



**西北工业大学**  
NORTHWESTERN POLYTECHNICAL UNIVERSITY

## Lab I Report

**Report Subject: OS Experiment - Lab I**

**Student ID : 2019380141**

**Student Name : ABID ALI**

**Experiment Name : Linux Shell Commands、 C programming  
and debugger tools**

**Email : abiduu354@gmail.com**

**Data: 2021/10/16**

## Objective:

- 1) To familiar with Linux Operating system.
- 2) To practice the usage of Linux shell commands.
- 3) To practice with the creation of a Makefile.
- 4) To get acquainted with C programming.
- 5) To practice your code reading skills.

## Equipment:

VMware with Ubuntu Linux

## Methodology:

Answer all the questions in this lab sheet.

### 1. Tools

Tools are important for every programmer. If we spend time learning to use our tools, you will save even more time when you are writing and debugging code. This section will introduce the most important tools for this course.

#### 1) **Make**

GNU Make is program that is commonly used to build other programs. When you run make, GNU Make looks in your current directory for a file named Makefile and executes the commands inside, according to the makefile language.

```
#This is a makefile

all:

    gcc -o first first.c

#Compile for debugging

deb:

    gcc -Wall -g first.c

exec:

    ./first
```

The building block of GNU Make is a rule. We just created a rule, whose target is first and then the output is executed. When we run make first, GNU Make will execute the

steps in the recipe and print the program ". Files can also contain a list of dependencies, which are other targets that must be executed before the first.

## 2) **GDB: The GNU Debugger**

GDB is a debugger that supports C, C++, and other languages. We will not be able to debug our projects effectively without advanced knowledge of GDB.

