIPTION("A simple Linux char driver for the Linux System");
///< The description -- see modinfo
MODULE_VERSION("0.1");              ///< A version number to inform users

staticintmajorNumber;                  ///< Stores the device number -- determined automatically
static char   message[256] = {0};      ///< Memory for the string that is passed from userspace
static short  size_of_message;         ///< Used to remember the size of the string stored
staticintnumberOpens = 0;              ///< Counts the number of times the device is opened
staticstruct class*  mycharClass  = NULL; ///< The device-driver class struct pointer
staticstruct device* mycharDevice = NULL; ///< The device-driver device struct pointer

// The prototype functions for the character driver -- must come before the struct definition
staticintdev_open(structinode *, struct file *);
staticintdev_release(structinode *, struct file *);
staticssize_tdev_read(struct file *, char *, size_t, loff_t *);
staticssize_tdev_write(struct file *, const char *, size_t, loff_t *);

/** @brief Devices are represented as file structure in the kernel. The file_operations structure from
*  /linux/fs.h lists the callback functions that you wish to associated with your file operations
*  using a C99 syntax structure. char devices usually implement open, read, write and release calls
 */
staticstructfile_operations fops =
{
   .open = dev_open,
   .read = dev_read,
   .write = dev_write,
   .release = dev_release,
};

/** @brief The LKM initialization function
*  The static keyword restricts the visibility of the function to within this C file. The __init
```

```
*  macro means that for a built-in driver (not a LKM) the function is only used at initialization
*  time and that it can be discarded and its memory freed up after that point.
*  @return returns 0 if successful
 */
staticint __initmychar_init(void){
printk(KERN_INFO "myChar: Initializing the myChar LKM\n");

   // Try to dynamically allocate a major number for the device -- more difficult but worth it
majorNumber = register_chrdev(0, DEVICE_NAME, &fops);
if (majorNumber<0){
printk(KERN_ALERT "myChar failed to register a major number\n");
returnmajorNumber;
   }
printk(KERN_INFO "myChar: registered correctly with major number %d\n", majorNumber);

   // Register the device class
mycharClass = class_create(THIS_MODULE, CLASS_NAME);
if (IS_ERR(mycharClass)){                // Check for error and clean up if there is
unregister_chrdev(majorNumber, DEVICE_NAME);
printk(KERN_ALERT "Failed to register device class\n");
return PTR_ERR(mycharClass);          // Correct way to return an error on a pointer
   }
printk(KERN_INFO "myChar: device class registered correctly\n");

   // Register the device driver
mycharDevice = device_create(mycharClass, NULL, MKDEV(majorNumber, 0), NULL, DEVICE_NAME);
if (IS_ERR(mycharDevice)){               // Clean up if there is an error
class_destroy(mycharClass);           // Repeated code but the alternative is goto statements
unregister_chrdev(majorNumber, DEVICE_NAME);
printk(KERN_ALERT "Failed to create the device\n");
return PTR_ERR(mycharDevice);
   }
printk(KERN_INFO "myChar: device class created correctly\n"); // Made it! device was initialized
return 0;
}

/** @brief The LKM cleanup function
```

```
*  Similar to the initialization function, it is static. The __exit macro notifies that if this
*  code is used for a built-in driver (not a LKM) that this function is not required.
 */
static void __exit mychar_exit(void){
device_destroy(mycharClass, MKDEV(majorNumber, 0));      // remove the device
class_unregister(mycharClass);                           // unregister the device class
class_destroy(mycharClass);                              // remove the device class
unregister_chrdev(majorNumber, DEVICE_NAME);             // unregister the major number
printk(KERN_INFO "myChar: Goodbye from the LKM!\n");
}

/** @brief The device open function that is called each time the device is opened
*  This will only increment the numberOpens counter in this case.
*  @paraminodep A pointer to an inode object (defined in linux/fs.h)
*  @paramfilep A pointer to a file object (defined in linux/fs.h)
 */
staticintdev_open(structinode *inodep, struct file *filep){
numberOpens++;
printk(KERN_INFO "myChar: Device has been opened %d time(s)\n", numberOpens);
return 0;
}

/** @brief This function is called whenever device is being read from user space i.e. data is
*  being sent from the device to the user. In this case is uses the copy_to_user() function to
*  send the buffer string to the user and captures any errors.
*  @paramfilep A pointer to a file object (defined in linux/fs.h)
*  @param buffer The pointer to the buffer to which this function writes the data
*  @paramlen The length of the b
*  @param offset The offset if required
 */
staticssize_tdev_read(struct file *filep, char *buffer, size_tlen, loff_t *offset){
interror_count = 0;
  // copy_to_user has the format ( * to, *from, size) and returns 0 on success
error_count = copy_to_user(buffer, message, size_of_message);
```

```
if (error_count==0){            // if true then have success
printk(KERN_INFO "myChar: Sent %d characters to the user\n", size_of_message);
return (size_of_message=0);  // clear the position to the start and return 0
   }
else {
printk(KERN_INFO "myChar: Failed to send %d characters to the user\n", error_count);
return -EFAULT;              // Failed -- return a bad address message (i.e. -14)
   }
}

/** @brief This function is called whenever the device is being written to from user space i.e.
*   data is sent to the device from the user. The data is copied to the message[] array in this
*  LKM using the sprintf() function along with the length of the string.
*  @paramfilep A pointer to a file object
*  @param buffer The buffer to that contains the string to write to the device
*   @paramlen The length of the array of data that is being passed in the const char buffer
*  @param offset The offset if required
 */
staticssize_tdev_write(struct file *filep, const char *buffer, size_tlen, loff_t *offset){
sprintf(message, "%s(%zu letters)", buffer, len);     // appending received string with its length
size_of_message = strlen(message);                 // store the length of the stored message
printk(KERN_INFO "myChar: Received %zu characters from the user\n", len);
returnlen;
}

/** @brief The device release function that is called whenever the device is closed/released by
*  theuserspace program
*  @paraminodep A pointer to an inode object (defined in linux/fs.h)
*  @paramfilep A pointer to a file object (defined in linux/fs.h)
 */
staticintdev_release(structinode *inodep, struct file *filep){
printk(KERN_INFO "myChar: Device successfully closed\n");
return 0;
}
```

```
/** @brief A module must use the module_init() module_exit() macros from linux/init.h, which
*  identify the initialization function at insertion time and the cleanup function (as
*  listed above)
 */
module_init(mychar_init);
module_exit(mychar_exit);
```

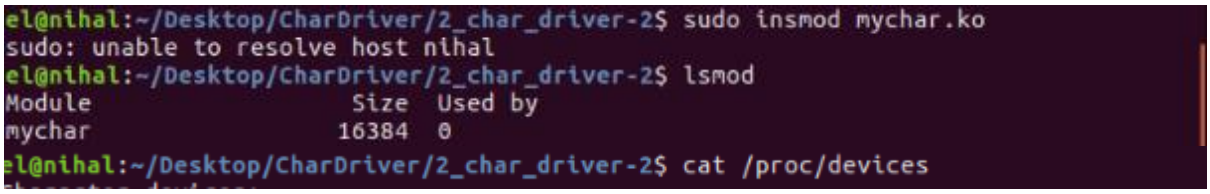
Testmychar.c:

```
/* ----------------------------------------------------------------------------------------
File Name:  Testchar.c
Description: This example program demonstrating the simple code to test
                the character device driver.
          Author: Embedded Linux Team, EC Department, CSPIT, CHARUSAT
Date & Time: 5th April, 2019 & 9:10 am
----------------------------------------------------------------------------------------*/


#include<stdio.h>
#include<stdlib.h>
#include<errno.h>
#include<fcntl.h>
#include<string.h>
#include<unistd.h>

#define BUFFER_LENGTH 256               ///< The buffer length (crude but fine)
static char receive[BUFFER_LENGTH];     ///< The receive buffer from the LKM

int main(){
int ret, fd;
charstringToSend[BUFFER_LENGTH];
printf("Starting device test code example...\n");
fd = open("/dev/mychar", O_RDWR);               // Open the device with read/write access
if (fd< 0){
perror("Failed to open the device...");
returnerrno;
  }
printf("Type in a short string to send to the kernel module:\n");
scanf("%[^\n]%*c", stringToSend);                   // Read in a string (with spaces)
printf("Writing message to the device [%s].\n", stringToSend);
```

```
ret = write(fd, stringToSend, strlen(stringToSend)); // Send the string to the
LKM
if (ret < 0){
perror("Failed to write the message to the device.");
returnerrno;
   }

printf("Press ENTER to read back from the device...\n");
getchar();

printf("Reading from the device...\n");
ret = read(fd, receive, BUFFER_LENGTH);          // Read the response from
the LKM
if (ret < 0){
perror("Failed to read the message from the device.");
returnerrno;
   }
printf("The received message is: [%s]\n", receive);
printf("End of the program\n");
return 0;
}
```

EXECUTION STEPS:

Execute make command first then follow the steps below.



NOTE:

The steps mentioned below are to executed on two different terminals.

OUTPUT:

```
el@nihal:~/Desktop/CharDriver/2_CharDriver-0/Driver$ sudo insmod CharDriver1.ko
sudo: unable to resolve host nihal
[sudo] password for el:
el@nihal:~/Desktop/CharDriver/2_CharDriver-0/Driver$ cat /proc/devices
Character devices:
  1 mem
  4 /dev/vc/0
  4 tty
```

Outputs on both terminals:

```
root@nihal:/home/el/Desktop/CharDriver/2_char_driver-2# ./testmychar
Starting device test code example...
Type in a short string to send to the kernel module:
hi there
Writing message to the device [hi there].
Press ENTER to read back from the device...

