CHAROTAR UNIVERSITY OF SCIENCE & TECHNOLOGY

FACULTY OF TECHNOLOGY AND ENGINEERING

Devang Patel Institute of Advance Technology & Research

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

CE248 Operating System

Semester: IV

Academic Year: 2019-20

PRACTICAL LIST

Sr. No.	AIM OF THE PRACTICAL	Date	Page No.	Remark
1.	**Working of Different Kernels:** A. UNIX Architecture B. Types of OS- Linux, Unix, MAC, Window etc. C. Flavors of LINUX		4	
2.	Study of Unix Architecture and the following Unix commands with option: **User Access:** login, logout, passwd, exit **Help:** man, help **Directory:** mkdir, rmdir, cd, pwd, ls, mv **Editor:** vi, gedit, ed, sed **File Handling / Text Processing:** cp, mv, rm, sort, cat, pg, lp, pr, file, find, more, cmp, diff, comm, head, tail, cut, grep, touch, tr, uniq **Security and Protection:** chmod, chown, chgrp, newgrp **Information:** learn, man, who, date, cal, tty, calendar, time, bc, whoami, which, hostname, history, wc		11	

	System Administrator:	su or root, date, fsck, init 2, wall, shutdown, mkfs, mount, unmount, dump, restor, tar, adduser, rmuser			
	Terminal:	echo, printf, clear			
	Process:	ps, kill, exec			
	I/O Redirection (<, >, >>), **Pipe** (|), *, gcc				
3.	1. Write a shell script which calculates nth Fibonacci number where n will be provided as input when prompted. 2. Write a shell script which takes one number from user and finds factorial of a given number. 3. Write a shell script to sort the number in ascending order. (Using array).			29	
4.	**Write programs using the following system calls of UNIX operating system: fork, exec, getpid, exit, wait, stat, readdir, opendir.** 1. Write a program to execute fork() and find out the process id by getpid() system call. 2. Write a program to execute following system call fork(), execl(), getpid(), exit(), wait() for a process. 3. Write a program to find out status of named file (program of working stat() system call). Write a program for “ls” command implementation using opendir() & readdir() system call.			33	
5.	Process control system calls: A. The demonstration of fork() B. execve() and wait() system calls along with zombie and orphan states.			37	
6.	Write a C program in UNIX to implement Process scheduling algorithms and compare. A. First Come First Serve (FCFS) Scheduling B. Shortest-Job-First (SJF) Scheduling C. Priority Scheduling (Non-preemption) after completion extend on Preemption. D. Round Robin(RR) Scheduling			41	

7.	Thread management using pthread library. Write a simple program to understand it.		49	
8.	Write a C program in UNIX to implement Bankers algorithm for Deadlock Avoidance.		50	
9.	Write a C program in UNIX to perform Memory allocation algorithms and calculate Internal and External Fragmentation. (First Fit, Best Fit, Worst Fit).		54	
10.	Thread synchronization using counting semaphores and mutual exclusion using mutex.		59	
11.	Write a C program in UNIX to implement inter process communication (IPC) using Semaphore.		61	
12.	Kernel space programming: Implement and add a loadable kernel module to Linux kernel, demonstrate using insmod, lsmod and rmmod commands. A sample kernel space program should print the "Hello World" while loading the kernel module and "Goodbye World" while unloading the kernel module.		64	

Practical: 1

AIM:

Working of Different Kernels:

A. UNIX Architecture

B. Types of OS- Linux, UNIX, MAC, Window etc.

C. Flavors of LINUX

Theory:

What is UNIX?

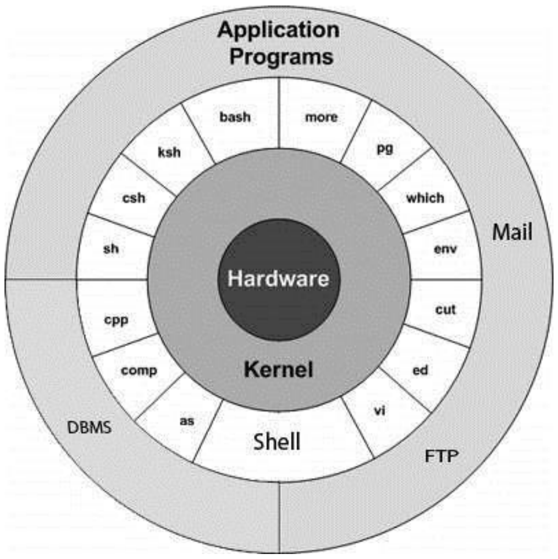
- The UNIX operating system is a set of programs that act as a link between the computer and the user.

- The computer programs that allocate the system resources and coordinate all the details of the computer's internals is called the **operating system** or the **kernel**.

- Users communicate with the kernel through a program known as the **shell**. The shell is a command line interpreter; it translates commands entered by the user and converts them into a language that is understood by the kernel.

UNIX Architecture:

Here is a basic block diagram of a UNIX system –



The main concept that unites all the versions of UNIX is the following four basics −

- **Kernel** − the kernel is the heart of the operating system. It interacts with the hardware and most of the tasks like memory management, task scheduling and file management.
- **Shell** − the shell is the utility that processes your requests. When you type in a command at your terminal, the shell interprets the command and calls the program that you want. The shell uses standard syntax for all commands. C Shell, Bourne Shell and

Korn Shell are the most famous shells which are available with most of the UNIX variants.

- **Commands and Utilities** − There are various commands and utilities which you can make use of in your day to day activities. **cp**, **mv**, **cat** and **grep**, etc. are few examples of commands and utilities. There are over 250 standard commands plus numerous others provided through 3rd party software. All the commands come along with various options.
- **Files and Directories** − All the data of UNIX is organized into files. All files are then organized into directories. These directories are further organized into a treelike structure called the **file system**.

Types of Operating Systems

An Operating_System performs all the basic tasks like managing file process, and memory. Thus operating system acts as manager of all the resources, i.e. **resource manager**. Thus operating system becomes an interface between user and machine.

Types of Operating Systems: Some of the widely used operating systems are follows-

1. Batch Operating System

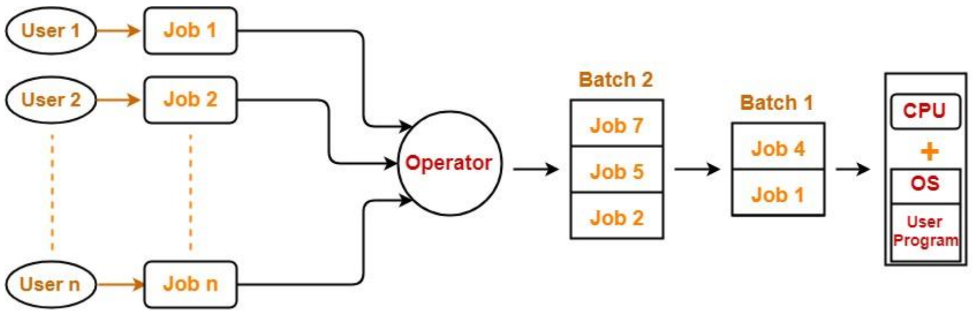
This type of operating system does not interact with the computer directly. There is an operator which takes similar jobs having same requirement and group them into batches. It is the responsibility of operator to sort the jobs with similar needs.
Examples of Batch based Operating System: Payroll System, Bank Statements

Advantages of Batch Operating System:

- It is very difficult to guess or know the time required by any job to Complete.
- Processors of the batch systems know how long the job would be when it is in queue.
- Multiple users can share the batch systems
- The idle time for batch system is very less
- It is easy to manage large work repeatedly in batch systems

Disadvantages of Batch Operating System:

- The computer operators should be well known with batch systems
- Batch systems are hard to debug
- It is sometime costly
- The other jobs will have to wait for an unknown time if any job fails



Batch Operating System

2. Time-Sharing Operating Systems / Multi-tasking

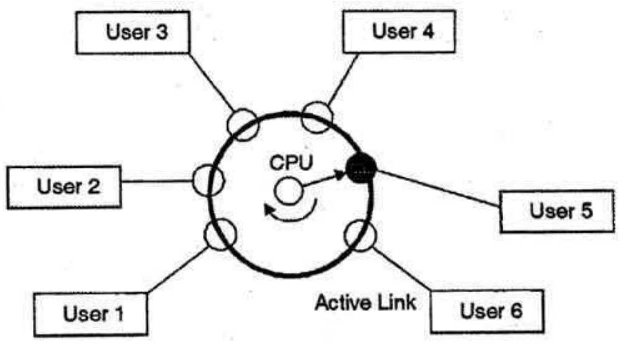
Each task is given some time to execute, so that all the tasks work smoothly. Each user gets time of CPU as they use single system. These systems are also known as Multitasking Systems. The task can be from single user or from different users also. The time that each task gets to execute is called quantum. After this time interval is over OS switches over to next task.

Advantages of Time-Sharing OS:

- Each task gets an equal opportunity
- Less chances of duplication of software
- CPU idle time can be reduced

Disadvantages of Time-Sharing OS:

- Reliability problem
- One must have to take care of security and integrity of user programs and data
- Data communication problem



3. Real time OS

A real time operating system time interval to process and respond to inputs is very small.
Examples: Military Software Systems, Space Software Systems.

Two types of Real-Time Operating System which are as follows:

- **Hard Real-Time Systems:**

These OSs are meant for the applications where time constraints are very strict and even the shortest possible delay is not acceptable. These systems are built for saving life like automatic parachutes or air bags which are required to be readily available in case of any accident. Virtual memory is almost never found in these systems.

- **Soft Real-Time Systems:**

These OSs are for applications where for time-constraint is less strict.

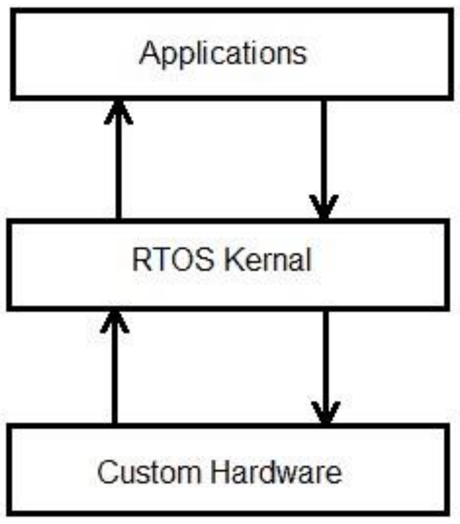
Advantages of RTOS:

Maximum Consumption: Maximum utilization of devices and system, thus more **OUTPUT** from all the resources

- **Task Shifting:** Time assigned for shifting tasks in these systems are very less. For example in older systems it takes about 10 micro seconds in shifting one task to another and in latest systems it takes 3 micro seconds.
- **Focus on Application:** Focus on running applications and less importance to applications which are in queue.
- **Real time operating system in embedded system:** Since size of programs are small, RTOS can also be used in embedded systems like in transport and others.
- **Error Free:** These types of systems are error free.
- **Memory Allocation:** Memory allocation is best managed in these type of systems.

Disadvantages of RTOS:

- **Limited Tasks:** Very few tasks run at the same time and their concentration is very less on few applications to avoid errors.
- **Use heavy system resources:** Sometimes the system resources are not so good and they are expensive as well.
- **Complex Algorithms:** The algorithms are very complex and difficult for the designer to write on.
- **Device driver and interrupt signals:** It needs specific device drivers and interrupt signals to response earliest to interrupts.
- **Thread Priority:** It is not good to set thread priority as these systems are very less prone to switching tasks.



4. Distributed Operating System

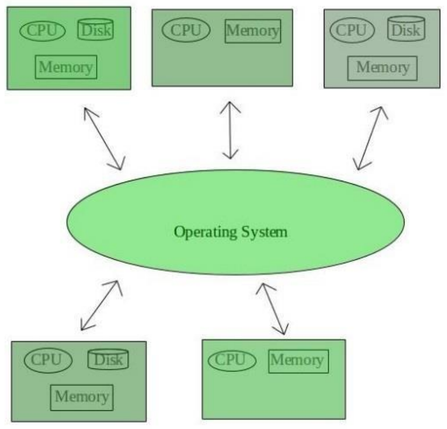
Distributed systems use many processors located in different machines to provide very fast computation to its users.

Advantages of Distributed Operating System:

- Failure of one will not affect the other network communication, as all systems are independent from each other
- Electronic mail increases the data exchange speed
- Since resources are being shared, computation is highly fast and durable
- Load on host computer reduces
- These systems are easily scalable as many systems can be easily added to the network
- Delay in data processing reduces

Disadvantages of Distributed Operating System:

- Failure of the main network will stop the entire communication
- To establish distributed systems the language which are used are not well defined yet
- These types of systems are not readily available as they are very expensive. Not only that underlying software is highly complex and not understood well yet



5. Network Operating System

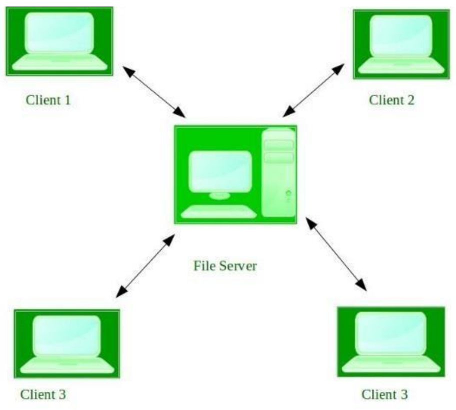
Network Operating System runs on a server. It provides the capability to serve to manage data, user, groups, security, application, and other networking functions.

Advantages of Network Operating System:

- Highly stable centralized servers
- Security concerns are handled through servers
- New technologies and hardware up-gradation are easily integrated to the system
- Server access are possible remotely from different locations and types of systems

Disadvantages of Network Operating System:

- Servers are costly
- User has to depend on central location for most operations
- Maintenance and updates are required regularly



6. Mobile OS

Mobile operating systems are those OS which is especially that are designed to power smartphones, tablets, and wearables devices.
Some most famous mobile operating systems are Android and iOS, but others include BlackBerry, Web, and watchOS.

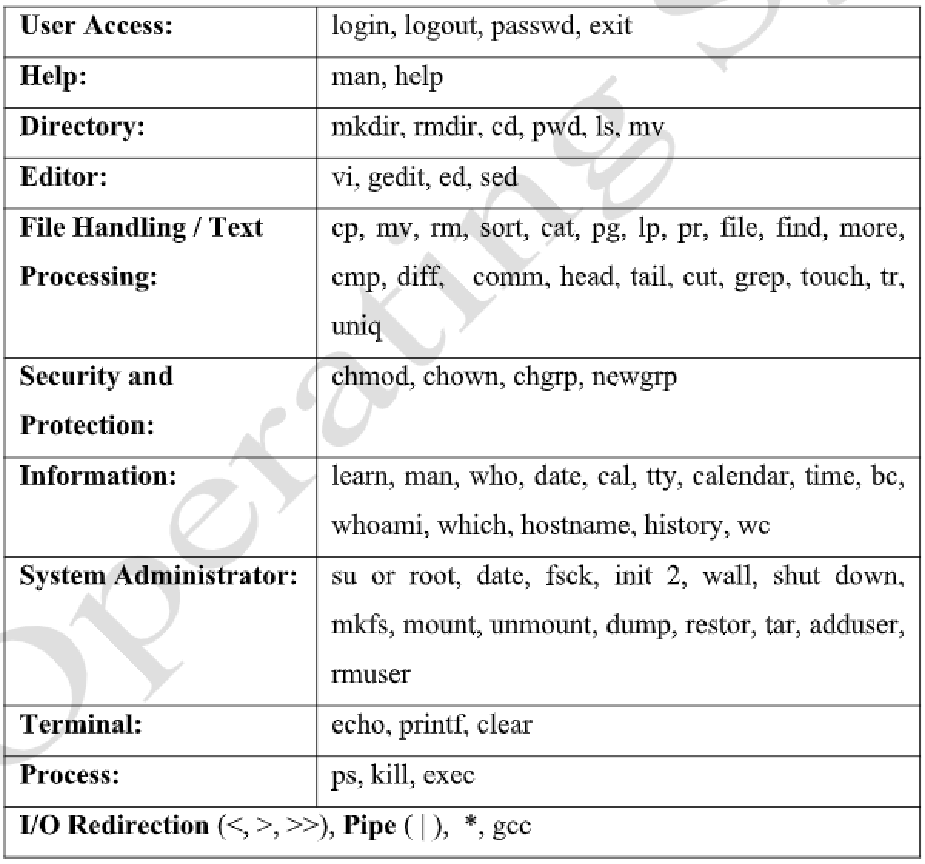
Flavors of LINUX:-

One of the most confusing things for the newcomer to Linux is how many distributions, or versions, of the operating system there are. Ubuntu is the one most people have heard of, but there are hundreds of others as well, each offering some variant on the basic Linux theme.

1. Ubuntu
2. Fedora
3. Linux Mint
4. openSUSE
5. PCLinuxOS
6. Debian
7. Mandriva
8. Sabayon/Gentoo
9. Arch Linux
10. Puppy Linux

Practical: 2

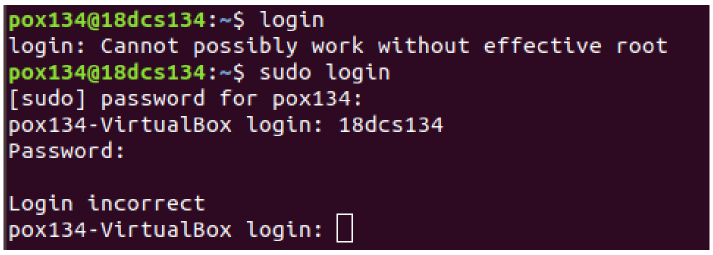
<u>AIM:</u> Study of UNIX Architecture and the following UNIX commands with option:



User Access:	login, logout, passwd, exit
Help:	man, help
Directory:	mkdir, rmdir, cd, pwd, ls, mv
Editor:	vi, gedit, ed, sed
File Handling / Text Processing:	cp, mv, rm, sort, cat, pg, lp, pr, file, find, more, cmp, diff, comm, head, tail, cut, grep, touch, tr, uniq
Security and Protection:	chmod, chown, chgrp, newgrp
Information:	learn, man, who, date, cal, tty, calendar, time, bc, whoami, which, hostname, history, wc
System Administrator:	su or root, date, fsck, init 2, wall, shut down, mkfs, mount, unmount, dump, restor, tar, adduser, rmuser
Terminal:	echo, printf, clear
Process:	ps, kill, exec
I/O Redirection (<, >, >>), **Pipe** (|), *, gcc	

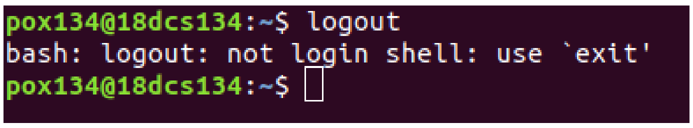
<u>User Access Commands:</u>

login: This command begin session on the system. The login program is used to establish a new session with the system. It is normally automatically by responding to the login: prompt on the user's terminal.



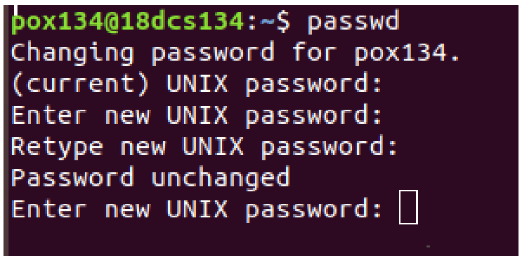
```
pox134@18dcs134:~$ login
login: Cannot possibly work without effective root
pox134@18dcs134:~$ sudo login
[sudo] password for pox134:
pox134-VirtualBox login: 18dcs134
Password:

Login incorrect
pox134-VirtualBox login: 
```

logout: logout command is used to exit from the system.