**Commands:**

```
gcc -Wall -g first.c
```

or

```
gcc -g -w first.c
```

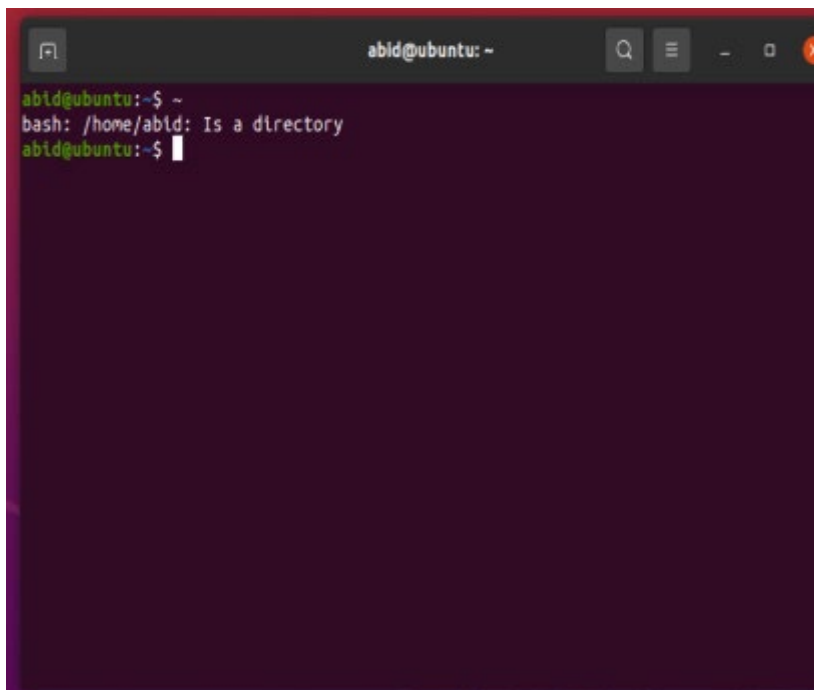
```
./a.out
```

## 2. Practice:

### 1) Practice and answer following question.

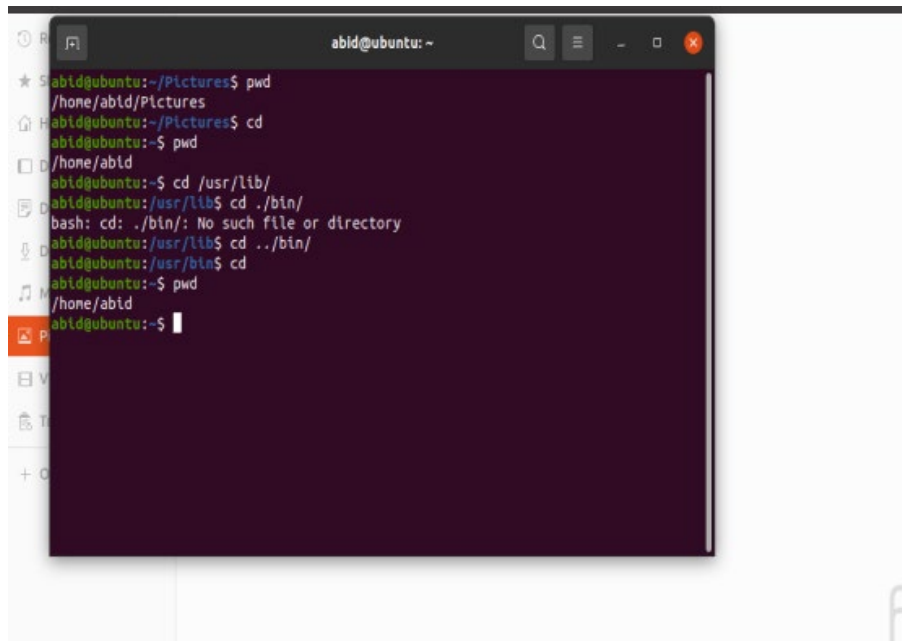
- (1) How do you find your home directory quickly? And change the directory to your home.

**Ans :** ~ and cd ~

A screenshot of a terminal window titled 'abid@ubuntu: ~'. The terminal shows a sequence of commands and their outputs: the user enters '~', the prompt changes to 'bash: /home/abid: Is a directory', and then the user enters 'cd ~', returning to the 'abid@ubuntu: ~' prompt. The terminal has a dark purple background and a standard Ubuntu window title bar with search, menu, and window control icons.

```
abid@ubuntu:~$ ~
bash: /home/abid: Is a directory
abid@ubuntu:~$
```

- (2) Type these commands “**cd /usr/lib/**”, “**cd ./bin/**”, “**cd ..**”, and “**pwd**”. Give the current directory which you are located. What are the meanings of **.** and **..** ?

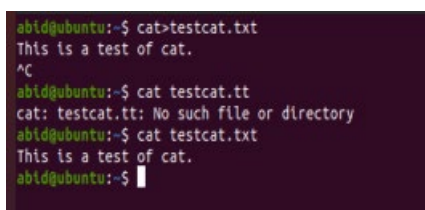


```
abid@ubuntu: ~  
$ cd /usr/lib/  
abid@ubuntu: /usr/lib$ cd ./bin/  
bash: cd: ./bin/: No such file or directory  
abid@ubuntu: /usr/lib$ cd ../bin/  
abid@ubuntu: /usr/bin$ cd  
abid@ubuntu: ~$ pwd  
/home/abid  
abid@ubuntu: ~$
```

**Answer:** The difference between “**.**” and “**..**” is “**.**” represents the current directory and “**..**” represents the parent directory or one directory up.

I am now at /home/abid

- (3) Type “**cd**” to go back to your home directory. Type “**cat > testcat.txt**” in the command line. After pressing return, type the following line of text “This is a test of cat.”, and then press **ctrl-d**. Type “**cat testcat.txt**” again. What do you see?



```
abid@ubuntu: ~$ cat>testcat.txt  
This is a test of cat.  
^C  
abid@ubuntu: ~$ cat testcat.tt  
cat: testcat.tt: No such file or directory  
abid@ubuntu: ~$ cat testcat.txt  
This is a test of cat.  
abid@ubuntu: ~$
```

**Answer:**

I have used those following commands to create a txt file(**cat > testcat.txt**) then gave input of what to store in the txt file(“This is a test of cat.”) then used a command(**ctrl-d**) to store those in the file and lastly checked(**cat testcat.txt**) what is inside that txt file that I have created

- (4) Type “**find . -name testcat.txt -print > list.lst**” in the command line. You will find a file “**list.lst**” in your current directory. Use **cat** commands to show its contents. What is the result? What is the meaning of **> list.lst**?