Reading from the device...
The received message is: [hi there(8 letters)]
End of the program
root@nihal:/home/el/Desktop/CharDriver/2_char_driver-2#
```

```
root@nihal:/home/el/Desktop/CharDriver/2_char_driver-2# ./testmychar
Starting device test code example...
Type in a short string to send to the kernel module:
hello
Writing message to the device [hello].
Press ENTER to read back from the device...

Reading from the device...
The received message is: [hello(5 letters)]
End of the program
root@nihal:/home/el/Desktop/CharDriver/2_char_driver-2# .
```

EXAMPLE PROGRAM 4:

```
/* ---------------------------------------------------------------------------------------
File Name:  mycharmutexc
Description:  This example program demonstrate simple opening and closed
              of kernel module, reading and writing operation from kernel
              to user and vice-versa along with assigning of major number
              will be performed automatically in this program ,however its
              also provide function of lock the execution process  for
              specified time.

        Author: Embedded Linux Team, EC Department, CSPIT, CHARUSAT
Date & Time: 5th April, 2019 & 9:10 am
---------------------------------------------------------------------------------------*/
```

SOURCE CODE:

Mycharmutex.c

```
#include <linux/init.h>             // Macros used to mark up functions e.g.
__init __exit
#include <linux/module.h>           // Core header for loading LKMs into the
kernel
#include <linux/device.h>           // Header to support the kernel Driver
Model
#include <linux/kernel.h>          // Contains types, macros, functions for the
kernel
#include <linux/fs.h>              // Header for the Linux file system support
#include <linux/uaccess.h>          // Required for the copy to user function
#include <linux/mutex.h>           // Required for the mutex functionality
#define  DEVICE_NAME "mychar"       ///< The device will appear at
/dev/mychar using this value
#define  CLASS_NAME  "my"        ///< The device class -- this is a character
device driver

MODULE_LICENSE("GPL");              ///< The license type -- this affects
available functionality
MODULE_AUTHOR("Ashutosh Bhatt");    ///< The author -- visible when
you use modinfo
MODULE_DESCRIPTION("A simple Linux char driver for the Linux System");
///< The description -- see modinfo
MODULE_VERSION("0.2");              ///< A version number to inform users

staticintmajorNumber;                  ///< Store the device number --
determined automatically
```

```
static char    message[256] = {0};           ///< Memory for the string that is passed from userspace
static short   size_of_message;              ///< Used to remember the size of the string stored
staticintnumberOpens = 0;                ///< Counts the number of times the device is opened
staticstruct class*  mycharClass  = NULL; ///< The device-driver class struct pointer
staticstruct device* mycharDevice = NULL; ///< The device-driver device struct pointer

static DEFINE_MUTEX(mychar_mutex);       ///< Macro to declare a new mutex

// The prototype functions for the character driver -- must come before the struct definition
staticintdev_open(structinode *, struct file *);
staticintdev_release(structinode *, struct file *);
staticssize_tdev_read(struct file *, char *, size_t, loff_t *);
staticssize_tdev_write(struct file *, const char *, size_t, loff_t *);

/** @brief Devices are represented as file structure in the kernel. The file_operations structure from
*  /linux/fs.h lists the callback functions that you wish to associated with your file operations
*  using a C99 syntax structure. char devices usually implement open, read, write and release calls
 */
staticstructfile_operations fops =
{
   .open = dev_open,
   .read = dev_read,
   .write = dev_write,
   .release = dev_release,
};

/** @brief The LKM initialization function
*  The static keyword restricts the visibility of the function to within this C file. The __init
*  macro means that for a built-in driver (not a LKM) the function is only used at initialization
*  time and that it can be discarded and its memory freed up after that point.
*  @return returns 0 if successful
 */
staticint __initmychar_init(void){
```

```
printk(KERN_INFO "myChar: Initializing the myChar LKM\n");

   // Try to dynamically allocate a major number for the device -- more difficult but worth it
majorNumber = register_chrdev(0, DEVICE_NAME, &fops);
if (majorNumber<0){
printk(KERN_ALERT "myChar failed to register a major number\n");
returnmajorNumber;
   }
printk(KERN_INFO "myChar: registered correctly with major number %d\n", majorNumber);

   // Register the device class
mycharClass = class_create(THIS_MODULE, CLASS_NAME);
if (IS_ERR(mycharClass)){          // Check for error and clean up if there is
unregister_chrdev(majorNumber, DEVICE_NAME);
printk(KERN_ALERT "Failed to register device class\n");
return PTR_ERR(mycharClass);       // Correct way to return an error on a pointer
   }
printk(KERN_INFO "myChar: device class registered correctly\n");

   // Register the device driver
mycharDevice = device_create(mycharClass, NULL, MKDEV(majorNumber, 0), NULL, DEVICE_NAME);
if (IS_ERR(mycharDevice)){         // Clean up if there is an error
class_destroy(mycharClass);       // Repeated code but the alternative is goto statements
unregister_chrdev(majorNumber, DEVICE_NAME);
printk(KERN_ALERT "Failed to create the device\n");
return PTR_ERR(mycharDevice);
   }
printk(KERN_INFO "myChar: device class created correctly\n"); // Made it! device was initialized
mutex_init(&mychar_mutex);         // Initialize the mutex dynamically
return 0;
}

/** @brief The LKM cleanup function
*  Similar to the initialization function, it is static. The __exit macro notifies that if this
*  code is used for a built-in driver (not a LKM) that this function is not required.
 */
static void __exit mychar_exit(void){
```

```
mutex_destroy(&mychar_mutex);                    // destroy the dynamically-allocated mutex
device_destroy(mycharClass, MKDEV(majorNumber, 0)); // remove the device
class_unregister(mycharClass);                   // unregister the device class
class_destroy(mycharClass);                      // remove the device class
unregister_chrdev(majorNumber, DEVICE_NAME);     // unregister the major number
printk(KERN_INFO "myChar: Goodbye from the LKM!\n");
}

/** @brief The device open function that is called each time the device is opened
*  This will only increment the numberOpens counter in this case.
*  @paraminodep A pointer to an inode object (defined in linux/fs.h)
*  @paramfilep A pointer to a file object (defined in linux/fs.h)
 */
staticintdev_open(structinode *inodep, struct file *filep){

if(!mutex_trylock(&mychar_mutex)){                // Try to acquire the mutex (returns 0 on fail)
        printk(KERN_ALERT "myChar: Device in use by another process");
        return -EBUSY;
   }
numberOpens++;
printk(KERN_INFO "myChar: Device has been opened %d time(s)\n", numberOpens);
return 0;
}

/** @brief This function is called whenever device is being read from user space i.e. data is
*    being sent from the device to the user. In this case is uses the copy_to_user() function to
*  send the buffer string to the user and captures any errors.
*  @paramfilep A pointer to a file object (defined in linux/fs.h)
*  @param buffer The pointer to the buffer to which this function writes the data
*  @paramlen The length of the b
*  @param offset The offset if required
 */
staticssize_tdev_read(struct file *filep, char *buffer, size_tlen, loff_t *offset){
interror_count = 0;
   // copy_to_user has the format ( * to, *from, size) and returns 0 on success
error_count = copy_to_user(buffer, message, size_of_message);
```

```
if (error_count==0){          // success!
printk(KERN_INFO "myChar: Sent %d characters to the user\n", size_of_message);
return (size_of_message=0); // clear the position to the start and return 0
  }
else {
printk(KERN_INFO "myChar: Failed to send %d characters to the user\n", error_count);
return -EFAULT;      // Failed -- return a bad address message (i.e. -14)
  }
}

/** @brief This function is called whenever the device is being written to from user space i.e.
*  data is sent to the device from the user. The data is copied to the message[] array in this
*  LKM using message[x] = buffer[x]
*  @paramfilep A pointer to a file object
*  @param buffer The buffer to that contains the string to write to the device
*  @paramlen The length of the array of data that is being passed in the const char buffer
*  @param offset The offset if required
 */
staticssize_tdev_write(struct file *filep, const char *buffer, size_tlen, loff_t *offset){
sprintf(message, "%s(%zu letters)", buffer, len);     // appending received string with its length
size_of_message = strlen(message);                    // store the length of the stored message
printk(KERN_INFO "myChar: Received %zu characters from the user\n", len);
returnlen;
}

/** @brief The device release function that is called whenever the device is closed/released by
*  theuserspace program
*  @paraminodep A pointer to an inode object (defined in linux/fs.h)
*  @paramfilep A pointer to a file object (defined in linux/fs.h)
 */
staticintdev_release(structinode *inodep, struct file *filep){
mutex_unlock(&mychar_mutex);                          // release the mutex (i.e., lock goes up)
printk(KERN_INFO "myChar: Device successfully closed\n");
return 0;
```

```
}

/** @brief A module must use the module_init() module_exit() macros from linux/init.h, which
*  identify the initialization function at insertion time and the cleanup function (as
*  listed above)
 */
module_init(mychar_init);
module_exit(mychar_exit);
```

Testmycharmutex.c:

```
/* ----------------------------------------------------------------------------------------
File Name:  Testcharmutex.c
Description: This example program demonstrating the simple code to test
                the character device driver.
          Author: Embedded Linux Team, EC Department, CSPIT, CHARUSAT
Date & Time: 5th April, 2019 & 9:10 am
----------------------------------------------------------------------------------------*/