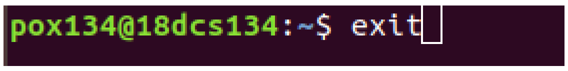
```
pox134@18dcs134:~$ logout
bash: logout: not login shell: use `exit'
pox134@18dcs134:~$ 
```

passwd: This command is used for change password for user.



```
pox134@18dcs134:~$ passwd
Changing password for pox134.
(current) UNIX password:
Enter new UNIX password:
Retype new UNIX password:
Password unchanged
Enter new UNIX password:
```

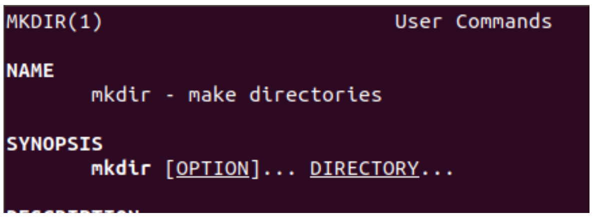
exit: This command is used to exit from process.



```
pox134@18dcs134:~$ exit
```

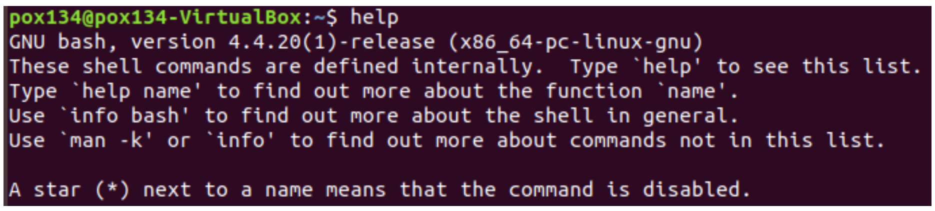
User Help command:

man: man command is used for retrieve manual information about any command.



```
MKDIR(1)                          User Commands

NAME
       mkdir - make directories

SYNOPSIS
       mkdir [OPTION]... DIRECTORY...
```

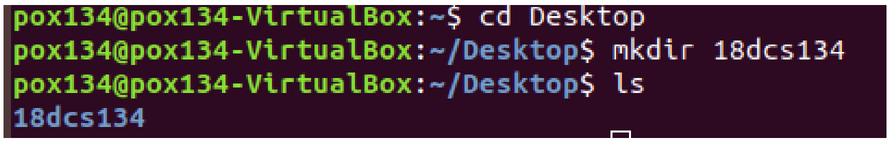
help: help command is used for get detail of any command.



```
pox134@pox134-VirtualBox:~$ help
GNU bash, version 4.4.20(1)-release (x86_64-pc-linux-gnu)
These shell commands are defined internally.  Type `help' to see this list.
Type `help name' to find out more about the function `name'.
Use `info bash' to find out more about the shell in general.
Use `man -k' or `info' to find out more about commands not in this list.

A star (*) next to a name means that the command is disabled.
```

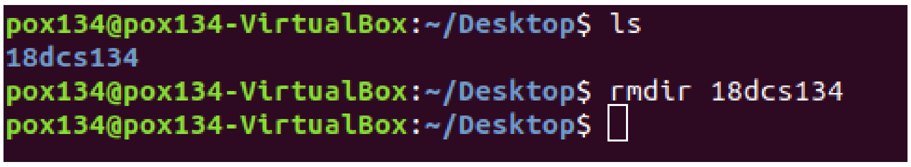
Directory command:

mkdir: This command is used for create directory in particular location or present location.



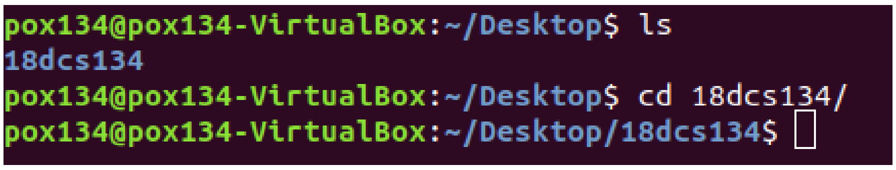
```
pox134@pox134-VirtualBox:~$ cd Desktop
pox134@pox134-VirtualBox:~/Desktop$ mkdir 18dcs134
pox134@pox134-VirtualBox:~/Desktop$ ls
18dcs134
```

rmdir: This command is used to remove empty directory.



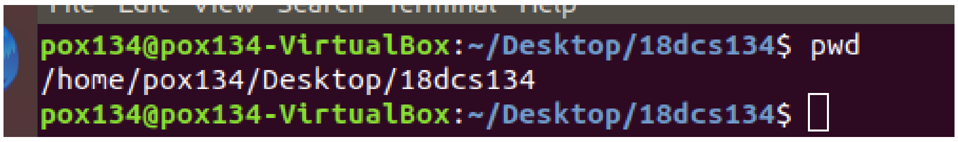
```
pox134@pox134-VirtualBox:~/Desktop$ ls
18dcs134
pox134@pox134-VirtualBox:~/Desktop$ rmdir 18dcs134
pox134@pox134-VirtualBox:~/Desktop$
```

cd: cd command is used for change directory.



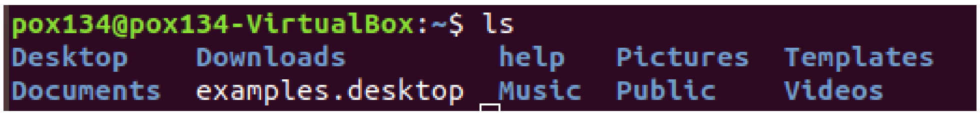
```
pox134@pox134-VirtualBox:~/Desktop$ ls
18dcs134
pox134@pox134-VirtualBox:~/Desktop$ cd 18dcs134/
pox134@pox134-VirtualBox:~/Desktop/18dcs134$
```

pwd: This command print name o f current working directory.



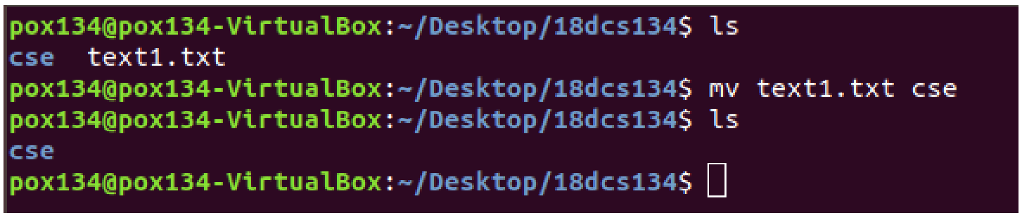
```
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ pwd
/home/pox134/Desktop/18dcs134
pox134@pox134-VirtualBox:~/Desktop/18dcs134$
```

ls: This command print list of directory in present directory.



```
pox134@pox134-VirtualBox:~$ ls
Desktop     Downloads         help   Pictures   Templates
Documents   examples.desktop  Music  Public     Videos
```

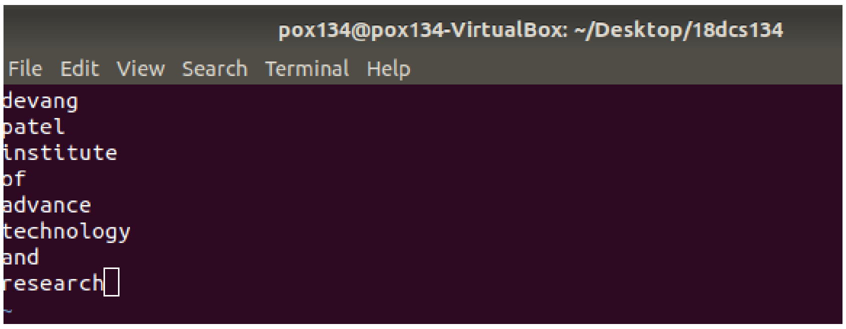
mv:mv command move files or rename file.



```
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ ls
cse  text1.txt
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ mv text1.txt cse
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ ls
cse
pox134@pox134-VirtualBox:~/Desktop/18dcs134$
```

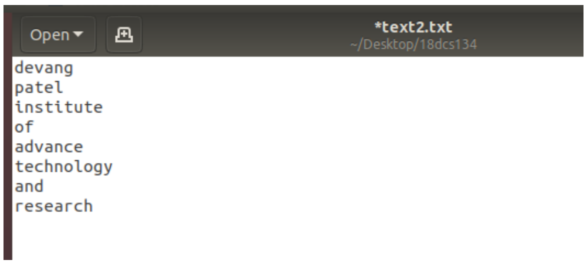
Editor command:

vi: vi is open text editor using command name and file name.


pox134@pox134-VirtualBox: ~/Desktop/18dcs134
File Edit View Search Terminal Help

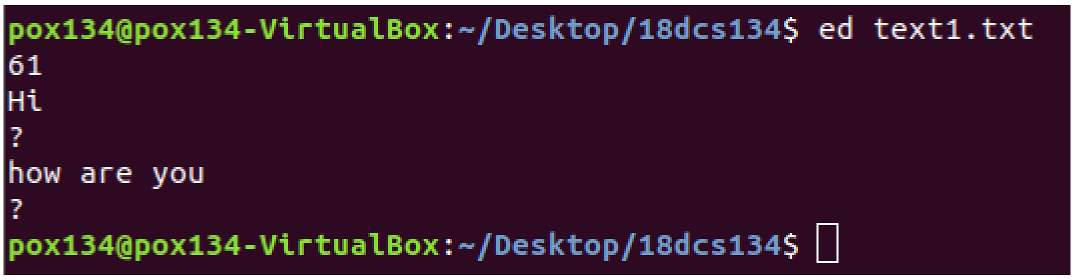
```
devang
patel
institute
of
advance
technology
and
research
~
```

gedit:gedit command is used to create and open gedit and it is text editor.



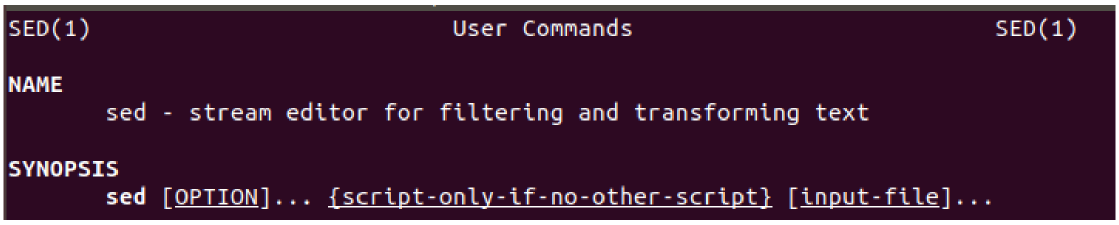
```
devang
patel
institute
of
advance
technology
and
research
```

ed: ed is line oriented text editor.



```
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ ed text1.txt
61
Hi
?
how are you
?
pox134@pox134-VirtualBox:~/Desktop/18dcs134$
```

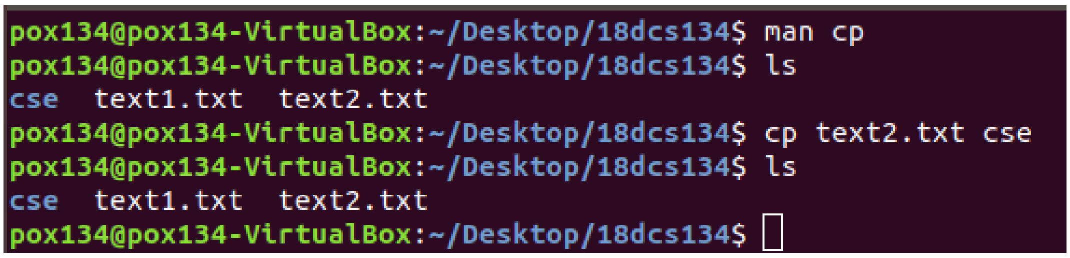
sed: stream editor for filtering and transforming text.



```
SED(1)                      User Commands                     SED(1)

NAME
       sed - stream editor for filtering and transforming text

SYNOPSIS
       sed [OPTION]... {script-only-if-no-other-script} [input-file]...
```

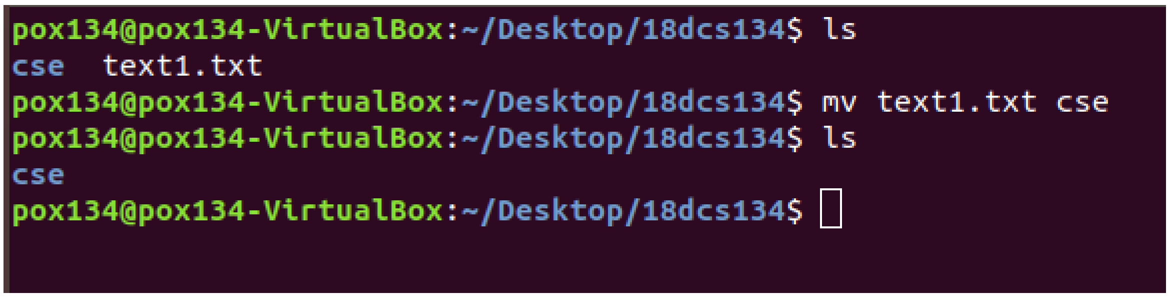
File Handling Commands:

cp: used to copy files and directories



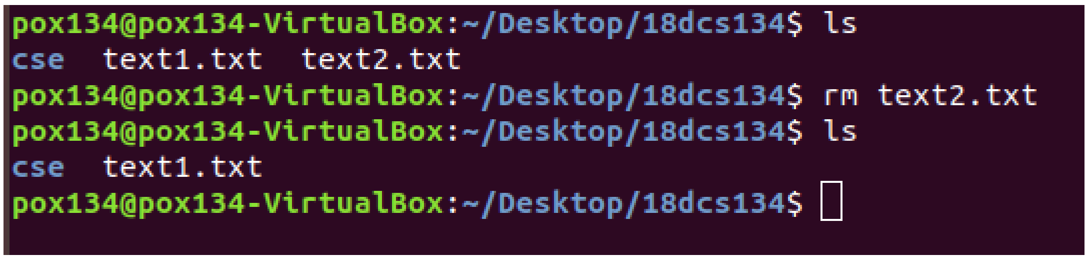
```
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ man cp
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ ls
cse  text1.txt  text2.txt
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ cp text2.txt cse
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ ls
cse  text1.txt  text2.txt
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ 
```

mv: This ommand moved file or directory to one location to other location and also used for rename.



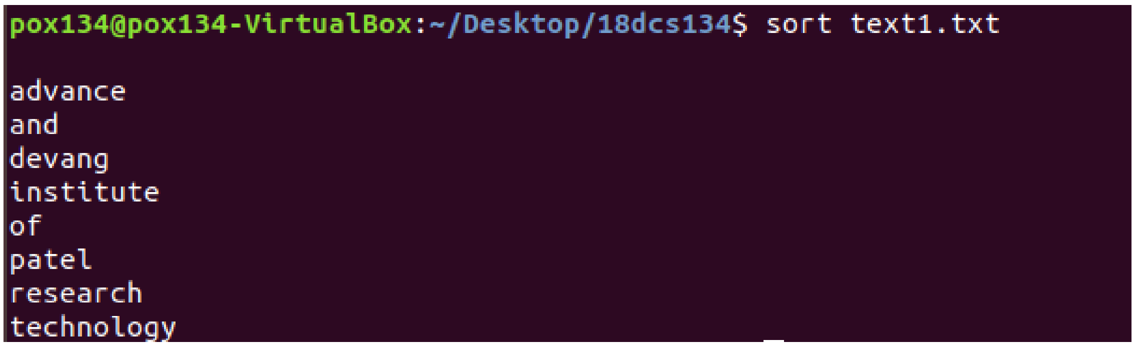
```
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ ls
cse  text1.txt
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ mv text1.txt cse
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ ls
cse
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ 
```

rm: removes files or directories.rm removes each specific file.By default, it does not remove directories.



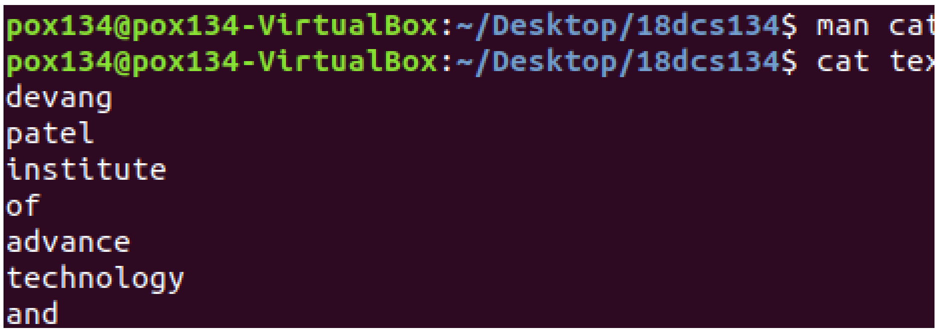
```
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ ls
cse  text1.txt  text2.txt
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ rm text2.txt
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ ls
cse  text1.txt
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ 
```

sort: sort command sort lines of text files and show on terminal.



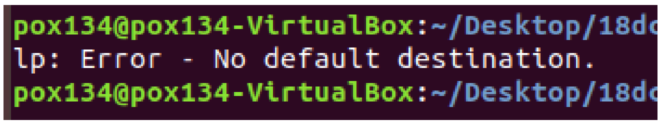
```
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ sort text1.txt

advance
and
devang
institute
of
patel
research
technology
```

cat: this command is used to concatenate files and print on the standard **OUTPUT**.



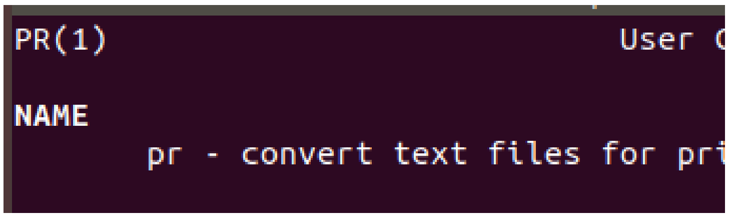
```
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ man cat
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ cat tex
devang
patel
institute
of
advance
technology
and
```

lp: This command is print files



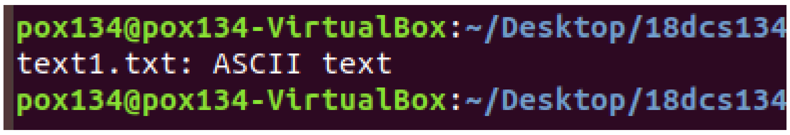
```
pox134@pox134-VirtualBox:~/Desktop/18dc
lp: Error - No default destination.
pox134@pox134-VirtualBox:~/Desktop/18dc
```

pr: This convert text files for printing.



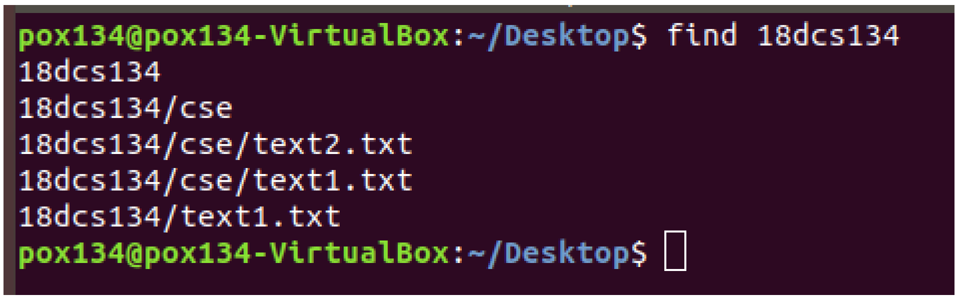
```
PR(1)                              User C

NAME
       pr - convert text files for pri
```

file: This command determine file type.