**Answer:**

```
abid@ubuntu:~$ find . -name testcat.txt -print>list.lst
abid@ubuntu:~$ cat list.lst
./testcat.txt
abid@ubuntu:~$
```

I have created another file in the home directory named “list.lst”. After I have used cat command to see what is in that file the output shows “./testcat.txt”.(the location of the ‘testcat.txt’ file)

The meaning of ‘>’ is the terminal is asking for an input and the input(print) was to create ‘list.lst’ file in that directory that will contain the location of “testcat.txt” file’s location.

- (5) Go to “/” directory and type the command to find the bin directory.

```
abid@ubuntu:~$ pwd
/home/abid
abid@ubuntu:~$ cd /usr/bin/
abid@ubuntu:~$
```

- (6) Open two terminals on Linux. In each terminal, type the command “ps”. Give the process number (PID) of the bash process in both terminals. Why they are different?

**Answer:**

```
abid@ubuntu:~$ ps
  PID TTY          TIME CMD
 3323 pts/0    00:00:00 bash
 3330 pts/0    00:00:00 ps
abid@ubuntu:~$
```

```
abid@ubuntu:~$ ps
  PID TTY          TIME CMD
 3340 pts/1    00:00:00 bash
 3346 pts/1    00:00:00 ps
abid@ubuntu:~$
```

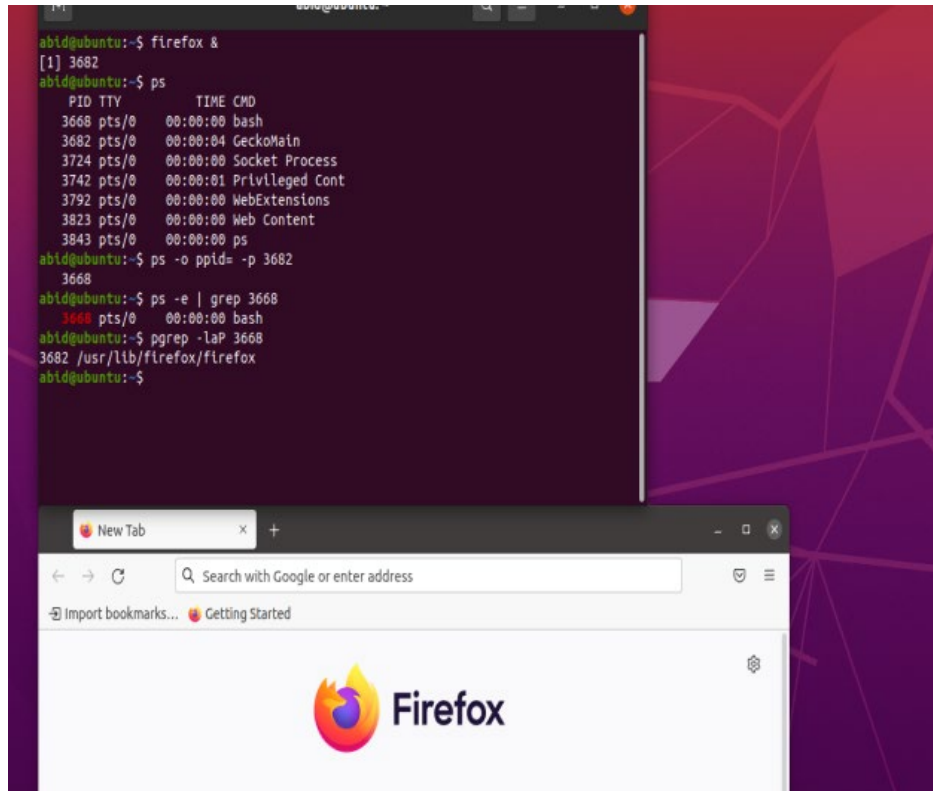
PID is the process id for a individual task at any given moment. It is unique every time it’s ran. That’s why running same command ps (process status) even in desktop from different terminal causes to show different PID for same tasks. Cause this id is unique to the exact moment

**PID -1<sup>st</sup> terminal :3323**

**PID -2<sup>nd</sup> terminal :3340**

- (7) Start the Fire-Fox web browser by typing "**firefox &**" at the command line and type "ps" in the terminal. Can you see the process of Fire-Fox web browser? Find the parent process and child process.