#include<stdio.h>
#include<stdlib.h>
#include<errno.h>
#include<fcntl.h>
#include<string.h>
#include<unistd.h>

#define BUFFER_LENGTH 256               ///< The buffer length (crude but fine)
static char receive[BUFFER_LENGTH];     ///< The receive buffer from the LKM

int main(){
int ret, fd;
charstringToSend[BUFFER_LENGTH];
printf("Starting device test code example...\n");
fd = open("/dev/mychar", O_RDWR);             // Open the device with read/write access
if (fd< 0){
perror("Failed to open the device...");
returnerrno;
  }
printf("Type in a short string to send to the kernel module:\n");
scanf("%[^\n]%*c", stringToSend);             // Read in a string (with spaces)
```

```
printf("Writing message to the device [%s].\n", stringToSend);
ret = write(fd, stringToSend, strlen(stringToSend)); // Send the string to the LKM
if (ret < 0){
perror("Failed to write the message to the device.");
returnerrno;
   }

printf("Press ENTER to read back from the device...");
getchar();

printf("Reading from the device...\n");
ret = read(fd, receive, BUFFER_LENGTH);          // Read the response from the LKM
if (ret < 0){
perror("Failed to read the message from the device.");
returnerrno;
   }
printf("The received message is: [%s]\n", receive);
printf("End of the program\n");
return 0;
}
```

EXECUION STEPS:

```
el@nihal:~/Desktop/CharDriver/2_char_driver-3$ sudo insmod mycharmutex.ko
sudo: unable to resolve host nihal
el@nihal:~/Desktop/CharDriver/2_char_driver-3$ cat /proc/devices
```

Execute make command first and then execute the following commands

The steps mentioned below are to executed on two differentterminals.

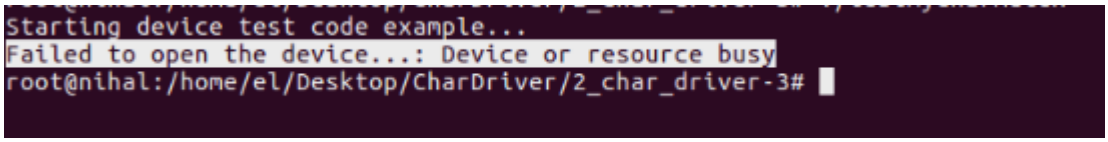
```
el@nihal:~/Desktop/CharDriver/2_char_driver-3$ sudo su
sudo: unable to resolve host nihal
root@nihal:/home/el/Desktop/CharDriver/2_char_driver-3# ./testmycharmutex
Starting device test code example...
```

OUTPUT:

Output on terminal 1

```
Starting device test code example...
Type in a short string to send to the kernel module:
hello
Writing message to the device [hello].
Press ENTER to read back from the device...
```

Output on terminal 2



Experiment No.15

Aim: Implementation of USB Device Driver.

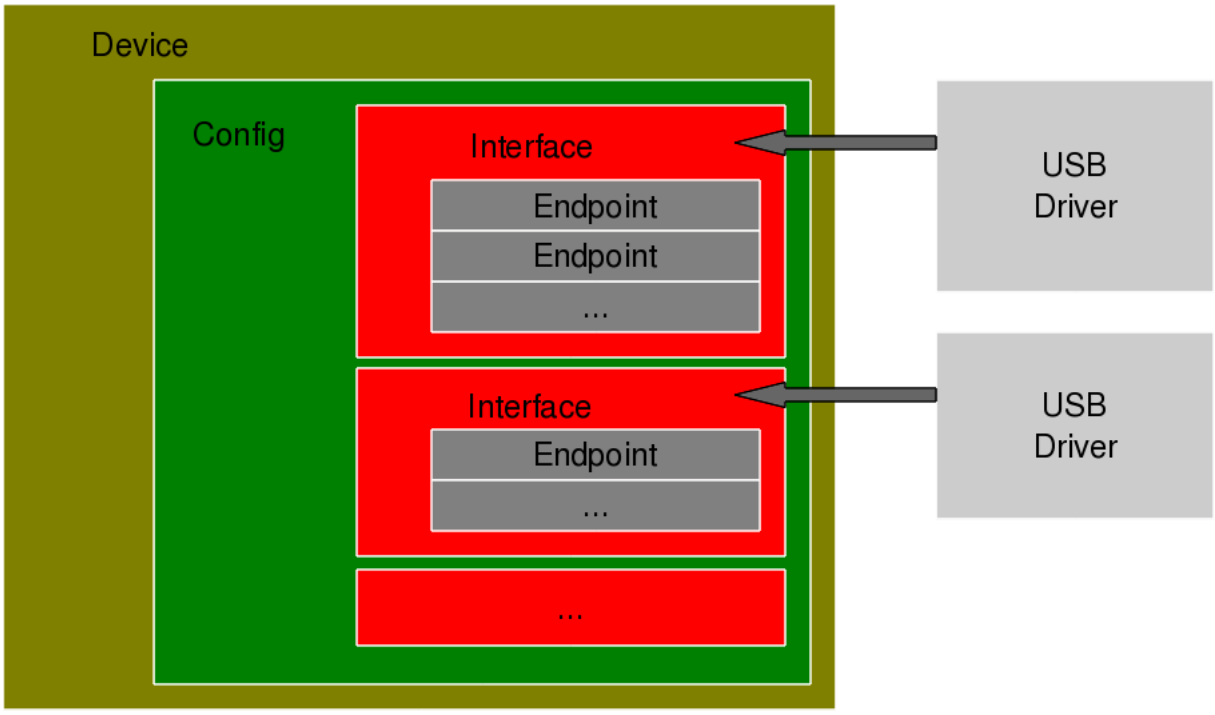
Theory:

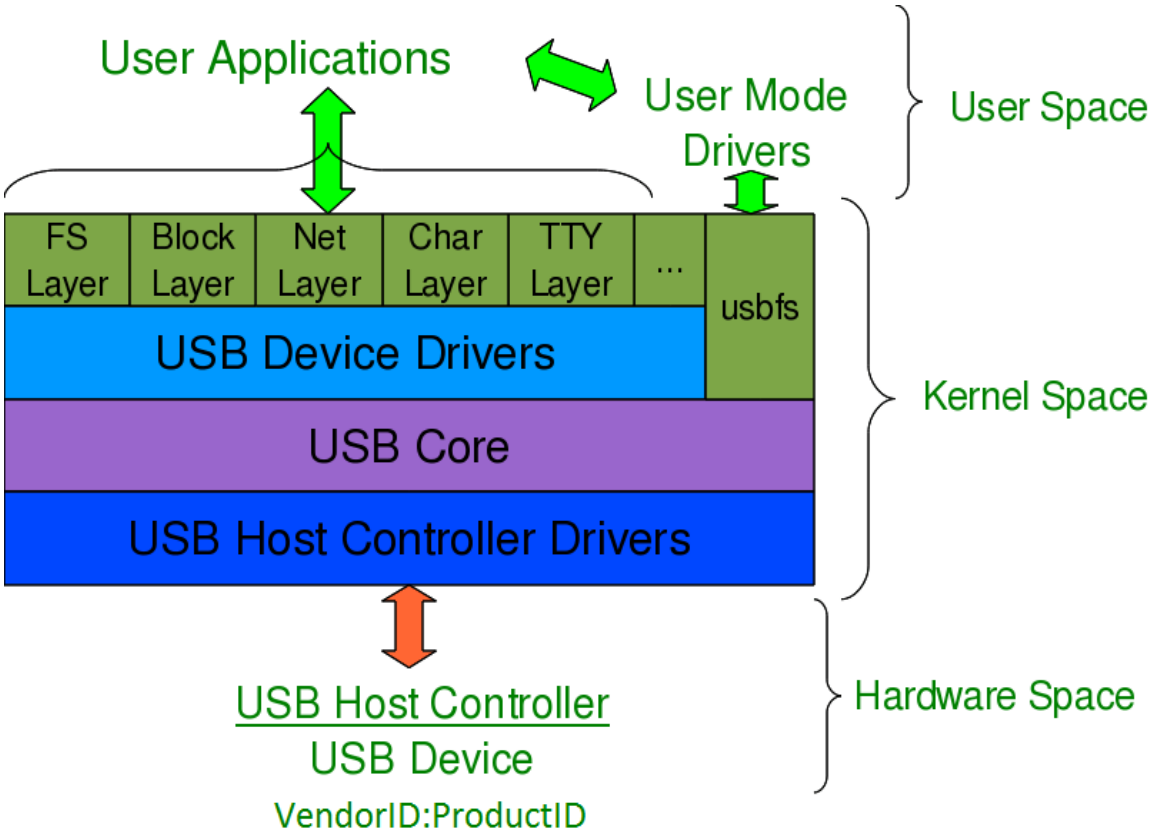
- Every valid USB device has it's vendor ID and product ID and that's how system is able to recognise the device.
- First of all, the host controller takes all the binary information and makes it available and understandable by an operating system independence system called the USB core.
- Further, the USB core gives that information over to linux kernel.
- At that time, kernel checks whether a particular device driver is available to handle a particular device.
- If one is available, the kernel makes that device available to the user space.
- If not available, the kernel still recognises the device however the user space or applications can never know of device's existence.
- The device is configured in such a way that every device has it's profile which is configuration and within configuration there are no. of interfaces.
- Interface describes the functionality of device. So, the device could be a printer or a scanner. So for each function there is an interface.
- Each interface for every device has n no. of endpoints. It basically, describes input and output.
- Endpoint 0 is used to initialise the device and make it ready for usage.

Depending on the type and attributes of information to be transferred, a USB device may have one or more endpoints, each belonging to one of the following four categories:

- Control – For transferring control information. Examples include resetting the device, querying information about the device, etc. All USB devices always have the default control endpoint point zero.
- Interrupt – For small and fast data transfer, typically of up to 8 bytes. Examples include data transfer for serial ports, human interface devices (HIDs) like keyboard, mouse, etc.
- Bulk – For big but comparatively slow data transfer. A typical example is data transfers for mass storage devices.
- Isochronous – For big data transfer with bandwidth guarantee, though data integrity may not be guaranteed. Typical practical usage examples include transfers of time-sensitive data like of audio, video, etc.

-Technically, an endpoint is identified using a 8-bit number, most significant bit (MSB) of which indicates the direction – 0 meaning out, and 1 meaning in. Control endpoints are bi-directional and the MSB is ignored.





Example Program:

```
/* ------------------------------------------------------------------------
                ------------
File Name: pen_register.c
Description: This example program demonstrate USB Device
             Driver for registration and deregistration of a
             pendrive.
Author: Embedded Linux Team, EC Department, CSPIT,
CHARUSAT
Date & Time: 2nd April, 2019 & 05:30 pm.
------------------------------------------------------------------------*/
#include <linux/module.h>

#include <linux/kernel.h>

#include <linux/usb.h>
```

//Probe Function

/* called if and only if no other driver takes claim of managing the device */

```
Static int pen_probe(structusb_interface *interface, const struct
usb_device_id *id)

{

printk(KERN_INFO "Pen drive (%04X:%04X) plugged\n", id-
>idVendor,

id->idProduct);

return 0;

}
```

//usb_disconnect

```
static void pen_disconnect(structusb_interface *interface)

{

printk(KERN_INFO "Pen drive removed\n");

}
```

//usb_device_id

```
Static struct usb_device_idpen_table[] =

{

 { USB_DEVICE(0x054c, 0x02a5) },/*Vendor ID and Product ID
obtained using “lsusb” at command Line*/

    {} /* Terminating entry */

};

MODULE_DEVICE_TABLE (usb, pen_table);
```

//usb_driver

```
Static struct usb_driverpen_driver =
```

```
{
    .name = "pen_driver",
    .id_table = pen_table,//To get Device id
    .probe = pen_probe,
    .disconnect = pen_disconnect,
};
```

//usb_registration

```
Static int __initpen_init(void)
{
return usb_register(&pen_driver);/*To register device into
                            kernel*/
}
```

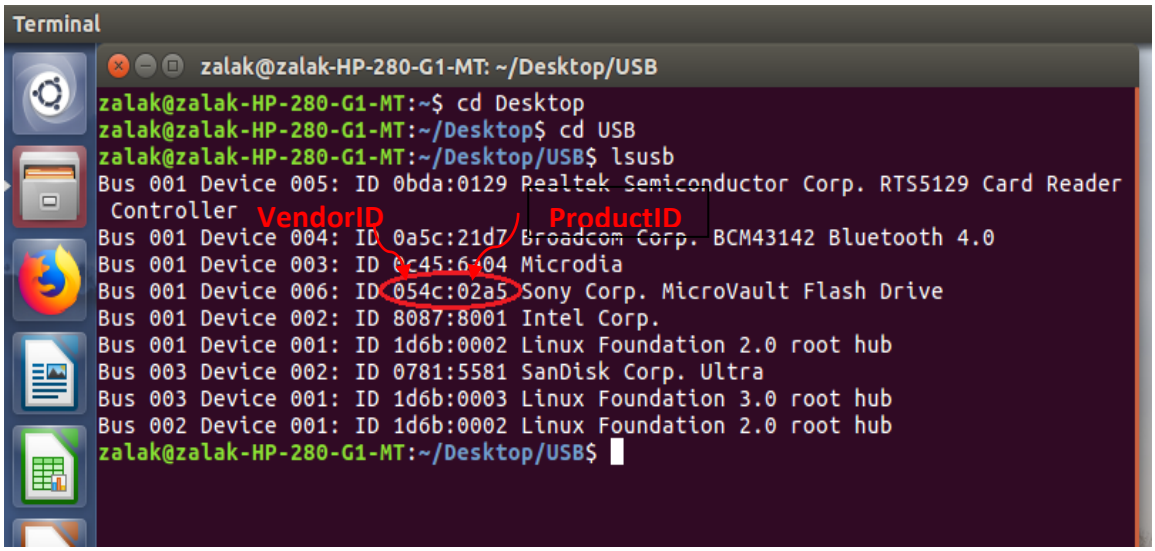
//usb_deregistration

```
static void __exit pen_exit(void)
{
usb_deregister(&pen_driver);//To deregister device from kernel
}

module_init(pen_init);
module_exit(pen_exit);

MODULE_LICENSE("GPL");
MODULE DESCRIPTION("USB Pen Registration Driver");
```

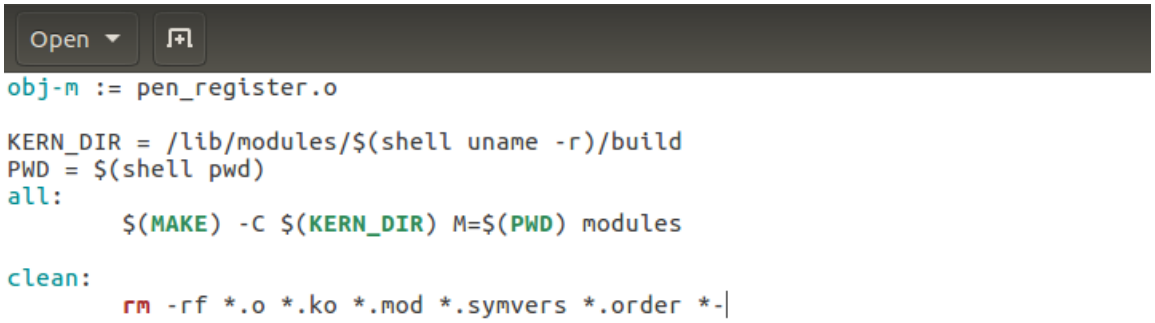
Execution Steps:

Step 1:

- Execute **lsusb** command to see vendorID and productID of the device connected.
- Create a folder named “usb”.
- Create “pen_register.c” file.



Step 2:



- Create Makefile.

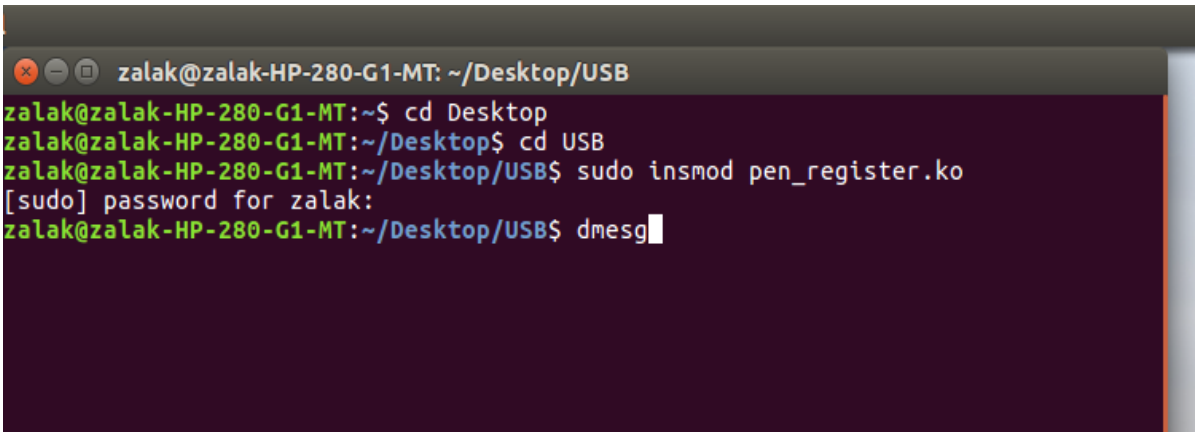
Step 3:

- Execute Makefile using **make** command.
- "pen_register.ko" file will be created.