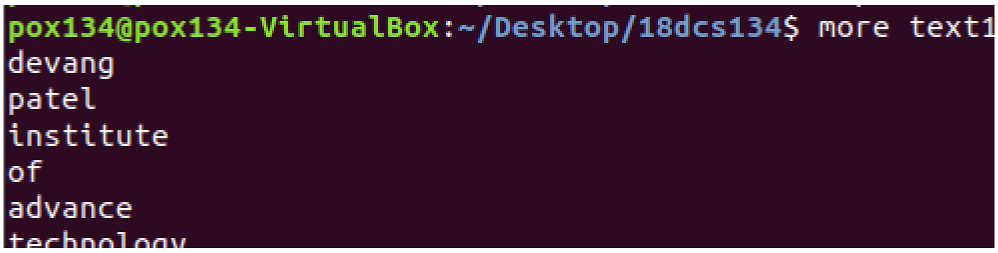
```
pox134@pox134-VirtualBox:~/Desktop/18dcs134
text1.txt: ASCII text
pox134@pox134-VirtualBox:~/Desktop/18dcs134
```

find: This command search for files in a directory hierarchy.



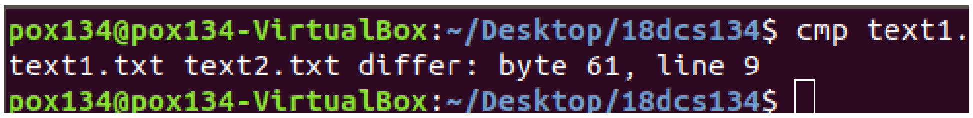
```
pox134@pox134-VirtualBox:~/Desktop$ find 18dcs134
18dcs134
18dcs134/cse
18dcs134/cse/text2.txt
18dcs134/cse/text1.txt
18dcs134/text1.txt
pox134@pox134-VirtualBox:~/Desktop$
```

more: This command used for file perusal filter for crt viewing.more is a filter for paging through text one screenful at a time.



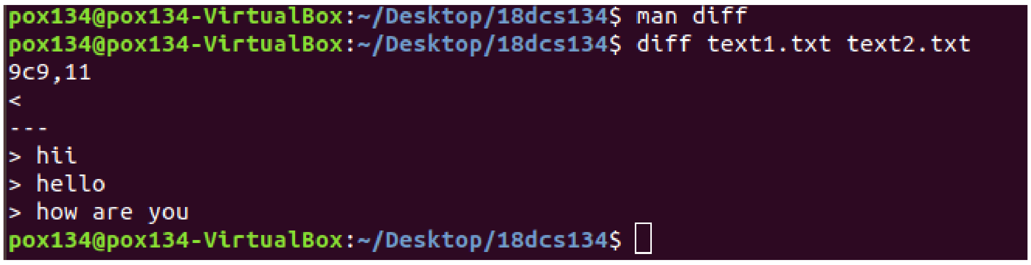
```
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ more text1
devang
patel
institute
of
advance
technology
```

cmp: compare two files byte by byte.if both file same no any **OUTPUT** display on terminal.



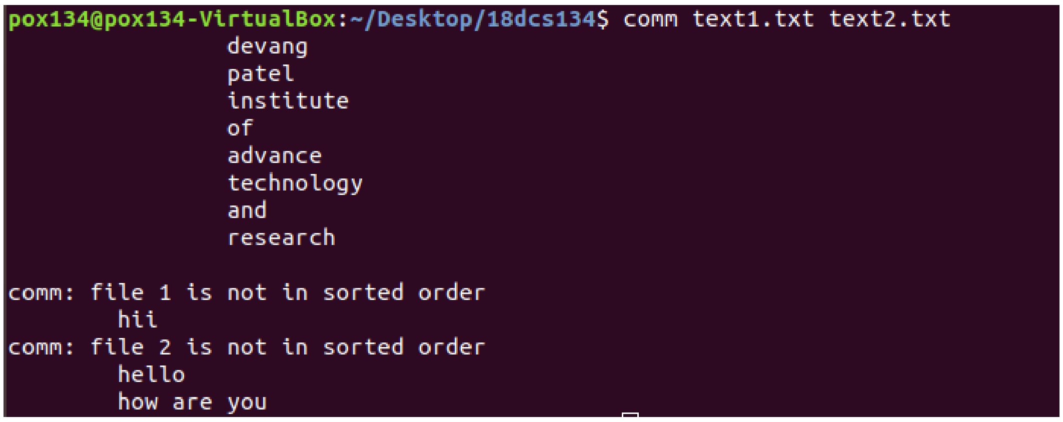
```
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ cmp text1.
text1.txt text2.txt differ: byte 61, line 9
pox134@pox134-VirtualBox:~/Desktop/18dcs134$
```

diff: compare two file line by line.



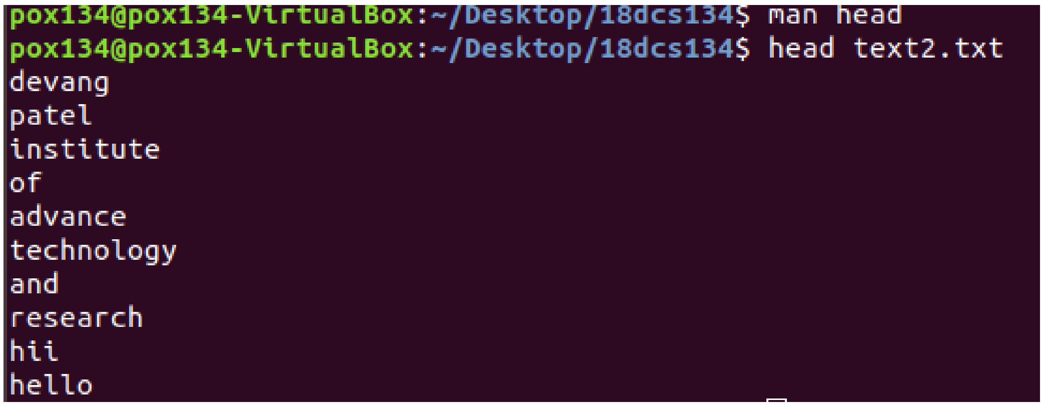
```
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ man diff
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ diff text1.txt text2.txt
9c9,11
<
---
> hii
> hello
> how are you
pox134@pox134-VirtualBox:~/Desktop/18dcs134$
```

comm: compare two sorted files line by line.



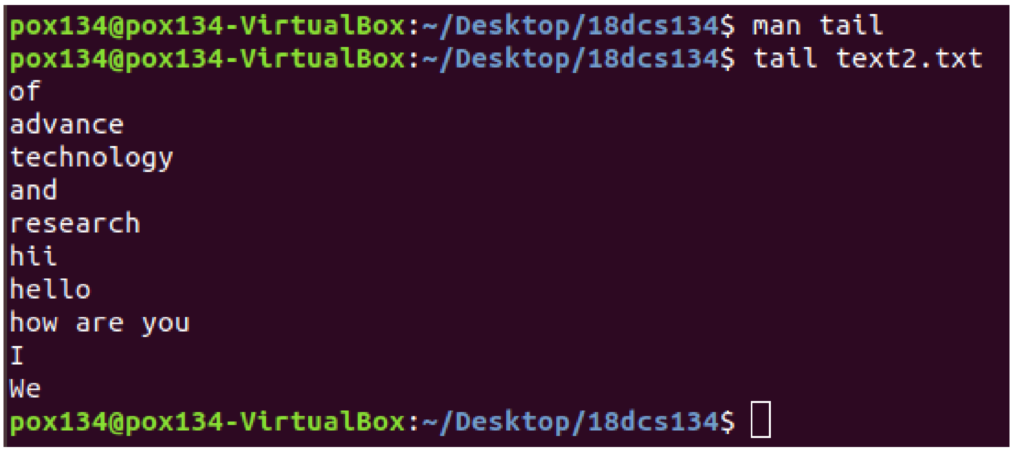
```
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ comm text1.txt text2.txt
                devang
                patel
                institute
                of
                advance
                technology
                and
                research

comm: file 1 is not in sorted order
        hii
comm: file 2 is not in sorted order
        hello
        how are you
```

head: This command is used to print first part of files. By default print first 10 lines if we want print specific no. lines we use –n where n is no. of line.



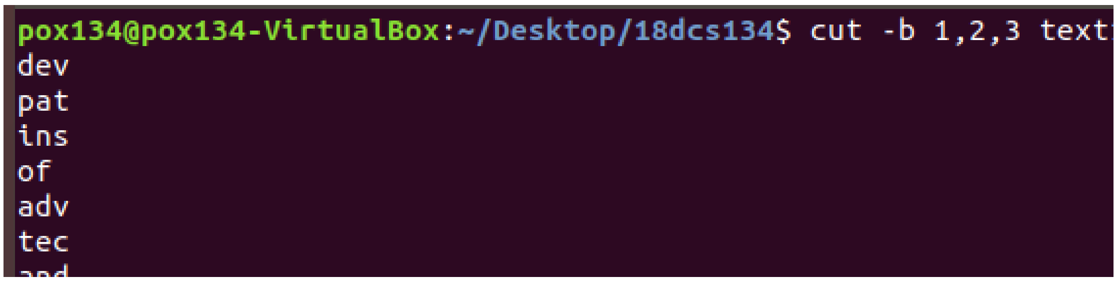
```
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ man head
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ head text2.txt
devang
patel
institute
of
advance
technology
and
research
hii
hello
```

tail: This command prints last part of file. By default print first 10 lines if we want print specific no. lines we use –n where n is no. of line.



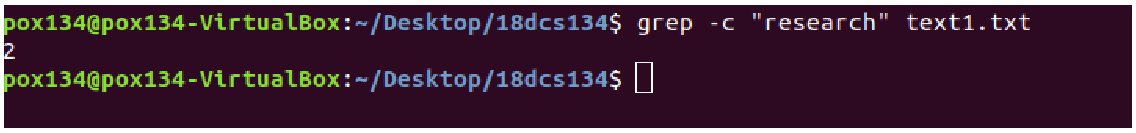
```
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ man tail
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ tail text2.txt
of
advance
technology
and
research
hii
hello
how are you
I
We
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ 
```

cut: This command rermove section from each line of files.



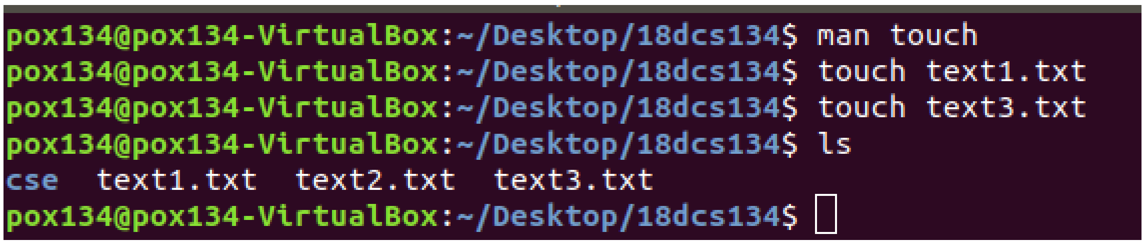
```
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ cut -b 1,2,3 text
dev
pat
ins
of
adv
tec
```

grep: grep search pattern in each file.



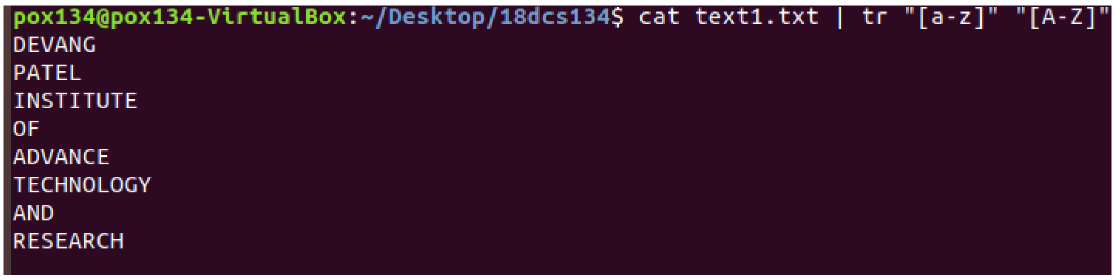
```
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ grep -c "research" text1.txt
2
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ 
```

touch: This command is used for change file timestamp and if file not exist create it.



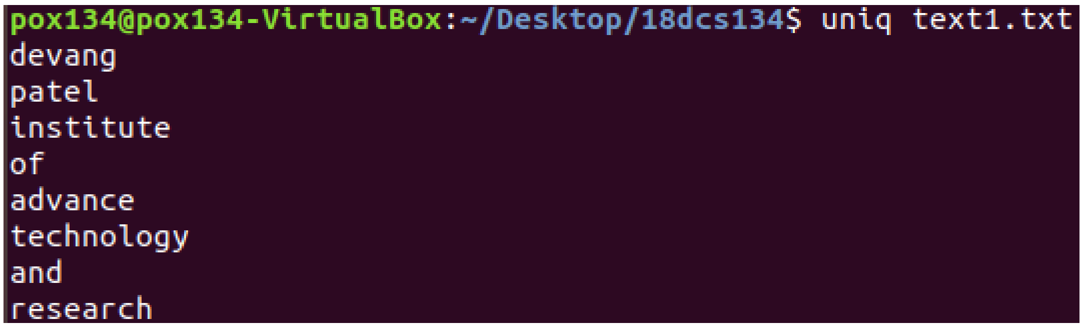
```
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ man touch
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ touch text1.txt
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ touch text3.txt
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ ls
cse  text1.txt  text2.txt  text3.txt
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ 
```

tr: This command translate or delete characters.Using –s replace each sequence of a repeated character that is listed in the last specified SET,with a single occurrence of that character.



```
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ cat text1.txt | tr "[a-z]" "[A-Z]"
DEVANG
PATEL
INSTITUTE
OF
ADVANCE
TECHNOLOGY
AND
RESEARCH
```

uniq: This commans report or omit repeated lines.Filter adjacent matching lines from input (or standard input), writing to **OUTPUT** (or standard **OUTPUT**).



```
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ uniq text1.txt
devang
patel
institute
of
advance
technology
and
research
```

Security and Protection:

chmod: This command is used to change the access mode of a file.The name is an abbreviation of change mode.

Syntax: chmod [reference][operator][mode] file…

Mode:

r-Permission to read the file.

w-Permission to write the file

x-Permission to execute the file or directory

operator:

+ - add specific modes to the specified classes

- - removes the specified modes from the specified classes

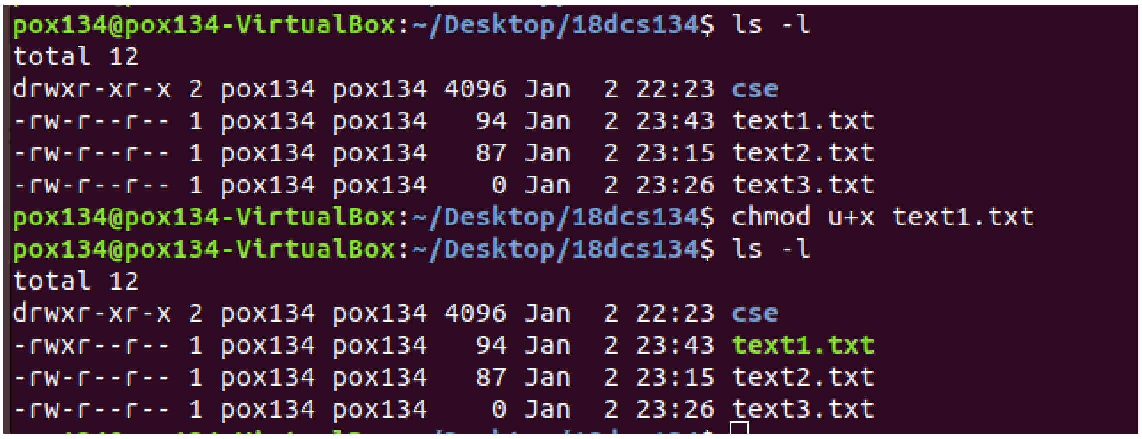
= - the mode specified are to be made the exact modes for the specified classes

Refrences:
u – owner

g - group

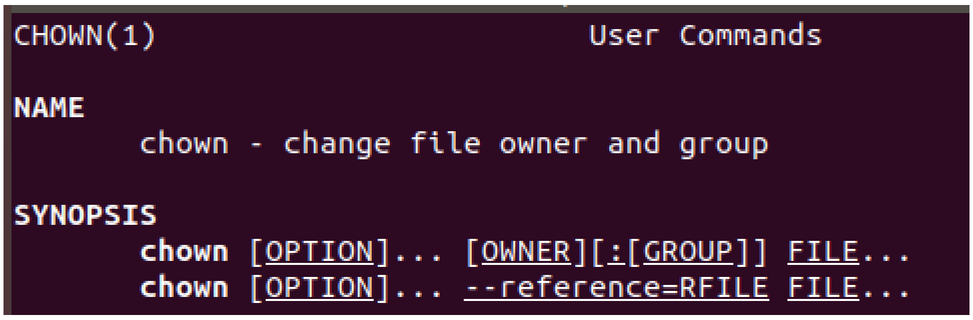
o – others

a – all



```
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ ls -l
total 12
drwxr-xr-x 2 pox134 pox134 4096 Jan  2 22:23 cse
-rw-r--r-- 1 pox134 pox134   94 Jan  2 23:43 text1.txt
-rw-r--r-- 1 pox134 pox134   87 Jan  2 23:15 text2.txt
-rw-r--r-- 1 pox134 pox134    0 Jan  2 23:26 text3.txt
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ chmod u+x text1.txt
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ ls -l
total 12
drwxr-xr-x 2 pox134 pox134 4096 Jan  2 22:23 cse
-rwxr--r-- 1 pox134 pox134   94 Jan  2 23:43 text1.txt
-rw-r--r-- 1 pox134 pox134   87 Jan  2 23:15 text2.txt
-rw-r--r-- 1 pox134 pox134    0 Jan  2 23:26 text3.txt
```

chown: This command changes user and/or group ownership of each given file.If only an owner is given, that user is made the owner of each given file and the files group is not changed.