**Answer:**



To find parents and child id,  
From running 'firefox &', I can find the PID of firefox(3682)  
Now , To determine the parent process of a specific process, we use the ps command.

```
ps -o ppid= -p 3682
```

We only get the parent process ID

```
3668
```

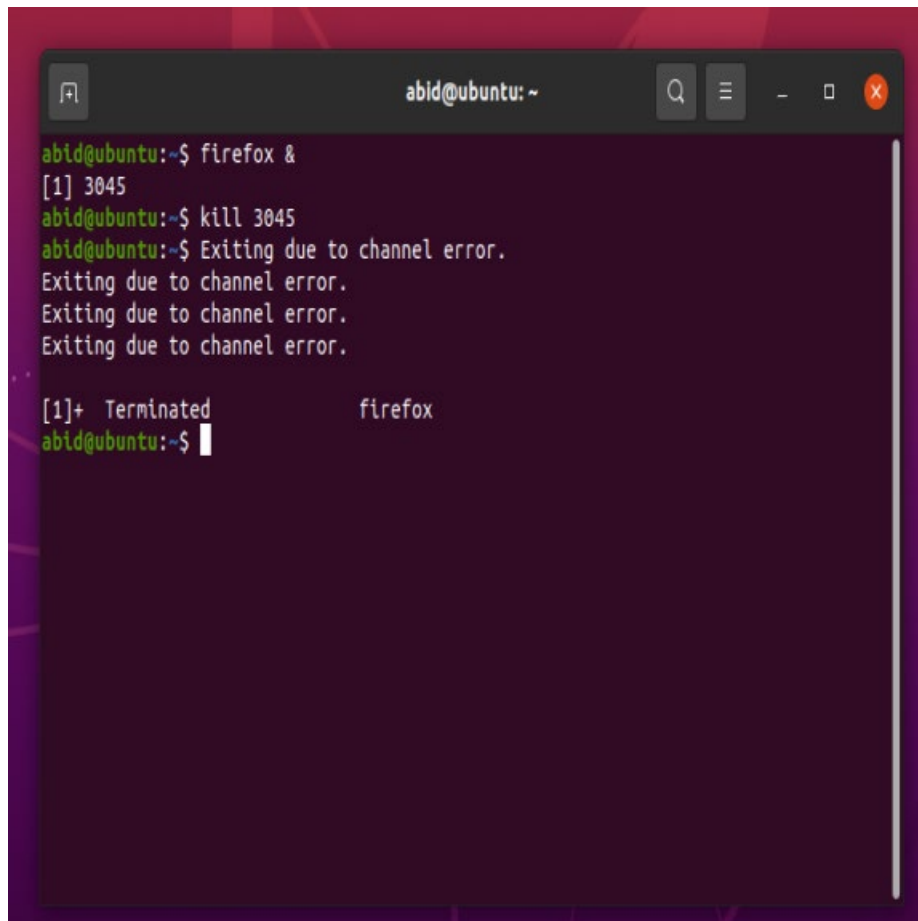
Now ,we use ps command

```
ps -e | grep 3668
```

We get the name of the program

```
pgrep -laP 3668
```

- (8) Use the command “kill” to kill the processes of Fire-Fox. Give the command you have used. What did you see after killing their processes?



```
abid@ubuntu: ~  
abid@ubuntu:~$ firefox &  
[1] 3045  
abid@ubuntu:~$ kill 3045  
abid@ubuntu:~$ Exiting due to channel error.  
Exiting due to channel error.  
Exiting due to channel error.  
Exiting due to channel error.  
[1]+  Terminated           firefox  
abid@ubuntu:~$
```

### Answer:

We type “firefox &” that opens the program along with that we get the ID  
ID = 3045