```
zalak@zalak-HP-280-G1-MT:~/Desktop/USB$ make
make -C /lib/modules/4.8.0-36-generic/build M=/home/zalak/Desktop/USB modules
make[1]: Entering directory '/usr/src/linux-headers-4.8.0-36-generic'
  CC [M]  /home/zalak/Desktop/USB/pen_register.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/zalak/Desktop/USB/pen_register.mod.o
  LD [M]  /home/zalak/Desktop/USB/pen_register.ko
make[1]: Leaving directory '/usr/src/linux-headers-4.8.0-36-generic'
zalak@zalak-HP-280-G1-MT:~/Desktop/USB$ ls
Makefile       Module.symvers  pen_register.ko     pen_register.mod.o
modules.order  pen_register.c  pen_register.mod.c  pen_register.o
zalak@zalak-HP-280-G1-MT:~/Desktop/USB$
```

Step 4:

- Insert that module into kernel using **sudo insmod ./pen_register.ko.**
- Type **dmesg** to see messages in kernel message buffer**.**



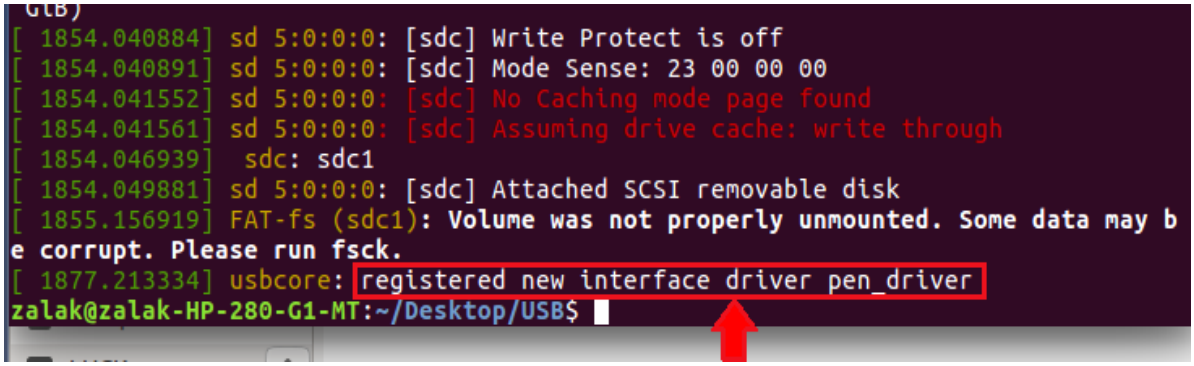
Step 5:

- Remove that module from kernel using **sudormmod ./pen_register.ko.**

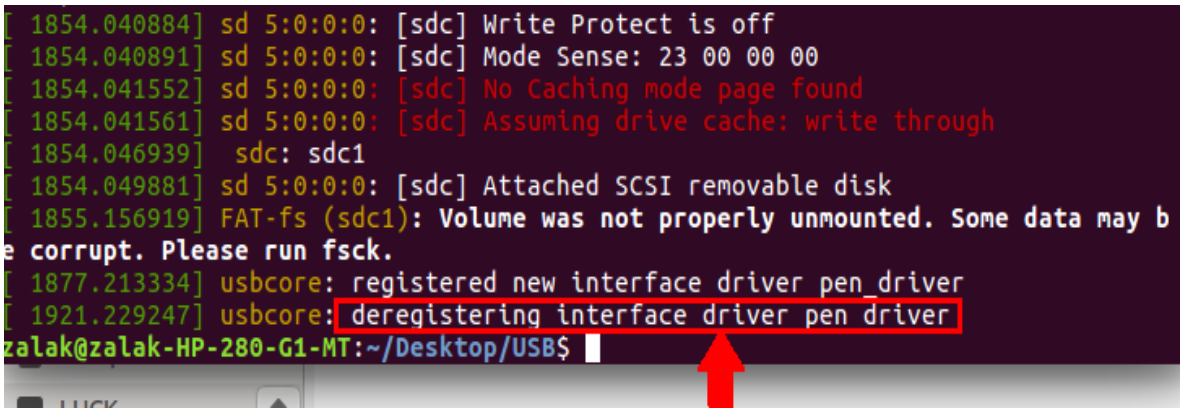
```
zalak@zalak-HP-280-G1-MT: ~/Desktop/USB
zalak@zalak-HP-280-G1-MT:~/Desktop/USB$ sudo rmmod pen_register.ko
zalak@zalak-HP-280-G1-MT:~/Desktop/USB$
```

Output:

After Inserting Module:



After Removing Module:



Experiment No.16

Aim: Toolchain and Cross compilation

Theory:

Toolchain:-

A toolchain is a set of distinct software development tools that are linked (or chained) together by specific stages such as GCC, binutils and glibc (a portion of the GNU Toolchain). Optionally, a toolchain may contain other tools such as a debugger or a compiler for a specific programming language, such as C++. Quite often, the toolchain used for embedded development is a cross toolchain, or more commonly known as a cross compiler.

All the programs (like GCC) run on a host system of a specific architecture (such as x86), but they produce binary code (executables) to run on a different architecture (for example, ARM). This is called cross compilation and is the typical way of building embedded software. It is possible to compile natively, running GCC on your target. Before searching for a prebuilt toolchain or building your own, it's worth checking to see if one is included with your target hardware's Board Support Package (BSP) if you have one.

Introduction:-

When talking about toolchains, one must distinguish three different machines:

- The build machine, on which the toolchain is built
- The host machine, on which the toolchain is executed
- The target machine, for which the toolchain generates code

From these three different machines, we distinguish four different types of toolchain building processes:

- A native toolchain, as can be found in normal Linux distributions, has usually been compiled on x86, runs on x86 and generates code for x86.
- A cross-compilation toolchain, which is the most interesting toolchain type for embedded development, is typically compiled on x86, runs on x86 and generates code for the target architecture (be it ARM, MIPS, PowerPC or any other architecture supported by the different toolchain components)
- A cross-native toolchain, is a toolchain that has been built on x86, but runs on your target architecture and generates code for your target

architecture. It's typically needed when you want a native GCC on your target platform, without building it on your target platform.

- A Canadian build is the process of building a toolchain on machine A, so that it runs on machine B and generates code for machine C. It's usually not really necessary.

Introduction to cross-compiling for Linux:

- Host vs Target

A compiler is a program that turns source code into executable code. Like all programs, a compiler runs on a specific type of computer, and the new programs it outputs also run on a specific type of computer.

The computer the compiler runs on is called the **host**, and the computer the new programs run on is called the **target**. When the host and target are the same type of machine, the compiler is a **native compiler**. When the host and target are different, the compiler is a **cross compiler**.

- **Why cross-compile?**

In theory, a PC user who wanted to build programs for some device could get the appropriate target hardware (or emulator), boot a Linux distro on that, and compile natively within that environment. While this is a valid approach (and possibly even a good idea when dealing with something like a Mac Mini), it has a few prominent downsides for things like a linksys router or iPod:

- **Speed** - Target platforms are usually much slower than hosts, by an order of magnitude or more. Most special-purpose embedded hardware is designed for low cost and low power consumption, not high performance. Modern emulators (like qemu) are actually faster than a lot of the real world hardware they emulate, by virtue of running on high-powered desktop hardware.
- **Capability** - Compiling is very resource-intensive. The target platform usually doesn't have gigabytes of memory and hundreds of gigabytes of disk space the way a desktop does; it may not even have the resources to build "hello world", let alone large and complicated packages.
- **Availability** - Bringing Linux up on a hardware platform it's never run on before requires a cross-compiler. Even on long-established platforms like Arm or Mips, finding an up-to-date full-featured prebuilt native environment for a given target can be hard. If the platform in question isn't normally used as a development workstation, there may not be a recent prebuilt distro readily available for it, and if there is it's probably out of date. If you have to build your

own distro for the target before you can build on the target, you're back to cross-compiling anyway.

- **Flexibility** - A fully capable Linux distribution consists of hundreds of packages, but a cross-compile environment can depend on the host's existing distro from most things. Cross compiling focuses on building the target packages to be deployed, not spending time getting build-only prerequisites working on the target system.
- **Convenience** - The user interface of headless boxes tends to be a bit crampled. Diagnosing build breaks is frustrating enough as it is. Installing from CD onto a machine that hasn't got a CD-ROM drive is a pain. Rebooting back and forth between your test environment and your development environment gets old fast, and it's nice to be able to recover from accidentally lobotomizing your test system.

➢ **BOOTLOADER:**

Boot-loader is a piece of code that runs before any operating system is running. Boot-loader are used to boot other operating systems, usually each operating system has a set of boot-loaders specific for it. Boot-loaders usually contain several ways to boot the OS kernel and also contain commands for debugging and/or modifying the kernel environment. Since it is usually the first software to run after power up or reset, it is highly processor based and board specific. Therefore, it's nearly possible to develop a boot-loader suited for every platform. The boot-loader performs the necessary initializations to prepare the system for the Operating system. It Initialized critical hardware components i.e. SDRAM controller, I/O controller, graphics controller, initialize system memory.Boot-loaders vary from CPU to CPU. Installation is only in non-volatile memory i.e. flash.

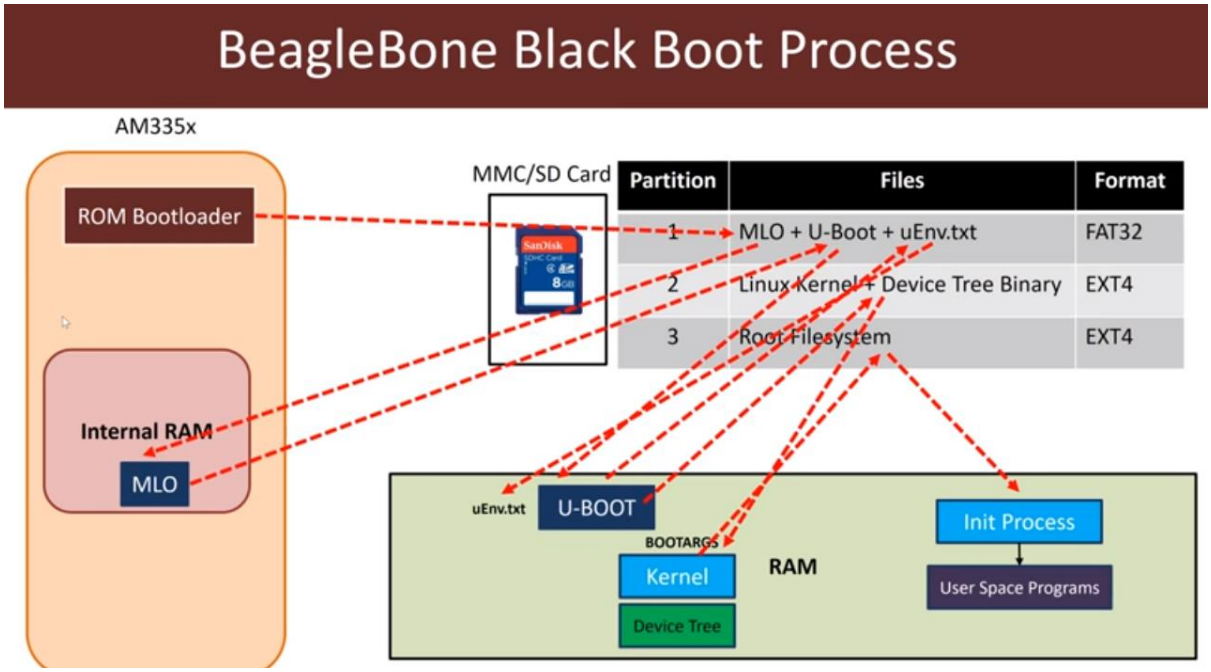
➢ **U-BOOT:**

U-Boot is both a first-stage and second-stage boot-loader. U-Boot runs a command-line interface on a serial port. Using the console, users can load and boot a kernel, possibly changing parameters from the default. There are also commands to read device information, read and write flash memory, download files (kernels, boot images, etc.) from the serial port or network, manipulate device trees. U-Boot is the boot-loader commonly used on our SoCs. Amongst others, it provides the basic infrastructure to bring up a board to a point where it can load a linux kernel and start booting your operating system.U-Boot's commands are actually generalized commands which can be used to read or write any arbitrary data. Using these commands, data can be read from or written to any storage system that U-Boot supports. U-Boot can also read/write from file systems. This way,

rather than requiring the data that U-Boot will load to be stored at a fixed location on the storage device, U-Boot can read the file system to search for and load the kernel, device tree, etc., by pathname.

Flow Diagram:

- **Typical boot flow of Boot-loader and U-Boot:**

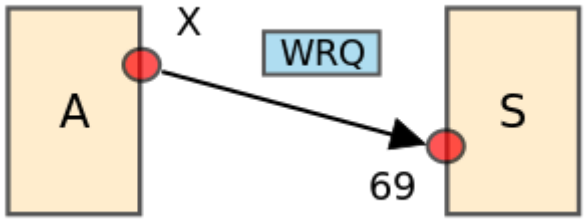


Trivial File Transfer Protocol (TFTP)

Trivial **F**ile **T**ransfer **P**rotocol (**TFTP**) is a simple lockstep File Transfer Protocol which allows a client to get a file from or put a file onto a remote host. One of its primary uses is in the early stages of nodes booting from a local area network. TFTP has been used for this application because it is very simple to implement.

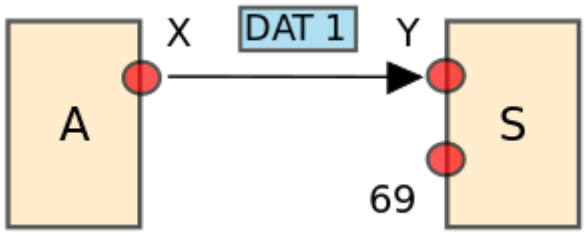
Due to its simple design, TFTP can be easily implemented by code with a small memory footprint. It is therefore the protocol of choice for the initial stages of any network booting strategy like BOOTP, PXE, BSDP, etc., when targeting from highly resourced computers to very low resourced Single-board computers (SBC) and System on a Chip (SoC). It is also used to transfer firmware images and configuration files to network appliances like routers, firewalls, IP phones, etc.

Host writing to the server



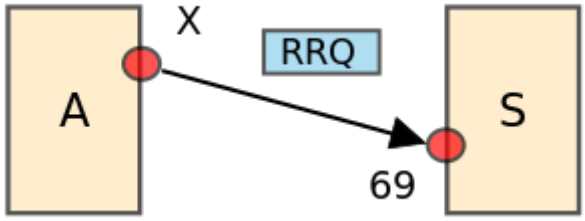
Host ‘A’ requests to write

Server ‘S’ acknowledges request

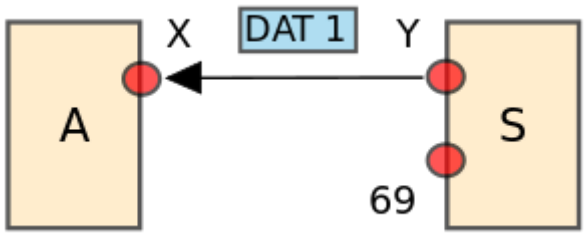


Host ‘A’ sends numbered data packets

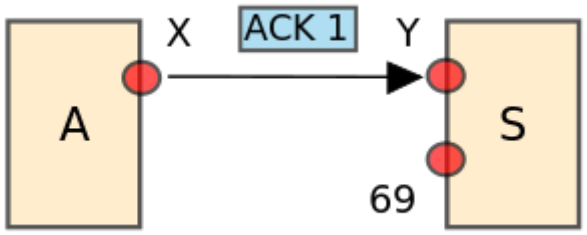
Host reading from the server



Host ‘A” requests to read



Server 'S' sends data packet



Host 'A" acknowledges data packet

TFTP includes no login or access control mechanisms. Care must be taken when using TFTP for file transfers where authentication, access control, confidentiality, or integrity checking are needed.

NFS Server

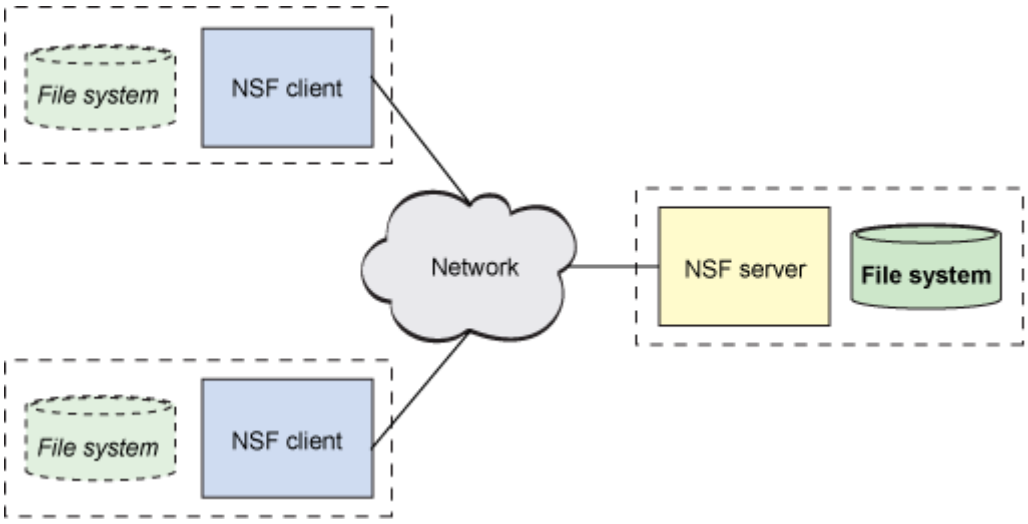
The **N**etwork **F**ile **S**ystem (NFS) is a way of mounting Linux discs/directories over a network. An NFS server can export one or more directories that can then be mounted on a remote Linux machine.

NFS allows remote hosts to mount file systems over a network and interact with those file systems as though they are mounted locally.

Currently, there are three versions of NFS. NFS version 2 (NFSv2) is older and widely supported. NFS version 3 (NFSv3) supports safe asynchronous writes and is more robust at error handling than NFSv2; it also supports 64-bit file sizes and offsets, allowing clients to access more than 2Gb of file data. All versions of NFS can use *Transmission Control Protocol* (*TCP*) running over an IP network.

There is no need for users to have separate home directories on every network machine. Home directories could be set up on the NFS server and made available throughout the network.

Storage devices such as floppy disks, CDROM drives, and USB Thumb drives can be used by other machines on the network. This may reduce the number of removable media drives throughout the network.



Kernel Compilation:-

How to compile and install Linux Kernel 4.20.12

The procedure to build (compile) and install the latest Linux kernel from source is as follows:

Grab the latest kernel from kernel.org

Verify kernel

Untar the kernel tarball

Copy existing Linux kernel config file

Compile and build Linux kernel 4.20.12

Install Linux kernel and modules (drivers)

Update Grub configuration

Reboot the system

Let us see all steps in details.

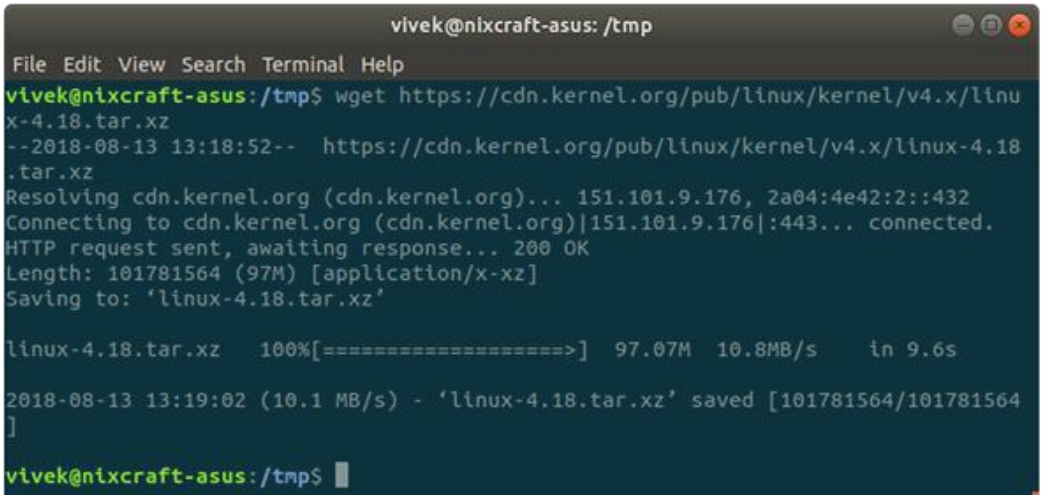
Step 1. Get the latest Linux kernel source code

Visit the official project site and download the latest source code. Click on the big yellow button that read as “Latest Stable Kernel“:

Download Linux Kernel Source Code

The filename would be linux-x.y.z.tar.xz, where x.y.z is actual Linux kernel version number. For example file linux-4.20.12.tar.xz represents Linux kernel version 4.20.12. Use the wget command to download Linux kernel source code:

$ wget https://cdn.kernel.org/pub/linux/kernel/v4.x/linux-4.20.12.tar.xz



Step 2. Extract tar.xz file

You really don't have to extract the source code in /usr/src. You can extract the source code in your $HOME directory or any other directory using the following unzx command or xz command:

$ unzx -v linux-4.20.12.tar.xz

OR

$ xz -d -v linux-4.20.12.tar.xz

Verify Linux kernel tartball with pgp

First grab the PGP signature for linux-4.20.12.tar:

$ wget https://cdn.kernel.org/pub/linux/kernel/v4.x/linux-4.20.12.tar.sign

Try to verify it:

$ gpg --verify linux-4.20.12.tar.sign

Sample outputs:

gpg: assuming signed data in 'linux-4.20.12.tar'

gpg: Signature made Sun 12 Aug 2018 04:00:28 PM CDT

gpg: using RSA key 79BE3E4300411886

gpg: Can't check signature: No public key

Grab the public key from the PGP keyserver in order to verify the signature i.e. RSA key ID 79BE3E4300411886 (from the above outputs):

$ gpg --recv-keys 79BE3E4300411886

Sample outputs:

gpg: key 79BE3E4300411886: 7 duplicate signatures removed

gpg: key 79BE3E4300411886: 172 signatures not checked due to missing keys

gpg: /home/vivek/.gnupg/trustdb.gpg: trustdb created

gpg: key 79BE3E4300411886: public key "Linus Torvalds <torvalds@kernel.org>" imported

gpg: no ultimately trusted keys found

gpg: Total number processed: 1

gpg: imported: 1

Now verify gpg key again with the gpg command:

$ gpg --verify linux-4.20.12.tar.sign

Sample outputs:

gpg: assuming signed data in 'linux-4.20.12.tar'

gpg: Signature made Sun 12 Aug 2018 04:00:28 PM CDT

gpg: using RSA key 79BE3E4300411886

gpg: Good signature from "Linus Torvalds <torvalds@kernel.org>" [unknown]

gpg: aka "Linus Torvalds <torvalds@linux-foundation.org>" [unknown]

gpg: WARNING: This key is not certified with a trusted signature!

gpg: There is no indication that the signature belongs to the owner.

Primary key fingerprint: ABAF 11C6 5A29 70B1 30AB E3C4 79BE 3E43 0041 1886

If you do not get “BAD signature” output from the “gpg –verify” command, untar/extract the Linux kernel tarball using the tar command, enter:

$ tar xvf linux-4.20.12.tar

Step 3. 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:

$ cd linux-4.20.12

$ cp -v /boot/config-$(uname -r) .config

Sample outputs:

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

Step 4. 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.

How to install GCC and development tools on a Debian/Ubuntu Linux

Type the following apt command or apt-get command to install the same:

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

See “Ubuntu Linux Install GNU GCC Compiler and Development Environment” for more info.

How to install GCC and development tools on a CentOS/RHEL/Oracle/Scientific Linux

Try yum command:

$ sudo yum group install "Development Tools"

OR

$ sudo yum groupinstall "Development Tools"

Additional packages too:

$ sudo yum install ncurses-devel bison flex elfutils-libelf-developenssl-devel

How to install GCC and development tools on a Fedora Linux

Run the following dnf command:

$ sudodnf group install "Development Tools"

$ sudodnf install ncurses-devel bison flex elfutils-libelf-developenssl-devel

Step 5. Configuring the kernel

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

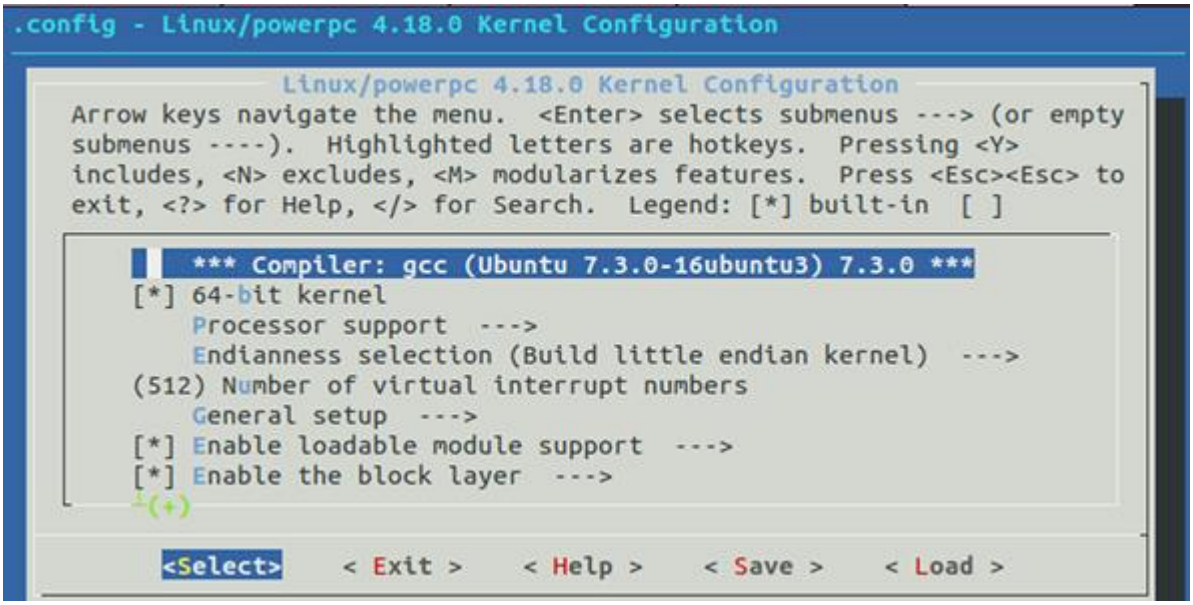
$ make menuconfig – Text based color menus, radiolists& dialogs. This option also useful on remote server if you wanna compile kernel remotely.

$ 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



How to compile and install Linux Kernel 4.20.12

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 6. How to compile a Linux Kernel

Start compiling and tocreate 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)