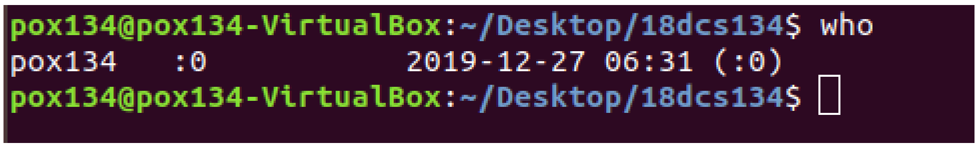
```
CHOWN(1)                         User Commands

NAME
       chown - change file owner and group

SYNOPSIS
       chown [OPTION]... [OWNER][:[GROUP]] FILE...
       chown [OPTION]... --reference=RFILE FILE...
```

Information:

who: This command is used to get information about currently logged in user on to system.Also find out 1) Time of last system boot

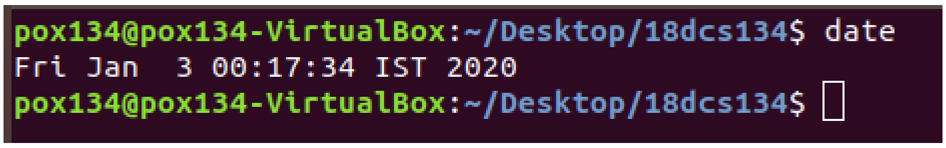
2) Current run level of the system

3) List of logged in users and more



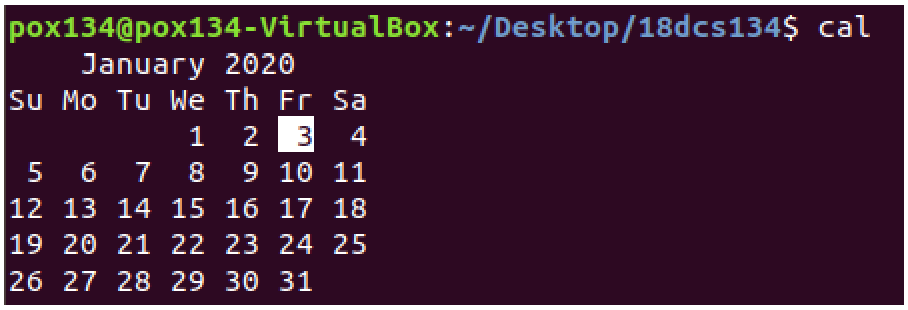
```
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ who
pox134   :0           2019-12-27 06:31 (:0)
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ 
```

date: Used to display the system date and time.It is also used to set date and time of the system.By default the date command displays the date in the time zone on which linux operating system is configured.You must be the super user to change the date and time.



```
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ date
Fri Jan  3 00:17:34 IST 2020
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ 
```

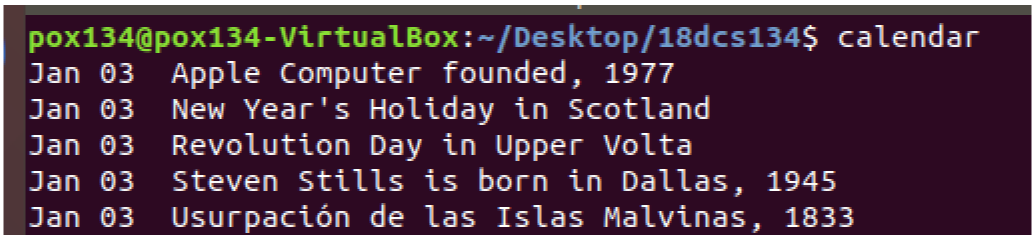
cal: This command is used to display quick view of calendar in terminal and also used to see the calendar of a specific month or a whole year.



```
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ cal
    January 2020
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
```

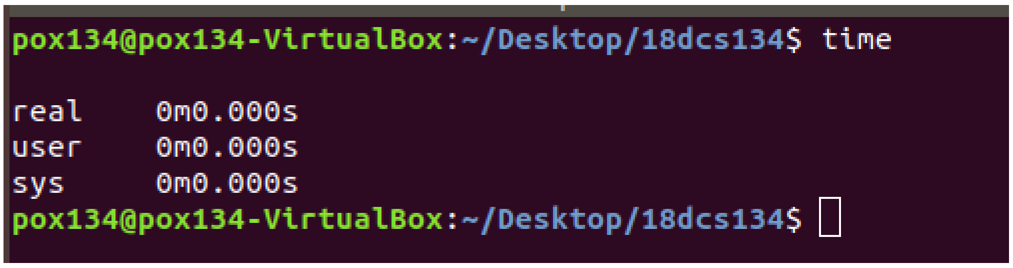
tty: tty which displays information related to terminal. The tty command of terminal basically prints the file name of the terminal connected to standard input. tty is short of teletype, but popularly known as a terminal it allows you to interact with the system by passing on the data (you input) to the system, and displaying the **OUTPUT** produced by the system.

calendar: This command is used to reminder service.The calendar utility the current directory or the directory specified by the CALENDAR_DIR environment variable for a file named calendar and displays lines that begin with either today’s date or tomorrow’s.



```
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ calendar
Jan 03  Apple Computer founded, 1977
Jan 03  New Year's Holiday in Scotland
Jan 03  Revolution Day in Upper Volta
Jan 03  Steven Stills is born in Dallas, 1945
Jan 03  Usurpación de las Islas Malvinas, 1833
```

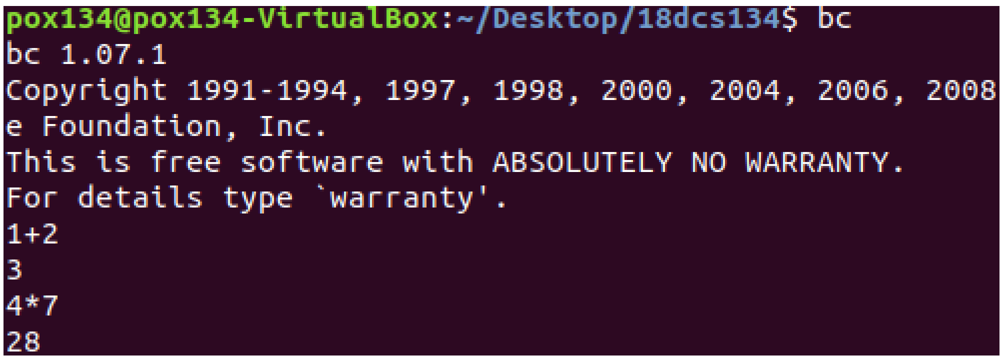
time: This command run the program COMMAND with any given arguments ARG..When COMMAND finishes,time displays information about resources used by COMMAND.



```
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ time

real	0m0.000s
user	0m0.000s
sys	0m0.000s
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ 
```

bc: bc command is used for command line calculator. It is similar to basic calculator by using which we can do basic mathematical calculations.

Arithmetic operations are the most basic in any kind of programming language. Linux or UNIX operating system provides the bc command and expr command for doing arithmetic calculations.

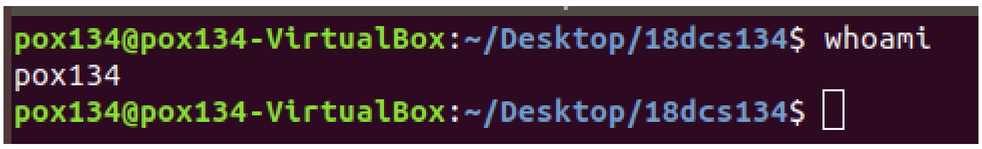


```
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ bc
bc 1.07.1
Copyright 1991-1994, 1997, 1998, 2000, 2004, 2006, 2008
e Foundation, Inc.
This is free software with ABSOLUTELY NO WARRANTY.
For details type `warranty'.
1+2
3
4*7
28
```

whoami: It is basically the concatenation of the strings “who”,”am”,”i” as whoami.

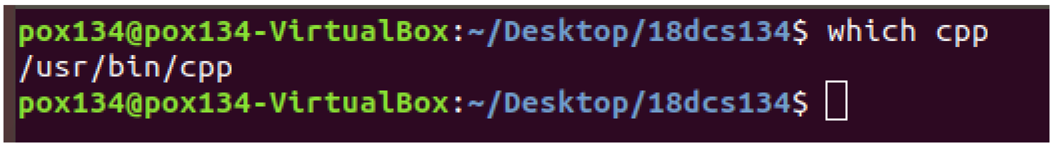
It displays the username of the current user when this command is invoked.

It is similar as running the id command with the options -un.



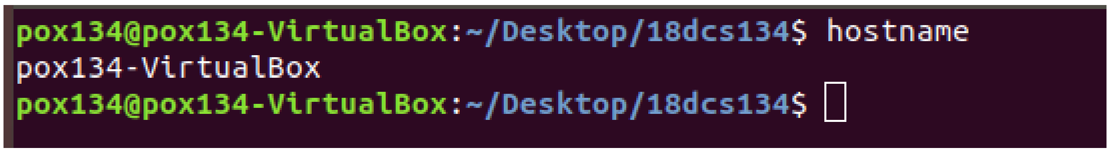
```
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ whoami
pox134
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ 
```

which: which command in Linux is a command which is used to locate the executable file associated with the given command by searching it in the path environment variable.



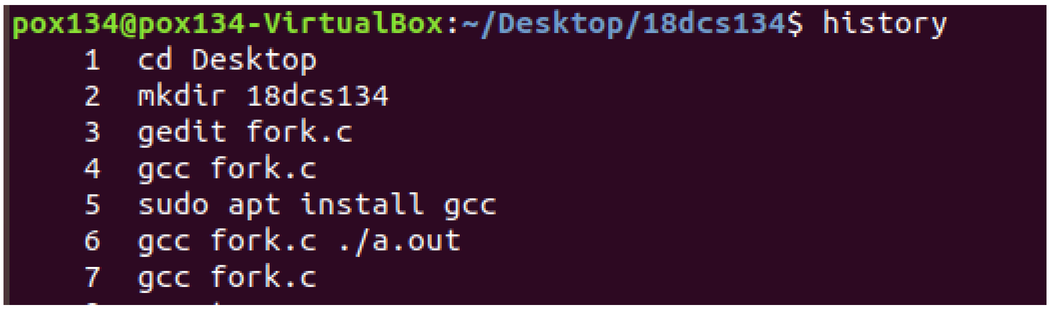
```
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ which cpp
/usr/bin/cpp
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ 
```

hostname: hostname command in Linux is used to obtain the DNS(Domain Name System) name and set the system's hostname or NIS(Network Information System) domain name. A hostname is a name which is given to a computer and it attached to the network. Its main purpose is to uniquely identify over a network.



```
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ hostname
pox134-VirtualBox
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ 
```

history: history command is used to view the previously executed command. This feature was not available in the Bourne shell. Bash and Korn support this feature in which every command executed is treated as the event and is associated with an event number using which they can be recalled and changed if required.

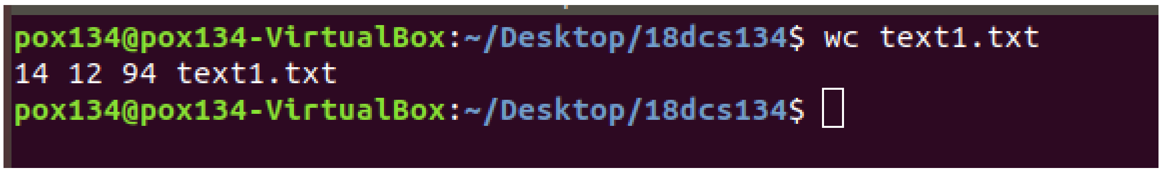


```
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ history
    1  cd Desktop
    2  mkdir 18dcs134
    3  gedit fork.c
    4  gcc fork.c
    5  sudo apt install gcc
    6  gcc fork.c ./a.out
    7  gcc fork.c
```

wc: wc stands for word count. As the name implies, it is mainly used for counting purpose.

It is used to find out number of lines, word count, byte and characters count in the files specified in the file arguments.

By default it displays four-columnar **OUTPUT**.



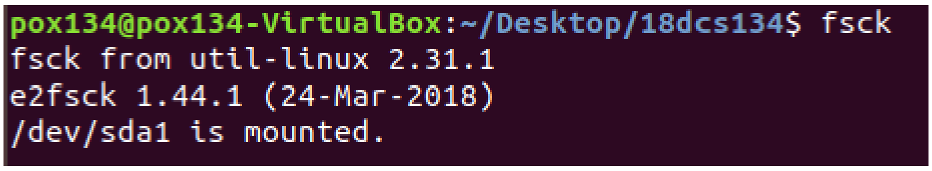
```
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ wc text1.txt
14 12 94 text1.txt
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ 
```

<u>System Administrator commands:</u>

fsck: File System Consistency Checker (FSCK)

File system inconsistency is a major issue in operating systems. FSCK is one of the standard solutions adopted.

FSCK is one approach still used by older Linux-based systems to find and repair inconsistencies. It is not a complete solution and may still have inodes pointing to garbage data. The major focus is to make the metadata internally consistent.



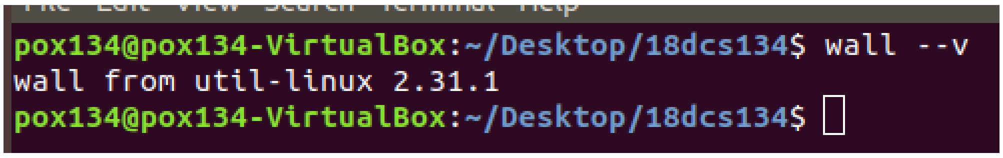
```
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ fsck
fsck from util-linux 2.31.1
e2fsck 1.44.1 (24-Mar-2018)
/dev/sda1 is mounted.
```

init 2: There are basically 8 runlevels in UNIX. I will briefly tell some thing about the different init levels and their use.

Run Level: At any given time, the system is in one of eight possible run levels.

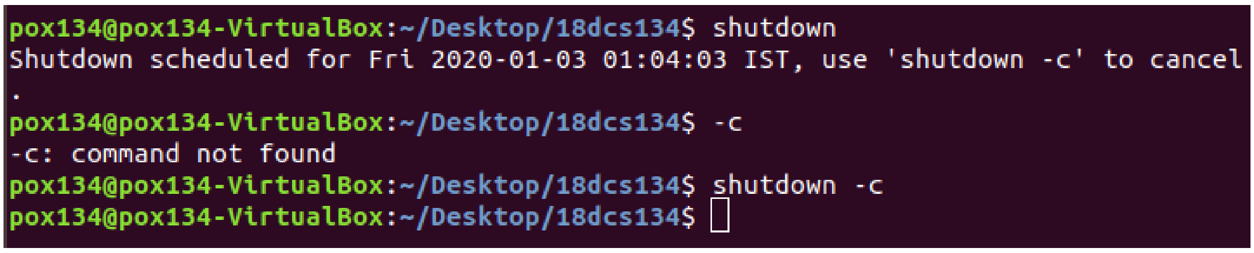
A run level is a software configuration under which only a selected group of processes exists.
init 2 : No network but multitasking support is present .

wall: wall command in Linux system is used to write a message to all users. This command displays a message, or the contents of a file, or otherwise its standard input, on the terminals of all currently logged in users. The lines which will be longer than 79 characters, wrapped by this command.



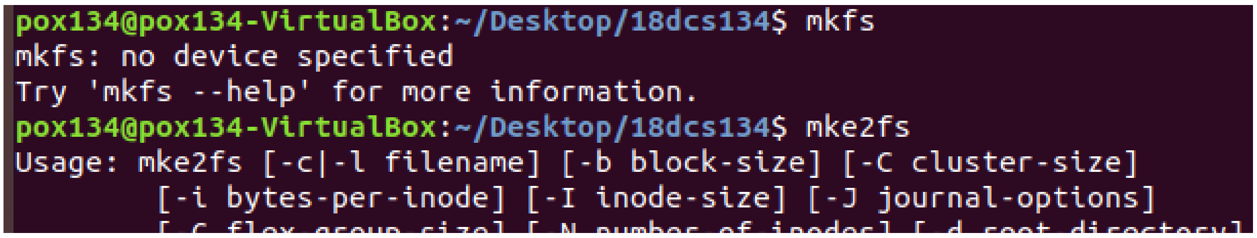
```
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ wall --v
wall from util-linux 2.31.1
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ 
```

shutdown: The shutdown command in Linux is used to shutdown the system in a safe way. You can shutdown the machine immediately, or schedule a shutdown using 24 hour format.It brings the system down in a secure way. When the shutdown is initiated, all logged-in users and processes are notified that the system is going down, and no further logins are allowed.



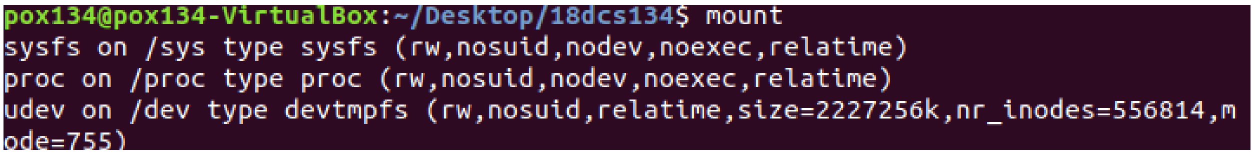
```
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ shutdown
Shutdown scheduled for Fri 2020-01-03 01:04:03 IST, use 'shutdown -c' to cancel
.
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ -c
-c: command not found
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ shutdown -c
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ 
```

mkfs: This command is used to buid a linux file system on a device, usally a hard disk partition.



```
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ mkfs
mkfs: no device specified
Try 'mkfs --help' for more information.
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ mke2fs
Usage: mke2fs [-c|-l filename] [-b block-size] [-C cluster-size]
        [-i bytes-per-inode] [-I inode-size] [-J journal-options]
        [-G flex-group-size] [-N number-of-inodes] [-d root-directory]
```

mount: mount command is used to mount the filesystem found on a device to big tree structure(Linux filesystem) rooted at '/'. Conversely, another command umount can be used to detach these devices from the Tree.



```
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ mount
sysfs on /sys type sysfs (rw,nosuid,nodev,noexec,relatime)
proc on /proc type proc (rw,nosuid,nodev,noexec,relatime)
udev on /dev type devtmpfs (rw,nosuid,relatime,size=2227256k,nr_inodes=556814,m
ode=755)
```

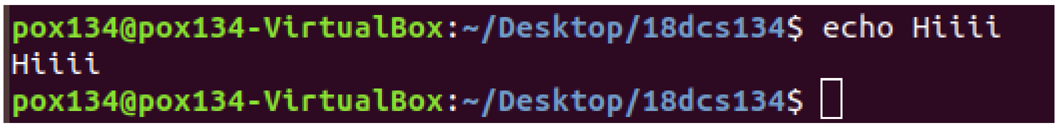
dump: dump command in Linux is used for backup the filesystem to some storage device. It backs up the complete file system and not the individual files. In other words, it backups the required files to tape, disk or any other storage device for safe storage.

restore: restore command in Linux system is used for restoring files from a backup created using dump. The restore command performs the exact inverse function of dump. A full backup of a file system is being restored and subsequent incremental backups layered is being kept on top of it. Single files and directory subtrees can easily be restored from full or partial backups. Restore simply works across a network. Other arguments which need to be pass to the command are file or directory names specifying the files that need to be restored.

tar: The Linux 'tar' stands for tape archive, is used to create Archive and extract the Archive files. tar command in Linux is one of the important command which provides archiving functionality in Linux. We can use Linux tar command to create compressed or uncompressed Archive files and also maintain and modify them.

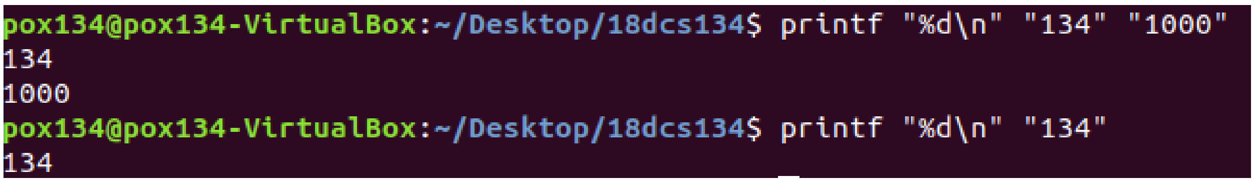
Terminal:

echo: echo command in linux is used to display line of text/string that are passed as an argument . This is a built in command that is mostly used in shell scripts and batch files to output status text to the screen or a file.