Kill process\_ID will kill the program:  
Kill 3045

- 2) Now you are ready to write a program that reveals its own executing structure. The first.c provides a rather complete skeleton. You will need to modify it to get the addresses that you are looking for.

```
#include <stdio.h>
#include <stdlib.h>

int var1;          /* a statically allocated variable */

int my_func (int i) {      /* a my_func sive function */
    int j = i;            /* a stack allocated variable within a my_func sive function */
    printf("my_func call (i)%d: stack@ %lx\n", i, (long unsigned int) &j); /* prints the address of j */
    if (i > 0) {
        return my_func (i-1);
    }
    return 0;
}

volatile int stuff = 7;    /* a statically allocated, pre-initialized variable */

int main (int argc, char *argv[]) {
    volatile int i = 0;    /* a stack allocated variable */
    volatile char *buf1 = malloc(10); /* dynamically allocate some stuff */
    volatile char *buf2 = malloc(10); /* and some more stuff */
    printf("Answer to the question Number 2(1)\n");
    printf("main @ %lx\n", (long unsigned int) &main); /* fix to print address of main */
    printf("my_func @ %lx\n", (long unsigned int) &my_func ); /* print address of my_func */
    printf("main call (i):stack@ %lx\n", (long unsigned int) &i); /* get address of the stack variable */
    printf("static var1: %lx\n", (long unsigned int) &var1); /* get address of the static variable */
    printf("static stuff: %lx\n", (long unsigned int) &stuff); /* get address of a static variable */
    printf("Heap: malloc 1: %lx\n", (long unsigned int) buf1);
    printf("Heap: malloc 2: %lx\n", (long unsigned int) buf2);
    my_func (3);
    printf("Answer to the question Number 2(2)\n");
    printf("Pointed to by argv is %s\n",argv[0]);
    printf("The value of argv is %s\n",*argv);
    printf("The value of argc is %d\n",argc);
    return 0;
}
```



- (1) output the variables and their addresses belongs to stack, heap and data segment. The output of the solution looks like the following (the addresses may be different).

...

### Answer to the question No.2(1)

```
Answer to the question Number 2(1)
main @ 555f0df58217
my_func @ 555f0df581a9
main call (i):stack@ 7fff42cfb014
static var1: 555f0df5b018
static stuff: 555f0df5b010
Heap: malloc 1: 555f0f7f52a0
Heap: malloc 2: 555f0f7f52c0
my_func call (i)3: stack@ 7fff42cfafe4
my_func call (i)2: stack@ 7fff42cfafb4
my_func call (i)1: stack@ 7fff42cfaf84
my_func call (i)0: stack@ 7fff42cfaf54
```

- (2) While you are looking through gdb to debug first.c, think about the following questions and put your answers in the file.

- What is the value of argv? (hint: print argv)
- What is pointed to by argv? (hint: print argv[0])

What is the address of the function main?

- Try info stack. Explain what you see.
- Try info registers. Which registers are holding aspects of the program that you recognize?

### Answer to the question No.2(2)

I got the value of argv by using GDB.

This is the value of argv

```
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
first.c
8+20 int main(int argc, char *argv[]) {
21     volatile int i = 0; /* a stack allocated variable */
22     volatile char *buf1 = malloc(10); /* dynamically allocate some stuff */
23     volatile char *buf2 = malloc(10); /* and some more stuff */
24     printf("Answer to the question Number 2(1)\n");
25     printf("main @ %lx\n", (long unsigned int) &main); /* fix to print address of main */
26     printf("my_func @ %lx\n", (long unsigned int) &my_func); /* fix to print address of my func */
native process 10340 In: main
(gdb) run
Starting program: /home/abid/Lab1/a.out
Breakpoint 1, main (argc=21845, argv=0x7ffff7faefc8 <_exit_funcs_lock>) at first.c:20
(gdb)
```