```
  AS      arch/x86/boot/compressed/head_64.o
  VOFFSET arch/x86/boot/compressed/../voffset.h
  CC      arch/x86/boot/compressed/string.o
  CC      arch/x86/boot/compressed/cmdline.o
  CC      arch/x86/boot/compressed/error.o
  OBJCOPY arch/x86/boot/compressed/vmlinux.bin
  RELOCS  arch/x86/boot/compressed/vmlinux.relocs
  HOSTCC  arch/x86/boot/compressed/mkpiggy
  CC      arch/x86/boot/compressed/early_serial_console.o
  CC      arch/x86/boot/compressed/kaslr.o
  CC      arch/x86/boot/compressed/cpuflags.o
  CC      arch/x86/boot/compressed/kaslr_64.o
  AS      arch/x86/boot/compressed/mem_encrypt.o
  CC      arch/x86/boot/compressed/pgtable_64.o
  CC      arch/x86/boot/compressed/eboot.o
  AS      arch/x86/boot/compressed/efi_stub_64.o
  AS      arch/x86/boot/compressed/efi_thunk_64.o
  CC      arch/x86/boot/compressed/misc.o
  GZIP    arch/x86/boot/compressed/vmlinux.bin.gz
  MKPIGGY arch/x86/boot/compressed/piggy.S
  AS      arch/x86/boot/compressed/piggy.o
  LD      arch/x86/boot/compressed/vmlinux
  ZOFFSET arch/x86/boot/zoffset.h
  OBJCOPY arch/x86/boot/vmlinux.bin
  AS      arch/x86/boot/header.o
  LD      arch/x86/boot/setup.elf
  OBJCOPY arch/x86/boot/setup.bin
  BUILD   arch/x86/boot/bzImage
Setup is 17052 bytes (padded to 17408 bytes).
System is 8357 kB
CRC e0320f3d
Kernel: arch/x86/boot/bzImage is ready  (#1)
```

Linux kernel compiled and bzImage is ready

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.

Install the Linux kernel modules

$ sudo make modules_install