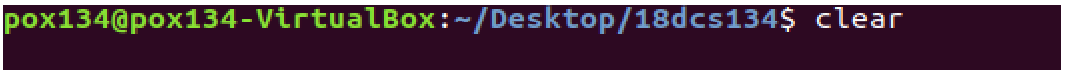
```
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ echo Hiiii
Hiiii
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ 
```

prinf: “printf” command in Linux is used to display the given string, number or any other format specifier on the terminal window. It works the same way as “printf” works in programming languages like C.



```
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ printf "%d\n" "134" "1000"
134
1000
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ printf "%d\n" "134"
134
```

clear: This command is used to clear terminal screen.



```
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ clear
```

Process:

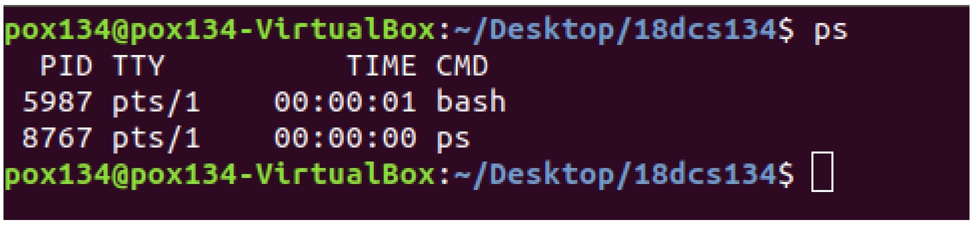
ps: This commad give this four information about process,

PID – the unique process ID

TTY – terminal type that the user is logged into

TIME – amount of CPU in minutes and seconds that the process has been running

CMD – name of the command that launched the process



```
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ ps
  PID TTY          TIME CMD
 5987 pts/1    00:00:01 bash
 8767 pts/1    00:00:00 ps
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ 
```

kill: kill command in Linux (located in /bin/kill), is a built-in command which is used to terminate processes manually. kill command sends a signal to a process which terminates the process.

Exec: exec command in Linux is used to execute a command from the bash itself. This command does not create a new process it just replaces the bash with the command to be executed. If the exec command is successful, it does not return to the calling process.

Practical: 3

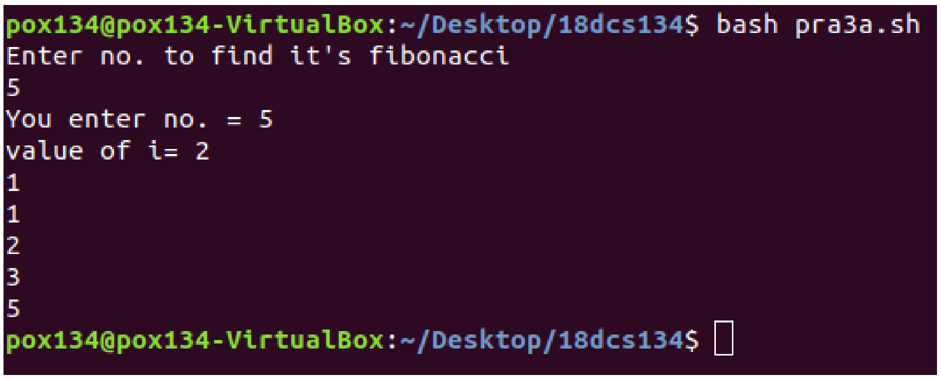
AIM:

1. **Write a shell script which calculates n th Fibonacci number where n will be provided as input when prompted.**
2. **Write a shell script which takes one number from user and finds factorial of a given number.**
3. **Write a shell script to sort the number in ascending order. (Using array).**

1) CODE:

```
#Fibonacci praogram

echo "Enter no. to find it's fibonacci"

read num

echo "You enter no. = $num"

i1=1
i2=1
i3=$((i1+i2))

echo "value of i=" $i3

echo "$i1"
echo "$i2"

for (( i=0; i<num-2; i++ ))
do
	echo "$i3"
	temp=$((i2+i3))
	i2=$i3
	i3=$temp
done
```

OUTPUT:

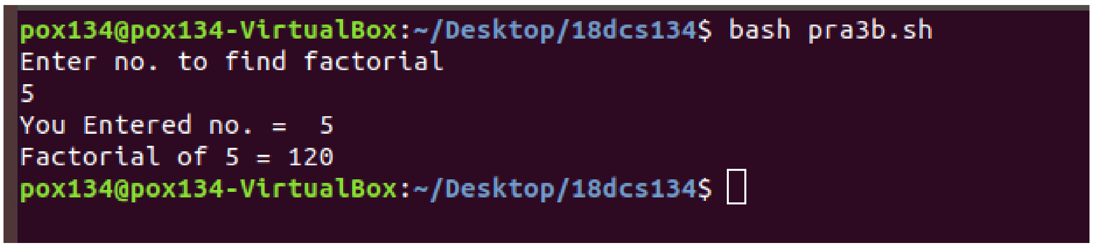


```
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ bash pra3a.sh
Enter no. to find it's fibonacci
5
You enter no. = 5
value of i= 2
1
1
2
3
5
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ 
```

2) CODE:

```
#Factorial Program

echo "Enter no. to find factorial"

read num

echo "You Entered no. =  $num"
mul=1
for ((i=1;i<num+1;i++))
do
        mul=$((mul*i))

done

echo "Factorial of $num = $mul"
```

OUTPUT:

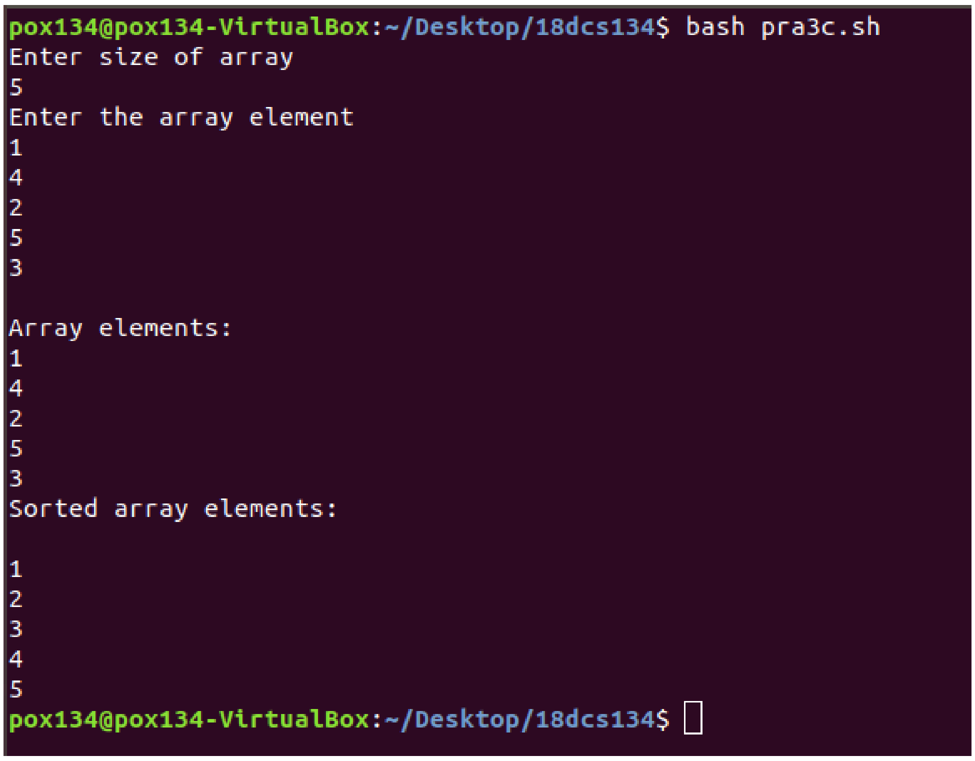


```
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ bash pra3b.sh
Enter no. to find factorial
5
You Entered no. =  5
Factorial of 5 = 120
pox134@pox134-VirtualBox:~/Desktop/18dcs134$
```

3) CODE:

```
#bubble sort

echo "Enter size of array"
read n

echo "Enter the array element"

for(( i=0;i<n;i++ ))
do
	read arr[i]

done

echo

echo "Array elements:"

for(( i=0;i<n;i++ ))
do
	echo ${arr[i]}

done

for(( i=0;i<n;i++ ))
do
	for(( j=0;j<n-i;j++ ))
	do
		k=$((j+1))
		if [[ ${arr[j]} -gt ${arr[k]} ]]
		then
			temp=${arr[j]}
			arr[j]=${arr[k]}
			arr[k]=$temp
		fi

	done
done



echo "Sorted array elements:"

for(( i=0;i<n+1;i++ ))
do
	echo ${arr[i]}

done
```

OUTPUT:



```
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ bash pra3c.sh
Enter size of array
5
Enter the array element
1
4
2
5
3

Array elements:
1
4
2
5
3
Sorted array elements:

1
2
3
4
5
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ 
```

Practical: 4

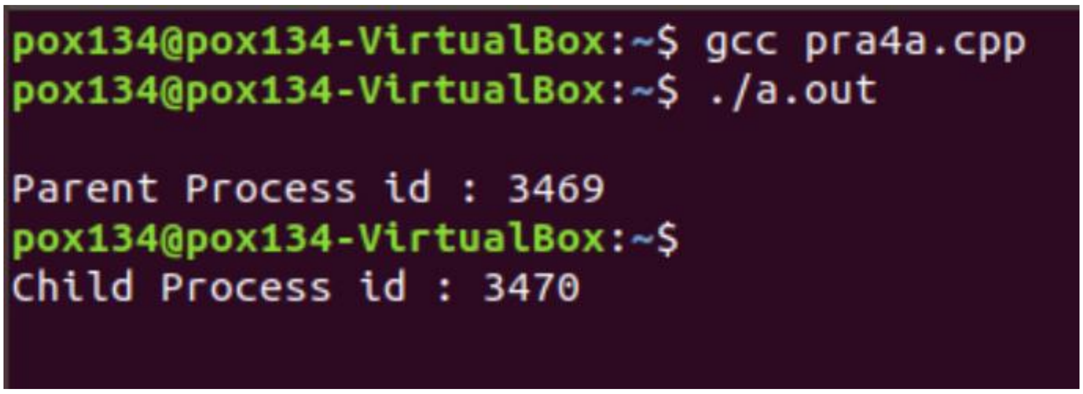
AIM: Write programs using the following system calls of UNIX operating system: fork, exec, getpid, exit, wait, stat, readdir, opendir.

1. Write a program to execute fork() and find out the process id by getpid() system call.

CODE:

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

int main()
{
        int pid;
        pid = fork();
        if(pid<0)
        {
                printf("\nFork unsuccessful");
        }
        else if (pid == 0)
        {
                printf("\nChild Process id : %d\n",getpid());
        }
        else
        {
                printf("\nParent Process id : %d\n",getpid());
        }
        return 0;
}
```

OUTPUT:



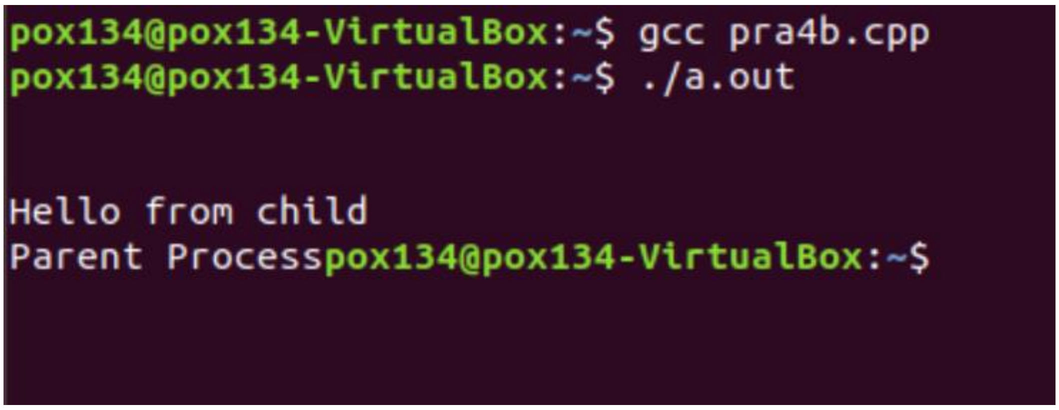
```
pox134@pox134-VirtualBox:~$ gcc pra4a.cpp
pox134@pox134-VirtualBox:~$ ./a.out

Parent Process id : 3469
pox134@pox134-VirtualBox:~$
Child Process id : 3470
```

2. Write a program to execute following system call fork(), execl(), getpid(), exit(), wait() for a process.

CODE:

```
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>
#include <stdlib.h>

int main()
{
	int pid;
	pid = fork();
	if(pid>0)
	{
		printf("\nParent Process");
		wait(NULL);
		exit(0);
	}
	else if (pid == 0)
	{
		printf("\nChild Process id");
		execlp("echo","echo","Hello from child",NULL);
	}
	return 0;
}
```

OUTPUT:



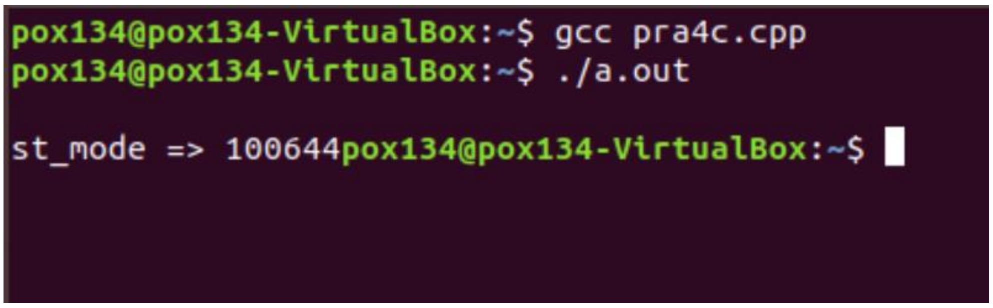
```
pox134@pox134-VirtualBox:~$ gcc pra4b.cpp
pox134@pox134-VirtualBox:~$ ./a.out


Hello from child
Parent Processpox134@pox134-VirtualBox:~$
```

3. Write a program to find out status of named file (program of working stat() system call).

CODE:

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

int main()
{
        struct stat sfile;
        stat("pra4a.cpp", &sfile);
        printf("\nst_mode => %o",sfile.st_mode);
        return 0;
}
```

OUTPUT:



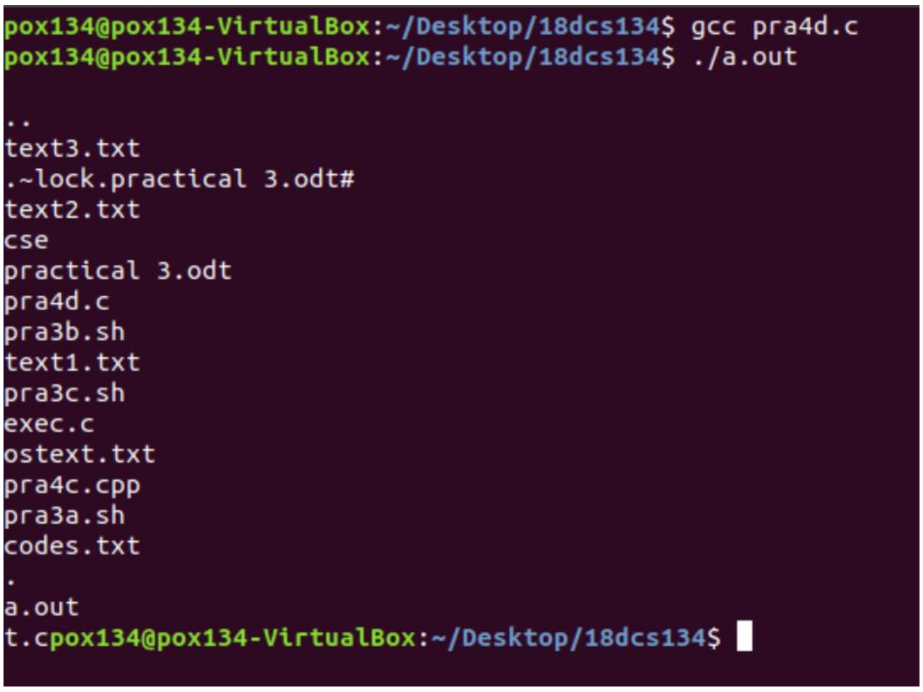
```
pox134@pox134-VirtualBox:~$ gcc pra4c.cpp
pox134@pox134-VirtualBox:~$ ./a.out

st_mode => 100644pox134@pox134-VirtualBox:~$ 
```

4. Write a program for "ls" command implementation using opendir() &amp; readdir() system call.

CODE:

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

int main()
{
        struct dirent *de;
        DIR *dr = opendir(".");

        if(dr == NULL)
        {
                printf("\nCould not open current directory");
                return 0;
        }
        while((de = readdir(dr)) != NULL)
                printf("\n%s", de->d_name);
        closedir(dr);
        return 0;
}
```

OUTPUT:



```
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ gcc pra4d.c
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ ./a.out

..
text3.txt
.~lock.practical 3.odt#
text2.txt
cse
practical 3.odt
pra4d.c
pra3b.sh
text1.txt
pra3c.sh
exec.c
ostext.txt
pra4c.cpp
pra3a.sh
codes.txt
.
a.out
t.cpox134@pox134-VirtualBox:~/Desktop/18dcs134$ 
```

Practical: 5

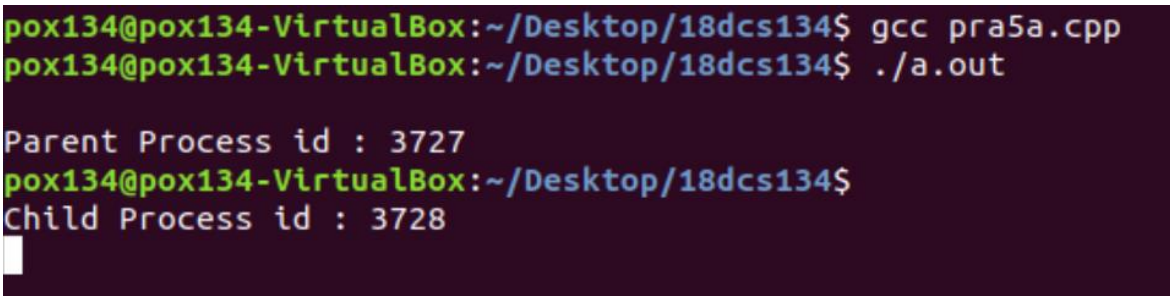
AIM:
Process control system calls:
A. The demonstration of fork()
B. execve() and wait() system calls along with zombie and orphan states.

A. The demonstration of fork()

CODE:

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

int main()
{
        int pid;
        pid = fork();
        if(pid<0)
        {
                printf("\nFork unsuccessful");
        }
        else if (pid == 0)
        {
                printf("\nChild Process id : %d\n",getpid());
        }
        else
        {
                printf("\nParent Process id : %d\n",getpid());
        }
        return 0;
}
```

OUTPUT:



```
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ gcc pra5a.cpp
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ ./a.out

Parent Process id : 3727
pox134@pox134-VirtualBox:~/Desktop/18dcs134$
Child Process id : 3728
```

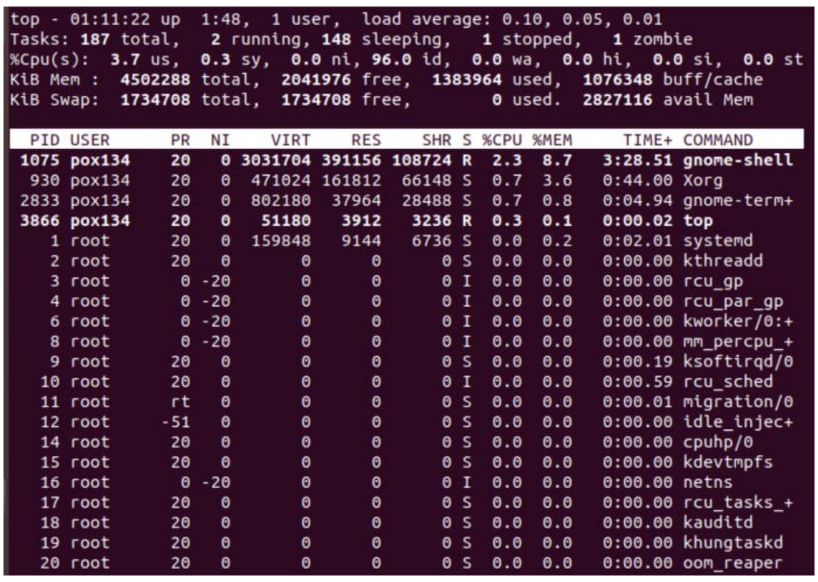
B. execve() and wait() system calls along with zombie and orphan states.