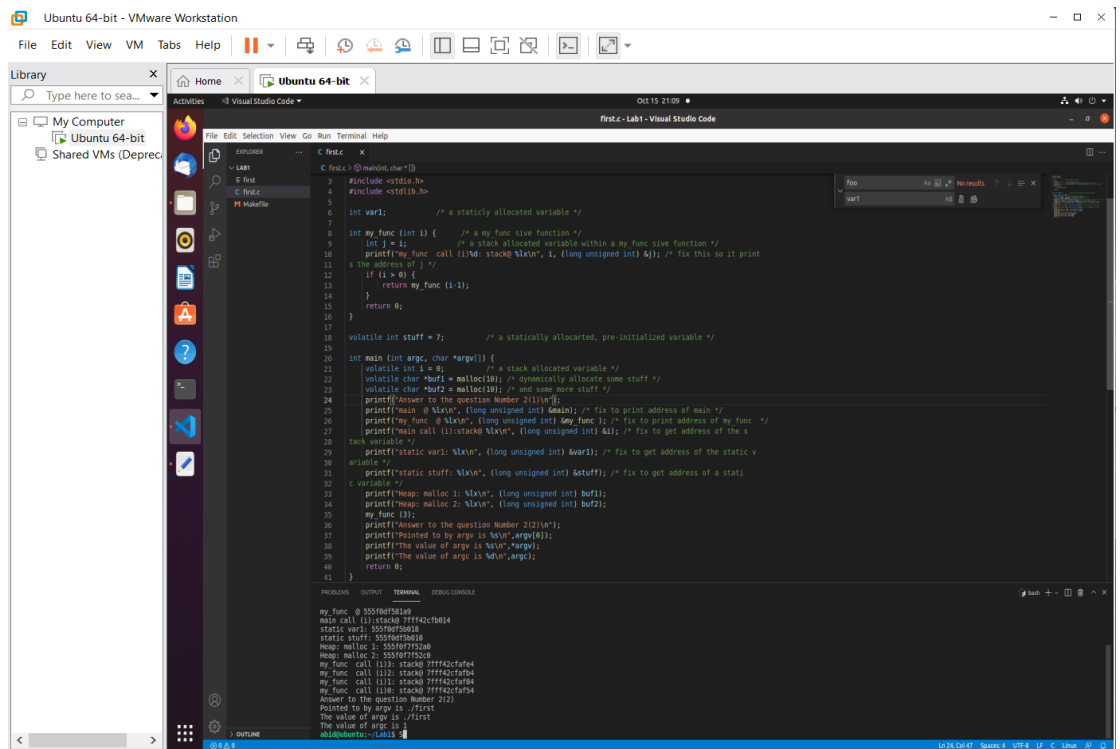
```
Breakpoint 1, main (argc=21845, argv=0x7ffff7faefc8 <_exit_funcs_lock>) at first.c:20
(gdb) print argc
$1 = 21845
(gdb) $
```

```

main call (i):stack@ 7fffffffde14
5
6     int var1;          /* a statically allocated variable */
7
8     int my_func (int i) { /* a my_func sive function */
9         int j = i;      /* a stack allocated variable within a my_func sive function */
10        printf("my_func call (i)%d: stack@ %lx\n", i, (long unsigned int) &j); /* fix this so it print
11        s the address of j */
my_func call (i): stack@ 7fffffffdd54
exec No process in: number 2(2)
(gdb) nextl (process 8838) exited normally
The program is not being run.
(gdb) next
The program is not being run.
(gdb)

```

We can see that , pointed to by argv (i.e argv[0]) is **./first**



```

Answer to the question Number 2(2)
Pointed to by argv is ./first
The value of argv is ./first
The value of argc is 1
abid@ubuntu:~/Lab1$

```

We know that from assembly language, \$1, \$2, \$3 are called registers where values are stored and manipulated.

```

$2 = 0x5555555550c0 <start> "\363\017\036\372\061\355I\211\321"H\211\342H\203\344\360PTL\215\005\066\003"
(gdb)

```

Here, we can see that values are continuously going through different register and different operation are performed.

```
(gdb) print argv
$2 = (char **) 0x7ffff7faefc8 <__exit_funcs_lock>
(gdb) s
```

```
(gdb) print argv[0]
$3 = 0x0
(gdb)
```

- 3) Write a shell script to capture Process ID by the process name and kill it if it exists.