```
INSTALL arch/powerpc/crypto/crc-vpmsum_test.ko
INSTALL arch/powerpc/crypto/crc32c-vpmsum.ko
INSTALL arch/powerpc/crypto/crct10dif-vpmsum.ko
INSTALL arch/powerpc/crypto/md5-ppc.ko
INSTALL arch/powerpc/crypto/sha1-powerpc.ko
INSTALL arch/powerpc/kernel/rtas_flash.ko
INSTALL arch/powerpc/kvm/kvm-hv.ko
INSTALL arch/powerpc/kvm/kvm-pr.ko
INSTALL arch/powerpc/kvm/kvm.ko
INSTALL arch/powerpc/oprofile/oprofile.ko
INSTALL arch/powerpc/platforms/powernv/opal-prd.ko
INSTALL arch/powerpc/platforms/pseries/cmm.ko
INSTALL arch/powerpc/platforms/pseries/hvcserver.ko
INSTALL arch/powerpc/platforms/pseries/pseries_energy.ko
INSTALL arch/powerpc/platforms/pseries/scanlog.ko
INSTALL arch/powerpc/sysdev/rtc_cmos_setup.ko
INSTALL block/bfq.ko
INSTALL block/kyber-iosched.ko
INSTALL block/mq-deadline.ko
INSTALL crypto/842.ko
INSTALL crypto/aes_ti.ko
INSTALL crypto/af_alg.ko
INSTALL crypto/algif_aead.ko
INSTALL crypto/algif_hash.ko
INSTALL crypto/algif_rng.ko
INSTALL crypto/algif_skcipher.ko
INSTALL crypto/ansi_cprng.ko
INSTALL crypto/anubis.ko
INSTALL crypto/arc4.ko
INSTALL crypto/asymmetric_keys/pkcs7_test_key.ko
INSTALL crypto/async_tx/async_memcpy.ko
INSTALL crypto/async_tx/async_pq.ko
INSTALL crypto/async_tx/async_raid6_recov.ko
INSTALL crypto/async_tx/async_tx.ko
INSTALL crypto/async_tx/async_xor.ko
INSTALL crypto/authenc.ko
INSTALL crypto/authencesn.ko
INSTALL crypto/blowfish_common.ko
INSTALL crypto/blowfish_generic.ko
INSTALL crypto/camellia_generic.ko
```

How to install the Linux kernel modules

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

$ sudo make install