CODE:(zombie state)

```
//Create a zombie process. (Child is zombie since parent is sleeping)
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
        // Fork returns process id
        // in parent process

        pid_t child_pid = fork();

        // Parent process
        if (child_pid > 0)
                sleep(50);

        // Child process
        else
                exit(0);

        return 0;
}
```

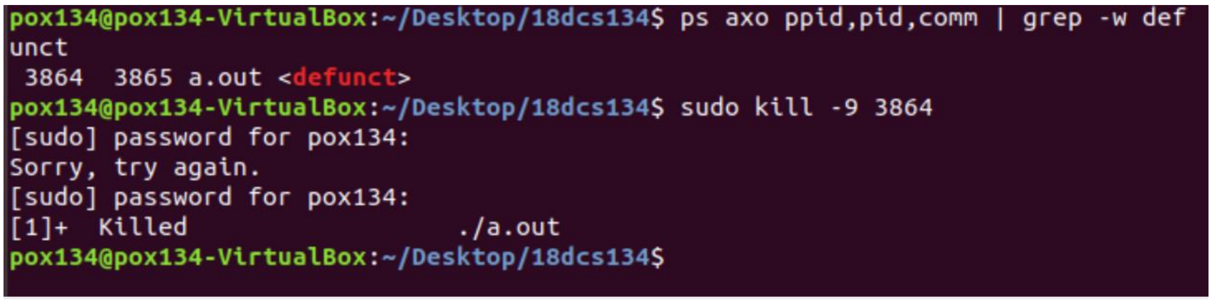
OUTPUT:

In top, we see 1 Zombie process



```
top - 01:11:22 up  1:48,  1 user,  load average: 0.10, 0.05, 0.01
Tasks: 187 total,   2 running, 148 sleeping,   1 stopped,   1 zombie
%Cpu(s):  3.7 us,  0.3 sy,  0.0 ni, 96.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem :  4502288 total,  2041976 free,  1383964 used,  1076348 buff/cache
KiB Swap:  1734708 total,  1734708 free,        0 used.  2827116 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+ COMMAND
 1075 pox134    20   0 3031704 391156 108724 R  2.3  8.7   3:28.51 gnome-shell
  930 pox134    20   0  471024 161812  66148 S  0.7  3.6   0:44.00 Xorg
 2833 pox134    20   0  802180  37964  28488 S  0.7  0.8   0:04.94 gnome-term+
 3866 pox134    20   0   51180   3912   3236 R  0.3  0.1   0:00.02 top
    1 root      20   0  159848   9144   6736 S  0.0  0.2   0:02.01 systemd
    2 root      20   0       0      0      0 S  0.0  0.0   0:00.00 kthreadd
    3 root       0 -20       0      0      0 I  0.0  0.0   0:00.00 rcu_gp
    4 root       0 -20       0      0      0 I  0.0  0.0   0:00.00 rcu_par_gp
    6 root       0 -20       0      0      0 I  0.0  0.0   0:00.00 kworker/0:+
    8 root       0 -20       0      0      0 I  0.0  0.0   0:00.00 mm_percpu_+
    9 root      20   0       0      0      0 S  0.0  0.0   0:00.19 ksoftirqd/0
   10 root      20   0       0      0      0 I  0.0  0.0   0:00.59 rcu_sched
   11 root      rt   0       0      0      0 S  0.0  0.0   0:00.01 migration/0
   12 root     -51   0       0      0      0 S  0.0  0.0   0:00.00 idle_injec+
   14 root      20   0       0      0      0 S  0.0  0.0   0:00.00 cpuhp/0
   15 root      20   0       0      0      0 S  0.0  0.0   0:00.00 kdevtmpfs
   16 root       0 -20       0      0      0 I  0.0  0.0   0:00.00 netns
   17 root      20   0       0      0      0 S  0.0  0.0   0:00.00 rcu_tasks_+
   18 root      20   0       0      0      0 S  0.0  0.0   0:00.00 kauditd
   19 root      20   0       0      0      0 S  0.0  0.0   0:00.00 khungtaskd
   20 root      20   0       0      0      0 S  0.0  0.0   0:00.00 oom_reaper
```

Killing the zombie process.



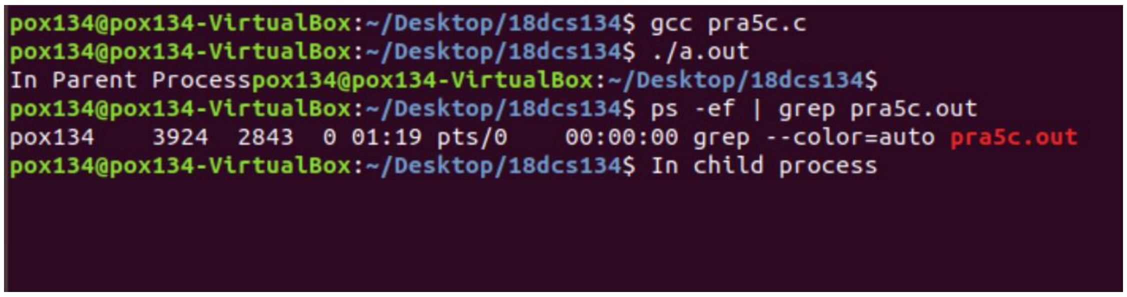
```
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ ps axo ppid,pid,comm | grep -w def
unct
 3864  3865 a.out <defunct>
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ sudo kill -9 3864
[sudo] password for pox134:
Sorry, try again.
[sudo] password for pox134:
[1]+  Killed                  ./a.out
pox134@pox134-VirtualBox:~/Desktop/18dcs134$
```

CODE:(Orphan Process)

```
// Creating an Orphan Process.
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
        //Create a child process

        int pid = fork();

        if (pid > 0)
                printf("In Parent Process");

        // Note that pid is 0 in child process
        // and negative if fork() fails
        else if (pid == 0)
        {
                sleep(30);
                printf("In child process");
        }

        return 0;
}
```

OUTPUT:

Finding an Orphan process.



```
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ gcc pra5c.c
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ ./a.out
In Parent Processpox134@pox134-VirtualBox:~/Desktop/18dcs134$
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ ps -ef | grep pra5c.out
pox134    3924  2843  0 01:19 pts/0    00:00:00 grep --color=auto pra5c.out
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ In child process
```

Practical: 6

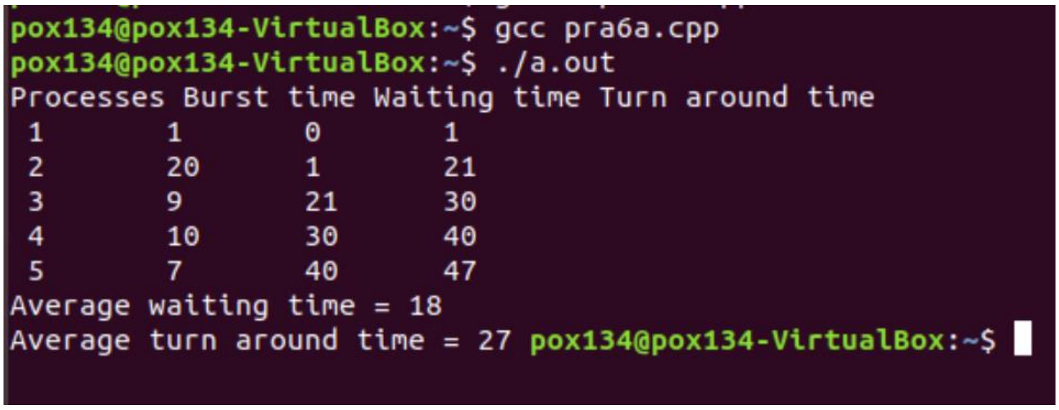
AIM: Write a C program in UNIX to implement Process scheduling algorithms and compare.

A. First Come First Serve (FCFS) Scheduling

CODE:

```
#include <stdio.h>
void findWaitingTime(int processes[], int n, int bt[], int wt[])
{
   wt[0] = 0;
   for (int i = 1; i < n; i++)
      wt[i] = bt[i - 1] + wt[i - 1];
}
void findTurnAroundTime(int processes[], int n, int bt[], int wt[], int tat[])
{
   for (int i = 0; i < n; i++)
      tat[i] = bt[i] + wt[i];
}
void findavgTime(int processes[], int n, int bt[])
{
   int wt[n], tat[n], total_wt = 0, total_tat = 0;
   findWaitingTime(processes, n, bt, wt);
   findTurnAroundTime(processes, n, bt, wt, tat);
   printf("Processes Burst time Waiting time Turn around time\n");
   for (int i = 0; i < n; i++)
   {
      total_wt = total_wt + wt[i];
      total_tat = total_tat + tat[i];
      printf(" %d ", (i + 1));
      printf("%d ", bt[i]);
      printf("%d", wt[i]);
      printf("%d\n", tat[i]);
   }
   int s = (float)total_wt / (float)n;
   int t = (float)total_tat / (float)n;
   printf("Average waiting time = %d", s);
   printf("\n");
   printf("Average turn around time = %d ", t);
}
int main()
{
   int processes[] = {1, 2, 3, 4, 5};
   int n = sizeof processes / sizeof processes[0];
   int burst_time[] = {1, 20, 9, 10, 7};
   findavgTime(processes, n, burst_time);
   return 0;
}
```

OUTPUT:



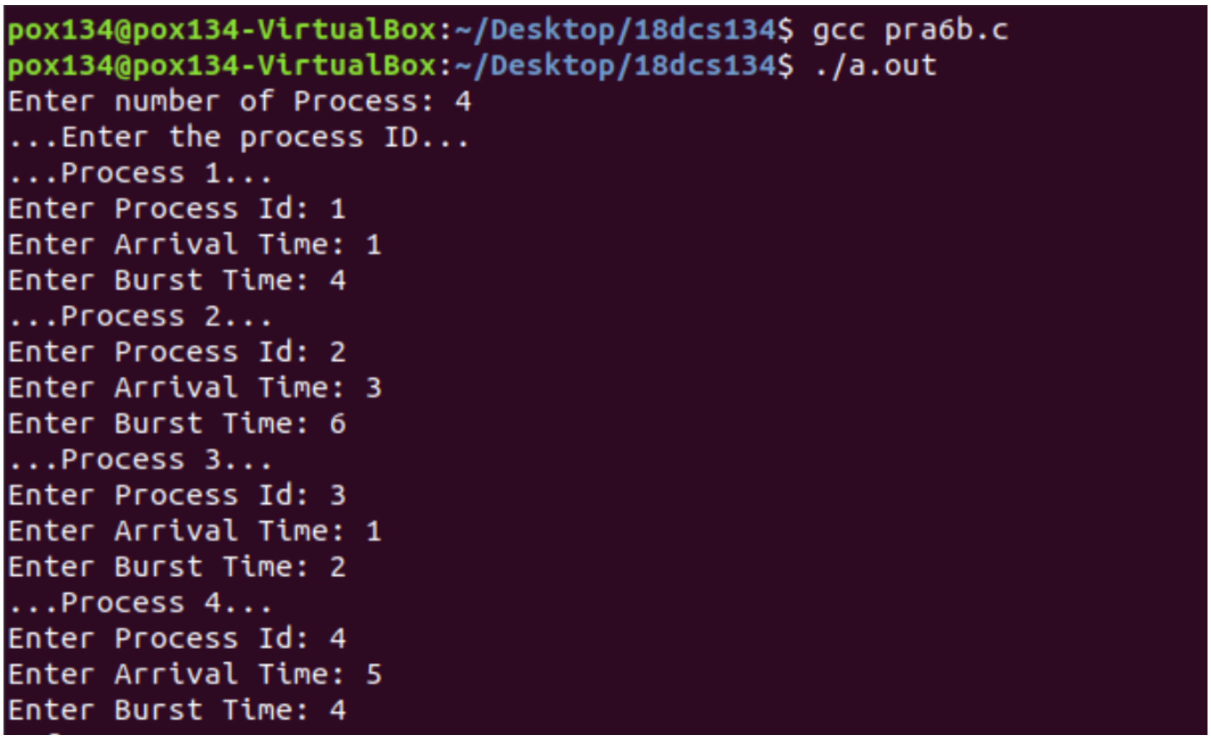
```
pox134@pox134-VirtualBox:~$ gcc pra6a.cpp
pox134@pox134-VirtualBox:~$ ./a.out
Processes Burst time Waiting time Turn around time
 1        1        0        1
 2        20       1        21
 3        9        21       30
 4        10       30       40
 5        7        40       47
Average waiting time = 18
Average turn around time = 27 pox134@pox134-VirtualBox:~$
```

B. Shortest-Job-First (SJF) Scheduling

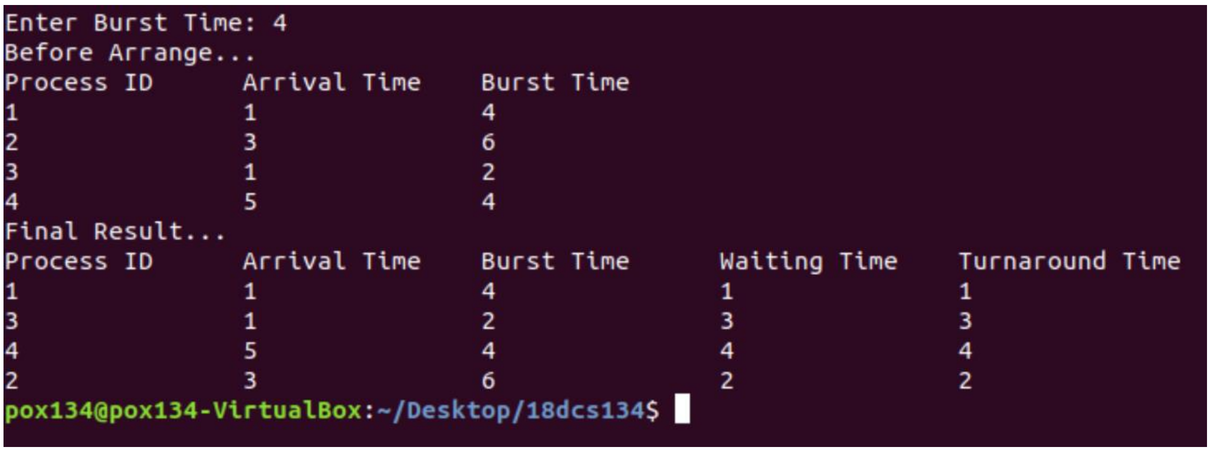
CODE:

```
#include <stdio.h>
int mat[10][6];

void swap(int *a, int *b)
{
    int temp = *a;
    *a = *b;
    *b = temp;
}

void arrangeArrival(int num, int mat[][6])
{
    int i, j, k;
    for (i = 0; i < num; i++)
    {
        for (j = 0; j < num - i - 1; j++)
        {
            if (mat[j][1] > mat[j + 1][1])
            {
                for (k = 0; k < 5; k++)
                {
                    swap(&mat[j][k], &mat[j + 1][k]);
                }
            }
        }
    }
}

void completionTime(int num, int mat[][6])
{
    int i, j, k;
    int temp, val;
    mat[0][3] = mat[0][1] + mat[0][2];
    mat[0][5] = mat[0][3] - mat[0][1];
    mat[0][4] = mat[0][5] - mat[0][2];

    for (i = 1; i < num; i++)
    {
        temp = mat[i - 1][3];
        int low = mat[i][2];
        for (j = i; j < num; j++)
        {
            if (temp >= mat[j][1] && low >= mat[j][2])
            {
                low = mat[j][2];
                val = j;
            }
        }
        mat[val][3] = temp + mat[val][2];
        mat[val][5] = mat[val][3] - mat[val][1];
        mat[val][4] = mat[val][5] - mat[val][2];
        for (k = 0; k < 6; k++)
        {
            swap(&mat[val][k], &mat[i][k]);
        }
    }
}
int main()
{
    int num, temp;
    printf("Enter number of Process: ");
    scanf("%d", &num);
    printf("...Enter the process ID...\n");
    for (int i = 0; i < num; i++)
    {
        printf("...Process %d...\n", (i + 1));
        printf("Enter Process Id: ");
        scanf("%d", &mat[i][0]);
        printf("Enter Arrival Time: ");
        scanf("%d", &mat[i][1]);
        printf("Enter Burst Time: ");
        scanf("%d", &mat[i][2]);
    }
    printf("Before Arrange...\n");
    printf("Process ID\tArrival Time\tBurst Time\n");
    for (int i = 0; i < num; i++)
    {
        printf("%d\t\t%d\t\t%d\n", mat[i][0], mat[i][1], mat[i][2]);
    }
    arrangeArrival(num, mat);
    completionTime(num, mat);
    printf("Final Result...\n");
    printf("Process ID\tArrival Time\tBurst Time\tWaiting Time\tTurnaround Time\n");
    for (int i = 0; i < num; i++)
    {
        printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\n", mat[i][0], mat[i][1], mat[i][2], mat[i][0], mat[i][0
]);
    }
}
```

OUTPUT:



```
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ gcc pra6b.c
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ ./a.out
Enter number of Process: 4
...Enter the process ID...
...Process 1...
Enter Process Id: 1
Enter Arrival Time: 1
Enter Burst Time: 4
...Process 2...
Enter Process Id: 2
Enter Arrival Time: 3
Enter Burst Time: 6
...Process 3...
Enter Process Id: 3
Enter Arrival Time: 1
Enter Burst Time: 2
...Process 4...
Enter Process Id: 4
Enter Arrival Time: 5
Enter Burst Time: 4
```



```
Enter Burst Time: 4
Before Arrange...
Process ID      Arrival Time    Burst Time
1               1               4
2               3               6
3               1               2
4               5               4
Final Result...
Process ID      Arrival Time    Burst Time      Waiting Time    Turnaround Time
1               1               4               1               1
3               1               2               3               3
4               5               4               4               4
2               3               6               2               2
pox134@pox134-VirtualBox:~/Desktop/18dcs134$
```

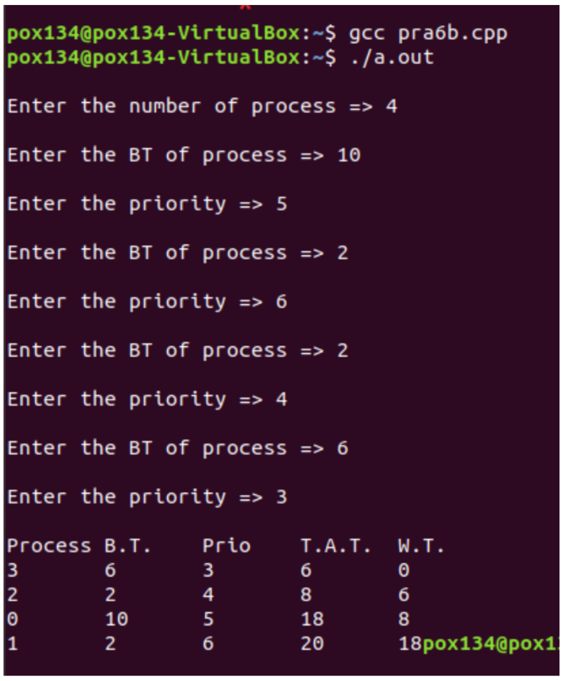
C. Priority Scheduling (Non-preemption) after completion extend on Preemption.