## pstree

```
abid@ubuntu:~$ pstree
systemd--ModemManager--2*[{ModemManager}]
--NetworkManager--2*[{NetworkManager}]
--VGAAuthService
--accounts-daemon--2*[{accounts-daemon}]
--acpid
--avahi-daemon--avahi-daemon
--bluetoothd
--colord--2*[{colord}]
--cron
--cups-browsed--2*[{cups-browsed}]
--cupsd--dbus
--dbus-daemon
--fwupd--4*[{fwupd}]
--gdm3--gdm-session-wor--gdm-x-session--Xorg--{Xorg}
--gnome-session-b--ssh-agent
--2*[{gnome-+
--2*[{gdm-x-session}]
--2*[{gdm-session-wor}]
--2*[{gdm3}]
--gnome-keyring-d--3*[{gnome-keyring-d}]
--irqbalance--{irqbalance}
--2*[{kerneloops}]
--networkd-dispat
--polkitd--2*[{polkitd}]
--rsyslogd--3*[{rsyslogd}]
--rtkit-daemon--2*[{rtkit-daemon}]
--snapd--14*[{snapd}]
--switcheroo-cont--2*[{switcheroo-cont}]
--systemd--(sd-pam)
--at-spi-bus-laun--dbus-daemon
--3*[{at-spi-bus-laun}]
--at-spi2-registr--2*[{at-spi2-registr}]
--dbus-daemon
--dconf-service--2*[{dconf-service}]
--evolution-addre--5*[{evolution-addre}]
--evolution-calen--8*[{evolution-calen}]
--evolution-sourc--3*[{evolution-sourc}]
--gjs--4*[{gjs}]
```

```

--gsd-sharing---3*[{gsd-sharing}]
--gsd-smartcard---4*[{gsd-smartcard}]
--gsd-sound---3*[{gsd-sound}]
--gsd-usb-protect---3*[{gsd-usb-protect}]
--gsd-wacom---2*[{gsd-wacom}]
--gsd-wwan---3*[{gsd-wwan}]
--gsd-xsettings---3*[{gsd-xsettings}]
--gvfs-afc-volume---3*[{gvfs-afc-volume}]
--gvfs-goa-volume---2*[{gvfs-goa-volume}]
--gvfs-gphoto2-vo---2*[{gvfs-gphoto2-vo}]
--gvfs-mtp-volume---2*[{gvfs-mtp-volume}]
--gvfs-udisks2-vo---3*[{gvfs-udisks2-vo}]
--gvfsd|gvfsd-trash---2*[{gvfsd-trash}]
|2*[{gvfsd}]
--gvfsd-fuse---5*[{gvfsd-fuse}]
--gvfsd-metadata---2*[{gvfsd-metadata}]
--ibus-portal---2*[{ibus-portal}]
--ibus-x11---2*[{ibus-x11}]
--pulseaudio---3*[{pulseaudio}]
--snap-store---4*[{snap-store}]
--tracker-miner-f---4*[{tracker-miner-f}]
--vmtoolsd---3*[{vmtoolsd}]
--xdg-desktop-por---4*[{xdg-desktop-por}]
--xdg-desktop-por---3*[{xdg-desktop-por}]
--xdg-document-po---5*[{xdg-document-po}]
--xdg-permission---2*[{xdg-permission-}]
--systemd-journal
--systemd-logind
--systemd-resolve
--systemd-timesyn---{systemd-timesyn}
--systemd-udev
--udisksd---4*[{udisksd}]
--unattended-upgr---{unattended-upgr}
--upowerd---2*[{upowerd}]
--vmtoolsd---2*[{vmtoolsd}]
--vmware-vmblock---2*[{vmware-vmblock-}]
--whoopsie---2*[{whoopsie}]
--wpa_supplicant
abid@ubuntu:~$ ps -ef | grep thunderbird | grep -v grep | awk '{print $2}' | xargs kill
abid@ubuntu:~$

```

```

--ibus-daemon---2*[{ibus-daemon}]
--thunderbird---37*[{thunderbird}]
--xorg-x11-xkb---5*[{xorg-x11-xkb}]

```

**ps -ef | grep thunderbird | grep -v grep | awk '{print \$2}' | xargs kill**

### **Problems:**

Uncomfortable with the Linux OS and how to use terminal and what to type. The linux keyword was unfamiliar with me

### **Solution:**

To solve those problems I looked for information in internet. In order to understand some questions and procedure I also asked the teacher to help me understand them. And provided instructions helped to solve some of my errors during the experiment.

### **Conclusion:**

At the beginning, I was unfamiliar with Linux operation system and terminal. Gradually, reading lot of article and reading teachers ppt then I solved those problem one by one. In this experiment ,small helps and suggestions from the teacher was very helpful that saved my time.I enjoyed the practical and learned lot of interesting things.

### **Attachments:**

- 1) ABID ALI\_2019380141\_OS(Lab 1).docx
- 2) ABID ALI\_2019380141\_OS(Lab 1).pdf
- 3) Code\_ABID ALI\_2019380141(Lab-1).zip