```
[root@fedora28-nixcraft linux-4.18]#  make install
sh ./arch/x86/boot/install.sh 4.18.0 arch/x86/boot/bzImage \
        System.map "/boot"
[root@fedora28-nixcraft linux-4.18]#
```

make install output

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

initramfs-4.20.12.img

System.map-4.20.12

vmlinuz-4.20.12

Step 7. Update grub config

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

CentOS/RHEL/Oracle/Scientific and Fedora Linux

$ sudo grub2-mkconfig -o /boot/grub2/grub.cfg

$ sudo grubby --set-default /boot/vmlinuz-4.20.12

You can confirm the details with the following commands:

grubby --info=ALL | more

grubby --default-index

grubby --default-kernel

Debian/Ubuntu Linux

The following commands are optional as make install does everything for your but included here for historical reasons only:

$ sudo update-initramfs -c -k 4.20.12

$ sudo update-grub

How to build and install the latest Linux kernel from source code

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.

Reboot Linux computer and boot into your new kernel

Just issue the reboot command or shutdown command:

reboot

Verify new Linux kernel version after reboot:

$ uname -mrs

Sample outputs:

Linux 4.20.12 x86_64

Steps to compile and install kernel:

- Installing the sources.
- Configuring the kernel (choosing which features and Drivers to compile).
- Compiling the kernel (i.e. typing a single command, and watching...).
- Installing the compiled kernel.
- Updating the boot loader to recognize the new kernel.
- Booting...
- Making the new kernel become the default.

Kernel Construction:

- Kernel modules need recompiling, linking and loading.
- Top-level directory: /usr/src/linux-header-3.1126-g.....
    - arch/ firmware/ kernel/ scripts/ block/ fs/ lib/ security/ crypto/ include/ mm/ sound/ Documentation/ init/ net/ usr/ drivers/ ipc/ samples/ virt/

Configuring the Kernel:

- A step before kernel compilation.
- It specify which drivers and features to compile.

- Also describes how to compile them (as a module or inside the kernel's main file).
- Command: make config or make ARCH=arm gconfig.
- The configuration process generates a file called ".config" in the top directory of the kernel sources.

Make utility for kernel configuration:

- Clean : remove most generated files, keep config file .
- Mrproper: remove all generated files + config + backup files.
- Config: update current config utilizing a line-oriented program.
- Menuconfig: update current config utilizing a menu based program.
  - Makes it easier to backup and correct mistakes.
- Xconfig: use QT-based front end.
- Gconfig: GTK-based front end
- kconfig file: Drives the configuration process for the features contained within its subdirectory.
  - During the build process, the .config file is processes in C header file i.e. autoconfig.h
  - collection of definition with simple format.

Advantages:

- A kernel can provide improvements, extra features, specialization and extreme configurability.
- Developers can add some additional features from other kernel.
- You can control the minimum and maximum frequencies at that the CPU can scale up or down to.

Disadvantages:

- If you are not careful enough while using the kernel you can end up harming your device to the extent of bricking it.

List of several tools to configure:

make config	Text based configuration. The options are prompted one after another. All options need to be answered, and out-of-order access to former options

	is not possible.
make menuconfig	An curses-based pseudo-graphical menu (only text input). Navigate through the menu to modify the desired options.
make xconfig	Graphical menu using Qt5. Requires dev-qt/qtgui to be installed.
make gconfig	Graphical menu using GTK+. Requires x11-libs/gtk+, dev-libs/glib,and gnome-base/libglade to be installed.
Kconfig	Drives the configuration process for the features contained within its subdirectory. During the build process, the .config file is processes in C header file i.e. autoconfig.h collection of definition with simple format.

Root File System (rootfs):

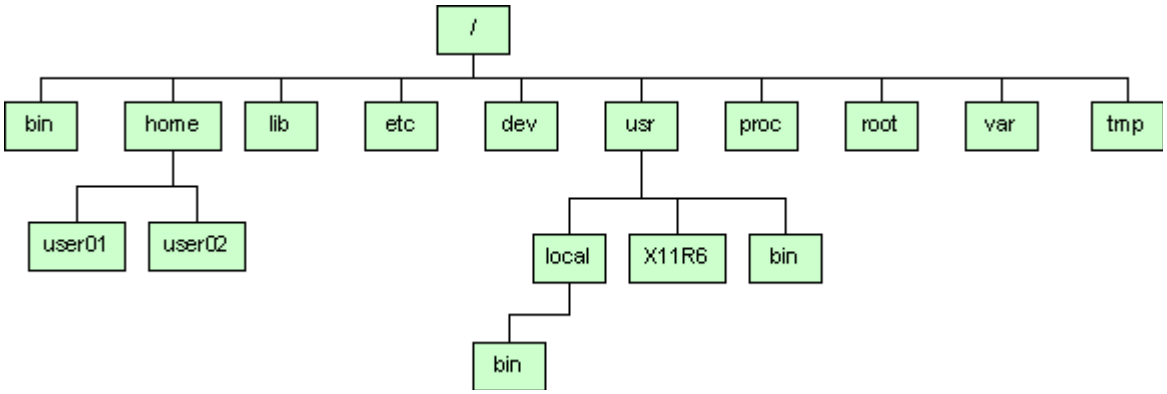
The root file system (named rootfs in our sample error message) is the most basic component of Linux. A root file system contains everything needed to support a full Linux system. It contains all the applications, configurations, devices, data, and more. **Without the root file system, your Linux system cannot run.**

The contents of the root filesystem must be adequate to boot, restore, recover, and/or repair the system.

- To boot a system, enough software and data must be present on the root partition to mount other filesystems. This includes utilities, configuration, boot loader information, and other essential start-up data. /usr, /opt, and /var are designed such that they may be located on other partitions or filesystems.

- To enable recovery and/or repair of a system, those utilities needed by an experienced maintainer to diagnose and reconstruct a damaged system must be present on the root filesystem.
- To restore a system, those utilities needed to restore from system backups (on floppy, tape, etc.) must be present on the root filesystem.

The root directory generally doesn't contain any files, except perhaps on older systems where the standard boot image for the system, usually called /vmlinuz was kept there. (Most distributions have moved those files the /boot directory. Otherwise, all files are kept in subdirectories under the root filesystem:



/bin

Commands needed during bootup that might be used by normal users (probably after bootup).

/sbin

Like /bin, but the commands are not intended for normal users, although they may use them if necessary and allowed. /sbin is not usually in the default path of normal users, but will be in root's default path.

/etc

Configuration files specific to the machine.

/root

The home directory for user root. This is usually not accessible to other users on the system

/lib

Shared libraries needed by the programs on the root filesystem.

/lib/modules

Loadable kernel modules, especially those that are needed to boot the system when recovering from disasters (e.g., network and filesystem drivers).

/dev

Device files. These are special files that help the user interface with the various devices on the system.

/tmp

Temporary files. As the name suggests, programs running often store temporary files in here.

/boot

Files used by the bootstrap loader, e.g., LILO or GRUB. Kernel images are often kept here instead of in the root directory. If there are many kernel images, the directory can easily grow rather big, and it might be better to keep it in a separate filesystem. Another reason would be to make sure the kernel images are within the first 1024 cylinders of an IDE disk. This 1024 cylinder limit is no longer true in most cases. With modern BIOSes and later versions of LILO (the LInuxLOader) the 1024 cylinder limit can be passed with logical block addressing (LBA). See the **lilo** manual page for more details.

/mnt

Mount point for temporary mounts by the system administrator. Programs aren't supposed to mount on /mnt automatically. /mnt might be divided into subdirectories (e.g., /mnt/dosa might be the floppy drive using an MS-DOS filesystem, and /mnt/exta might be the same with an ext2 filesystem).

/proc, /usr, /var, /home

Mount points for the other filesystems. Although /proc does not reside on any disk in reality it is still mentioned here.