CODE:

```
#include <stdio.h>

int main()
{
   int n, i, bt[20][3], tat[20], wt[20];
   printf("\nEnter the number of process => ");
   scanf("%d", &n);
   for (i = 0; i < n; i++)
   {
      printf("\nEnter the BT of process => ");
      scanf("%d", &bt[i][1]);
      bt[i][0] = i;
      printf("\nEnter the priority => ");
      scanf("%d", &bt[i][2]);
   }
   int min, j;
   for (i = 0; i < n; i++)
   {
      for (j = 0; j < n - 1; j++)
      {
         if (bt[j][2] > bt[j + 1][2])
         {
            int temp0 = bt[j][0];
            int temp1 = bt[j][1];
```

```
                int temp2 = bt[j][2];
                bt[j][0] = bt[j + 1][0];
                bt[j][1] = bt[j + 1][1];
                bt[j][2] = bt[j + 1][2];
                bt[j + 1][0] = temp0;
                bt[j + 1][1] = temp1;
                bt[j + 1][2] = temp2;
            }
        }
    }

    tat[0] = bt[0][1];
    wt[0] = 0;
    for (i = 1; i < n; i++)
    {
        tat[i] = tat[i - 1] + bt[i][1];
        wt[i] = tat[i] - bt[i][1];
    }
    printf("\nProcess\tB.T.\tPrio\tT.A.T.\tW.T.");
    for (i = 0; i < n; i++)
    {
        printf("\n%d\t%d\t%d\t%d\t%d", bt[i][0], bt[i][1], bt[i][2], tat[i], wt[i]);
    }
}
```

OUTPUT:



```
pox134@pox134-VirtualBox:~$ gcc pra6b.cpp
pox134@pox134-VirtualBox:~$ ./a.out

Enter the number of process => 4

Enter the BT of process => 10

Enter the priority => 5

Enter the BT of process => 2

Enter the priority => 6

Enter the BT of process => 2

Enter the priority => 4

Enter the BT of process => 6

Enter the priority => 3

Process B.T.    Prio    T.A.T.  W.T.
3       6       3       6       0
2       2       4       8       6
0       10      5       18      8
1       2       6       20      18pox134@pox1
```

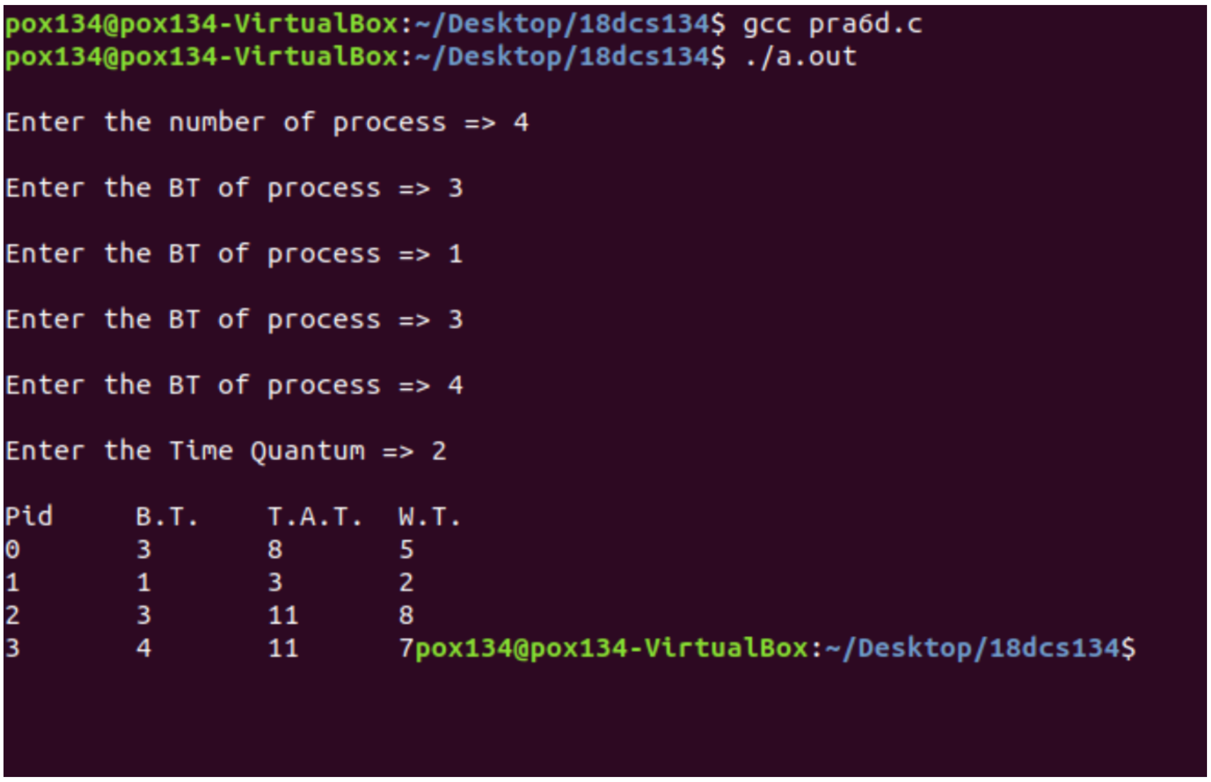
D. Round Robin(RR) Scheduling

CODE:

```
#include <stdio.h>
int main()
{
    int n, bt[20], rem[20], ct[20], tat[20], wt[20], i, sum = 0, tq, counter = 0;
    printf("\nEnter the number of process => ");
    scanf("%d", &n);
    for (i = 0; i < n; i++)
    {
        printf("\nEnter the BT of process => ");
        scanf("%d", &bt[i]);
        rem[i] = bt[i];
        sum += bt[i];
    }

    int flag = 1;
    printf("\nEnter the Time Quantum => ");
    scanf("%d", &tq);
    i = 0;

    while (flag == 1)
    {
        if (rem[i] == 0)
        {
            flag = 0;
        }
        else if (rem[i] <= tq)
        {
            flag = 1;
            counter += rem[i];
            rem[1] = 0;
            ct[i] = counter;
        }
        else
        {
            flag = 1;
            counter += tq;
            rem[i] -= tq;
        }
        i++;
        if (i == n)
```

```
            i = 0;
    }
    for (i = 0; i < n; i++)
    {
        tat[i] = ct[i];
        if (tat[i] == 0)
            tat[i] = sum;
        wt[i] = tat[i] - bt[i];
    }
    printf("\nPid\tB.T.\tT.A.T.\tW.T.");
    for (i = 0; i < n; i++)
    {
        printf("\n%d\t%d\t%d\t%d", i, bt[i], tat[i], wt[i]);
    }
}
```

OUTPUT:



```
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ gcc pra6d.c
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ ./a.out

Enter the number of process => 4

Enter the BT of process => 3

Enter the BT of process => 1

Enter the BT of process => 3

Enter the BT of process => 4

Enter the Time Quantum => 2

Pid     B.T.    T.A.T.  W.T.
0       3       8       5
1       1       3       2
2       3       11      8
3       4       11      7pox134@pox134-VirtualBox:~/Desktop/18dcs134$
```

Practical: 7

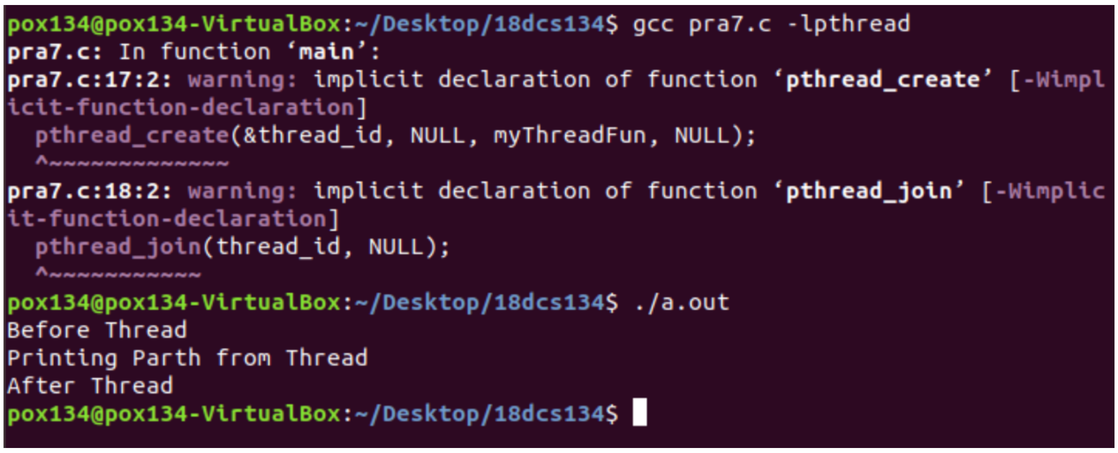
AIM:

Thread management using pthread library. Write a simple program to understand it.

CODE:

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>

void *myThreadFun(void *vargp)
{
   sleep(1);
   printf("Printing Parth from Thread \n");
   return NULL;
}

int main()
{
   pthread_t thread_id;
   printf("Before Thread\n");
   pthread_create(&thread_id, NULL, myThreadFun, NULL);
   pthread_join(thread_id, NULL);
   printf("After Thread\n");
   exit(0);
}
```

OUTPUT:



```
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ gcc pra7.c -lpthread
pra7.c: In function 'main':
pra7.c:17:2: warning: implicit declaration of function 'pthread_create' [-Wimpl
icit-function-declaration]
  pthread_create(&thread_id, NULL, myThreadFun, NULL);
  ^~~~~~~~~~~~~~
pra7.c:18:2: warning: implicit declaration of function 'pthread_join' [-Wimplic
it-function-declaration]
  pthread_join(thread_id, NULL);
  ^~~~~~~~~~~~
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ ./a.out
Before Thread
Printing Parth from Thread
After Thread
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ 
```

Practical: 8

AIM:

Write a C program in UNIX to implement Bankers algorithm for Deadlock Avoidance.

CODE:

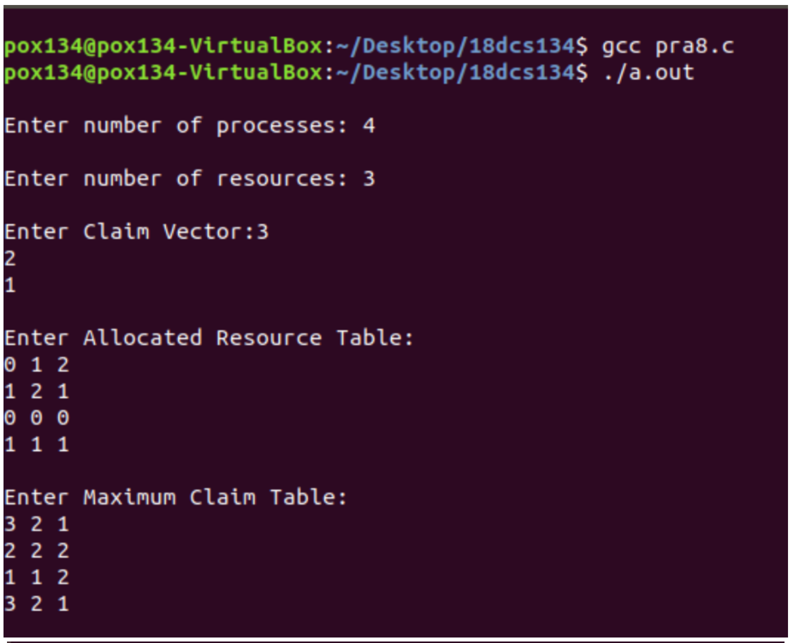
```
#include <stdio.h>
int current[5][5], maximum_claim[5][5], available[5];
int allocation[5] = {0, 0, 0, 0, 0};
int maxres[5], running[5], safe = 0;
int counter = 0, i, j, exec, resources, processes, k = 1;
int main()
{
   printf("\nEnter number of processes: ");
   scanf("%d", &processes);
   for (i = 0; i < processes; i++)
   {
      running[i] = 1;
      counter++;
   }
   printf("\nEnter number of resources: ");
   scanf("%d", &resources);
   printf("\nEnter Claim Vector:");
   for (i = 0; i < resources; i++)
   {
      scanf("%d", &maxres[i]);
   }
   printf("\nEnter Allocated Resource Table:\n");
   for (i = 0; i < processes; i++)
   {
      for (j = 0; j < resources; j++)
      {
         scanf("%d", &current[i][j]);
      }
   }
   printf("\nEnter Maximum Claim Table:\n");
   for (i = 0; i < processes; i++)
   {
      for (j = 0; j < resources; j++)
      {
         scanf("%d", &maximum_claim[i][j]);
      }
   }
```

```
printf("\nThe Claim Vector is: ");
for (i = 0; i < resources; i++)
{
    printf("\t%d", maxres[i]);
}
printf("\nThe Allocated Resource Table:\n");
for (i = 0; i < processes; i++)
{
    for (j = 0; j < resources; j++)
    {
        printf("\t%d", current[i][j]);
    }
    printf("\n");
}

printf("\nThe Maximum Claim Table:\n");
for (i = 0; i < processes; i++)
{
    for (j = 0; j < resources; j++)
    {
        printf("\t%d", maximum_claim[i][j]);
    }
    printf("\n");
}
for (i = 0; i < processes; i++)
{
    for (j = 0; j < resources; j++)
    {
        allocation[j] += current[i][j];
    }
}
printf("\nAllocated resources:");
for (i = 0; i < resources; i++)
{
    printf("\t%d", allocation[i]);
}
for (i = 0; i < resources; i++)
{
    available[i] = maxres[i] - allocation[i];
}
printf("\nAvailable resources:");
for (i = 0; i < resources; i++)
{
    printf("\t%d", available[i]);
```

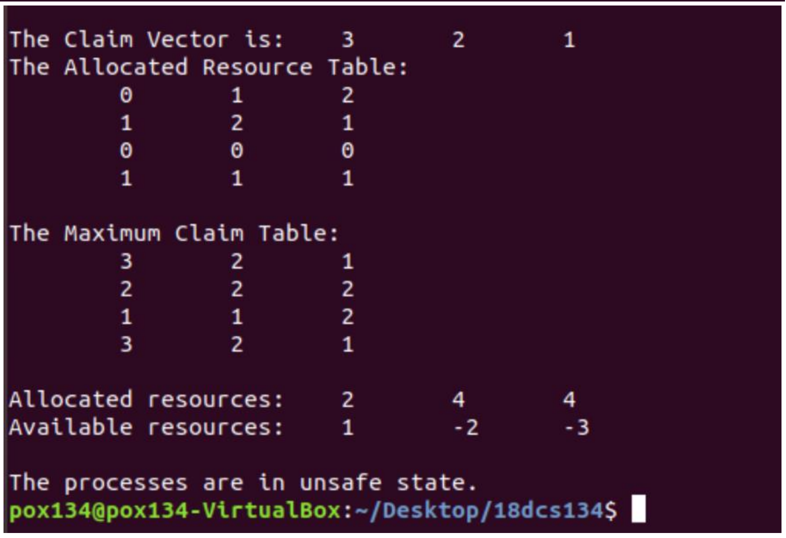
```
}
printf("\n");
while (counter != 0)
{
    safe = 0;
    for (i = 0; i < processes; i++)
    {
        if (running[i])
        {
            exec = 1;
            for (j = 0; j < resources; j++)
            {
                if (maximum_claim[i][j] - current[i][j] > available[j])
                {
                    exec = 0;
                    break;
                }
            }
            if (exec)
            {
                printf("\nProcess%d is executing\n", i + 1);
                running[i] = 0;
                counter--;
                safe = 1;

                for (j = 0; j < resources; j++)
                {
                    available[j] += current[i][j];
                }
                break;
            }
        }
    }
    if (!safe)
    {
        printf("\nThe processes are in unsafe state.\n");
        break;
    }
    else
    {
        printf("\nThe process is in safe state");
        printf("\nAvailable vector:");

        for (i = 0; i < resources; i++)
```

```
                {
                    printf("\t%d", available[i]);
                }

                printf("\n");
            }
        }
    return 0;
}
```

OUTPUT:



```
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ gcc pra8.c
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ ./a.out

Enter number of processes: 4

Enter number of resources: 3

Enter Claim Vector:3
2
1

Enter Allocated Resource Table:
0 1 2
1 2 1
0 0 0
1 1 1

Enter Maximum Claim Table:
3 2 1
2 2 2
1 1 2
3 2 1
```



```
The Claim Vector is:    3       2       1
The Allocated Resource Table:
        0       1       2
        1       2       1
        0       0       0
        1       1       1

The Maximum Claim Table:
        3       2       1
        2       2       2
        1       1       2
        3       2       1

Allocated resources:    2       4       4
Available resources:    1       -2      -3

The processes are in unsafe state.
pox134@pox134-VirtualBox:~/Desktop/18dcs134$
```

Practical: 9

AIM:

Write a C program in UNIX to perform Memory allocation algorithms and calculate Internal and External Fragmentation. (First Fit, Best Fit, Worst Fit).

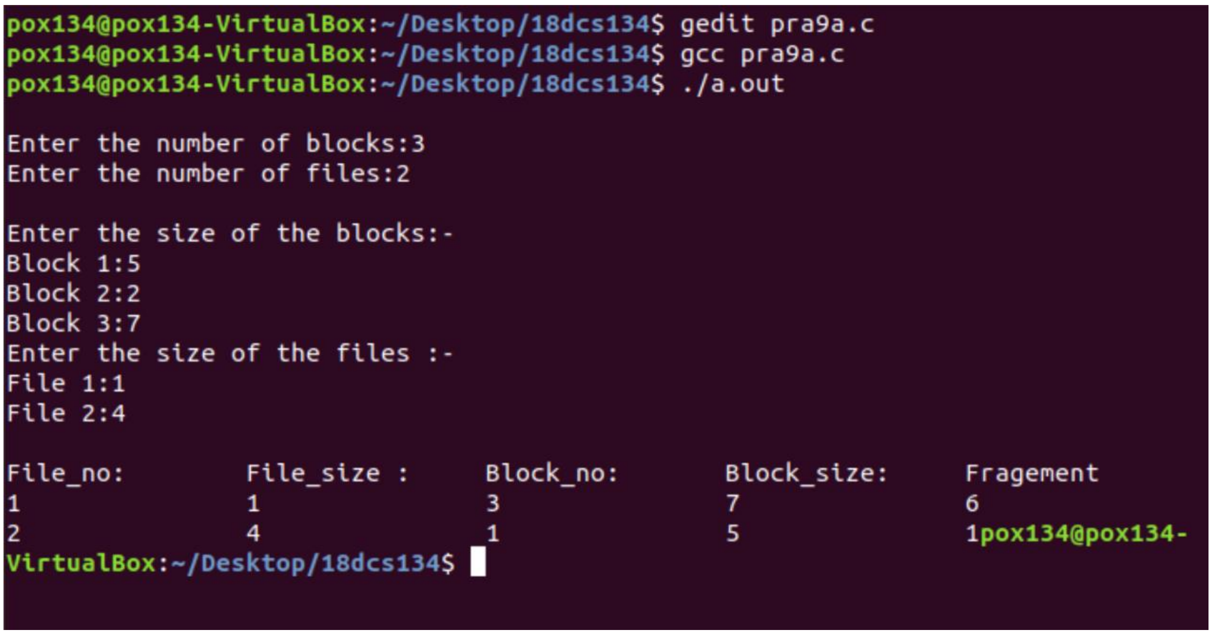
FIRST FIT:

CODE:

```
#include <stdio.h>
#define max 25
void main()
{
   int frag[max], b[max], f[max], i, j, nb, nf, temp, highest = 0;
   static int bf[max], ff[max];
   printf("\nEnter the number of blocks:");
   scanf("%d", &nb);
   printf("Enter the number of files:");
   scanf("%d", &nf);
   printf("\nEnter the size of the blocks:-\n");
   for (i = 1; i <= nb; i++)
   {
      printf("Block %d:", i);
      scanf("%d", &b[i]);
   }
   printf("Enter the size of the files :-\n");
   for (i = 1; i <= nf; i++)
   {
      printf("File %d:", i);
      scanf("%d", &f[i]);
   }
   for (i = 1; i <= nf; i++)
   {

      for (j = 1; j <= nb; j++)
      {
         if (bf[j] != 1) //if bf[j] is not allocated
         {
            temp = b[j] - f[i];
            if (temp >= 0)
               if (highest < temp)
               {
                  ff[i] = j;
                  highest = temp;
               }
         }
      }
```

```
        frag[i] = highest;
        bf[ff[i]] = 1;
        highest = 0;
    }
    printf("\nFile_no:\tFile_size :\tBlock_no:\tBlock_size:\tFragement");
    for (i = 1; i <= nf; i++)
        printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d", i, f[i], ff[i], b[ff[i]], frag[i]);
}
```

OUTPUT:



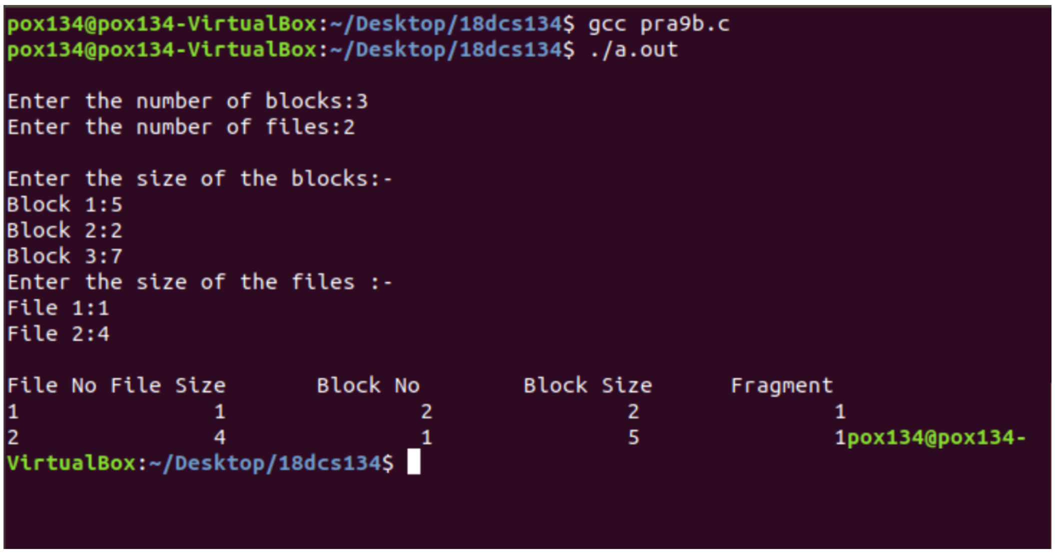
```
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ gedit pra9a.c
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ gcc pra9a.c
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ ./a.out

Enter the number of blocks:3
Enter the number of files:2

Enter the size of the blocks:-
Block 1:5
Block 2:2
Block 3:7
Enter the size of the files :-
File 1:1
File 2:4

File_no:        File_size :     Block_no:       Block_size:     Fragement
1               1               3               7               6
2               4               1               5               1pox134@pox134-
VirtualBox:~/Desktop/18dcs134$
```

BEST FIT:

CODE:

```
#include <stdio.h>
#define max 25
void main()
{
    int frag[max], b[max], f[max], i, j, nb, nf, temp, lowest = 10000;
    static int bf[max], ff[max];
    printf("\nEnter the number of blocks:");
    scanf("%d", &nb);
    printf("Enter the number of files:");
    scanf("%d", &nf);
    printf("\nEnter the size of the blocks:-\n");
    for (i = 1; i <= nb; i++)
    {
        printf("Block %d:", i);
        scanf("%d", &b[i]);
    }
    printf("Enter the size of the files :-\n");
    for (i = 1; i <= nf; i++)
    {
        printf("File %d:", i);
        scanf("%d", &f[i]);
    }
    for (i = 1; i <= nf; i++)
    {
        for (j = 1; j <= nb; j++)
        {
            if (bf[j] != 1)
            {
                temp = b[j] - f[i];
                if (temp >= 0)
                    if (lowest > temp)
                    {
                        ff[i] = j;

                        lowest = temp;
                    }
            }
        }
        frag[i] = lowest;
        bf[ff[i]] = 1;
        lowest = 10000;
    }
    printf("\nFile No\tFile Size \tBlock No\tBlock Size\tFragment");
    for (i = 1; i <= nf && ff[i] != 0; i++)
        printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d", i, f[i], ff[i], b[ff[i]], frag[i]);
}
```

OUTPUT:



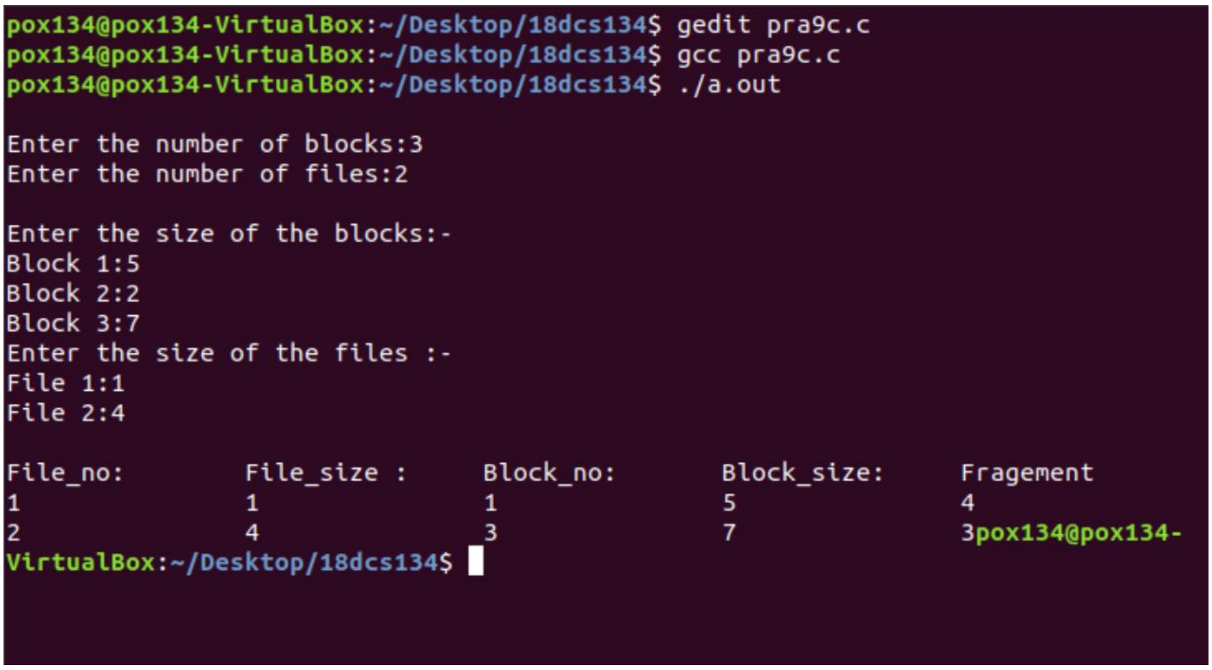
```
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ gcc pra9b.c
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ ./a.out

Enter the number of blocks:3
Enter the number of files:2

Enter the size of the blocks:-
Block 1:5
Block 2:2
Block 3:7
Enter the size of the files :-
File 1:1
File 2:4

File No File Size       Block No        Block Size      Fragment
1               1               2               2               1
2               4               1               5               1pox134@pox134-
VirtualBox:~/Desktop/18dcs134$
```

WORST FIT:

CODE:

```
#include <stdio.h>
#define max 25
void main()
{
int frag[max], b[max], f[max], i, j, nb, nf, temp;
static int bf[max], ff[max];
printf("\nEnter the number of blocks:");
scanf("%d", &nb);
printf("Enter the number of files:");
scanf("%d", &nf);
printf("\nEnter the size of the blocks:-\n");
for (i = 1; i <= nb; i++)
{
   printf("Block %d:", i);
   scanf("%d", &b[i]);
}
printf("Enter the size of the files :-\n");
for (i = 1; i <= nf; i++)
{
   printf("File %d:", i);
   scanf("%d", &f[i]);
}
for (i = 1; i <= nf; i++)
{
   for (j = 1; j <= nb; j++)
   {
      if (bf[j] != 1)
      {
         temp = b[j] - f[i];
         if (temp >= 0)
         {
            ff[i] = j;
            break;
         }
      }
   }
   frag[i] = temp;
   bf[ff[i]] = 1;
}
printf("\nFile_no:\tFile_size :\tBlock_no:\tBlock_size:\tFragement");
for (i = 1; i <= nf; i++)
   printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d", i, f[i], ff[i], b[ff[i]], frag[i]);
}
```

OUTPUT:



```
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ gedit pra9c.c
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ gcc pra9c.c
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ ./a.out

Enter the number of blocks:3
Enter the number of files:2

Enter the size of the blocks:-
Block 1:5
Block 2:2
Block 3:7
Enter the size of the files :-
File 1:1
File 2:4

File_no:        File_size :     Block_no:       Block_size:     Fragement
1               1               1               5               4
2               4               3               7               3pox134@pox134-
VirtualBox:~/Desktop/18dcs134$ 
```

Practical: 10

AIM:

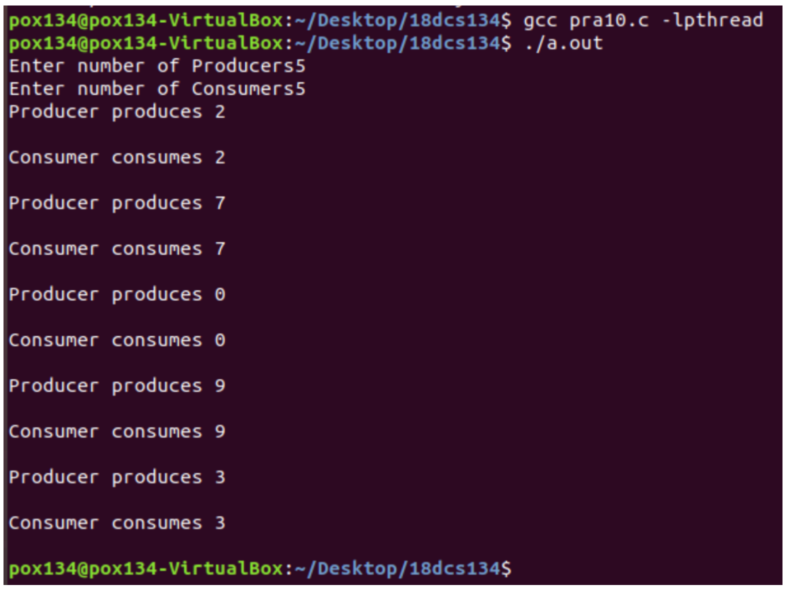
Thread synchronization using counting semaphores and mutual exclusion using mutex.

CODE:

```
#include <stdio.h>
#include <semaphore.h>
#include <sys/types.h>
#include <pthread.h>
#include <unistd.h>
#include <stdlib.h>
#define BUFFER_SIZE 1
pthread_mutex_t mutex;
sem_t empty, full;
int buffer[BUFFER_SIZE];
int counter;
pthread_t tid;
void *producer();
void *consumer();
void insert_item(int);
int remove_item();
void initilize()
{
   pthread_mutex_init(&mutex, NULL);
   sem_init(&full, 0, 0);
   sem_init(&empty, 0, BUFFER_SIZE);
}
void *producer()
{
   int item, wait_time;
   wait_time = rand() % 5;
   sleep(wait_time) % 5;
   item = rand() % 10;
   sem_wait(&empty);
   pthread_mutex_lock(&mutex);
   printf("Producer produces %d\n\n", item);
   insert_item(item);
   pthread_mutex_unlock(&mutex);
   sem_post(&full);
}
void *consumer()
{
   int item, wait_time;
   wait_time = rand() % 5;
   sleep(wait_time);
   sem_wait(&full);
   pthread_mutex_lock(&mutex);
   item = remove_item();
```

```
    printf("Consumer consumes %d\n\n", item);
    pthread_mutex_unlock(&mutex);
    sem_post(&empty);
}
void insert_item(int item)
{
    buffer[counter++] = item;
}

int remove_item()
{
    return buffer[--counter];
}

int main()
{
    int n1, n2;
    int i;
    printf("Enter number of Producers");
    scanf("%d", &n1);
    printf("Enter number of Consumers");
    scanf("%d", &n2);
    initilize();
    for (i = 0; i < n1; i++)
        pthread_create(&tid, NULL, producer, NULL);
    for (i = 0; i < n2; i++)
        pthread_create(&tid, NULL, consumer, NULL);
    sleep(5);
}
```

OUTPUT:



```
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ gcc pra10.c -lpthread
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ ./a.out
Enter number of Producers5
Enter number of Consumers5
Producer produces 2

Consumer consumes 2

Producer produces 7

Consumer consumes 7

Producer produces 0

Consumer consumes 0

Producer produces 9

Consumer consumes 9

Producer produces 3

Consumer consumes 3

pox134@pox134-VirtualBox:~/Desktop/18dcs134$
```

Practical: 11

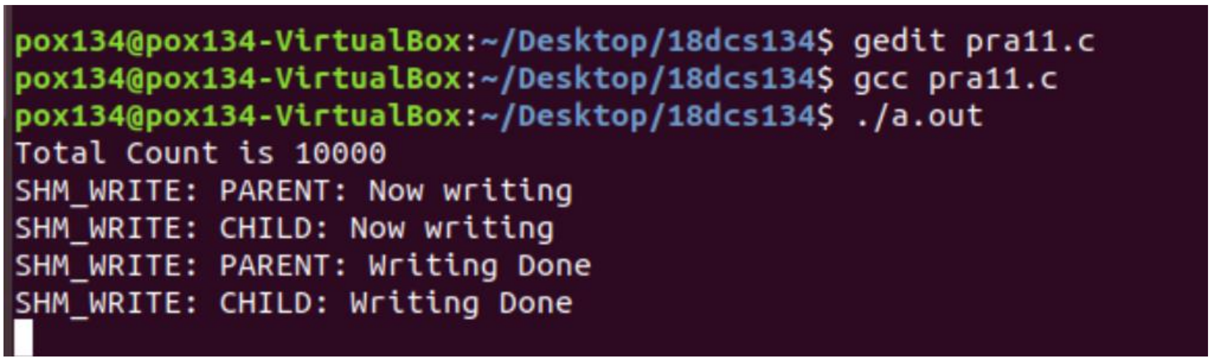
AIM:
Write a C program in UNIX to implement inter process communication (IPC) using Semaphore.

CODE:

```
#include <stdio.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/types.h>
#include <string.h>
#include <errno.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#define SHM_KEY 0x12345
struct shmseg
{
   int cntr;
   int write_complete;
   int read_complete;
};
void shared_memory_cntr_increment(int pid, struct shmseg *shmp, int total_count);
int main(int argc, char *argv[])
{
   int shmid;
   struct shmseg *shmp;
   char *bufptr;
   int total_count;
   int sleep_time;
   pid_t pid;
   if (argc != 2)
      total_count = 10000;
   else
   {
      total_count = atoi(argv[1]);
      if (total_count < 10000)
         total_count = 10000;
   }
   printf("Total Count is %d\n", total_count);
   shmid = shmget(SHM_KEY, sizeof(struct shmseg), 0644 | IPC_CREAT);
```

```
    if (shmid == -1)
    {
        perror("Shared memory");
        return 1;
    }
    shmp = shmat(shmid, NULL, 0);
    if (shmp == (void *)-1)
    {
        perror("Shared memory attach");
        return 1;
    }
    shmp->cntr = 0;
    pid = fork();
    if (pid > 0)
    {
        shared_memory_cntr_increment(pid, shmp, total_count);
    }
    else if (pid == 0)
    {
        shared_memory_cntr_increment(pid, shmp, total_count);
        return 0;
    }
    else
    {
        perror("Fork Failure\n");
        return 1;
    }
    while (shmp->read_complete != 1)
        sleep(1);
    if (shmdt(shmp) == -1)
    {
        perror("shmdt");
        return 1;
    }
    if (shmctl(shmid, IPC_RMID, 0) == -1)
    {
        perror("shmctl");
        return 1;
    }
    printf("Writing Process: Complete\n");
    return 0;
}
void shared_memory_cntr_increment(int pid, struct shmseg *shmp, int total_count)
{
    int cntr;
    int numtimes;
    int sleep_time;
    cntr = shmp->cntr;
    shmp->write_complete = 0;
    if (pid == 0)
       printf("SHM_WRITE: CHILD: Now writing\n");
    else if (pid > 0)
       printf("SHM_WRITE: PARENT: Now writing\n");
    //printf("SHM_CNTR is %d\n", shmp->cntr);
    for (numtimes = 0; numtimes < total_count; numtimes++)
    {
       cntr += 1;
       shmp->cntr = cntr;
       sleep_time = cntr % 1000;
       if (sleep_time == 0)
          sleep(1);
    }
    shmp->write_complete = 1;
    if (pid == 0)
       printf("SHM_WRITE: CHILD: Writing Done\n");
    else if (pid > 0)
       printf("SHM_WRITE: PARENT: Writing Done\n");
    return;
}
```

OUTPUT:



```
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ gedit pra11.c
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ gcc pra11.c
pox134@pox134-VirtualBox:~/Desktop/18dcs134$ ./a.out
Total Count is 10000
SHM_WRITE: PARENT: Now writing
SHM_WRITE: CHILD: Now writing
SHM_WRITE: PARENT: Writing Done
SHM_WRITE: CHILD: Writing Done
```

Practical: 12

AIM:

Kernel space programming: Implement and add a loadable kernel module to Linux kernel, demonstrate using insmod, lsmod and rmmod commands. A sample kernel space program should print the "Hello World" while loading the kernel module and "Goodbye World" while unloading the kernel module.

CODE:

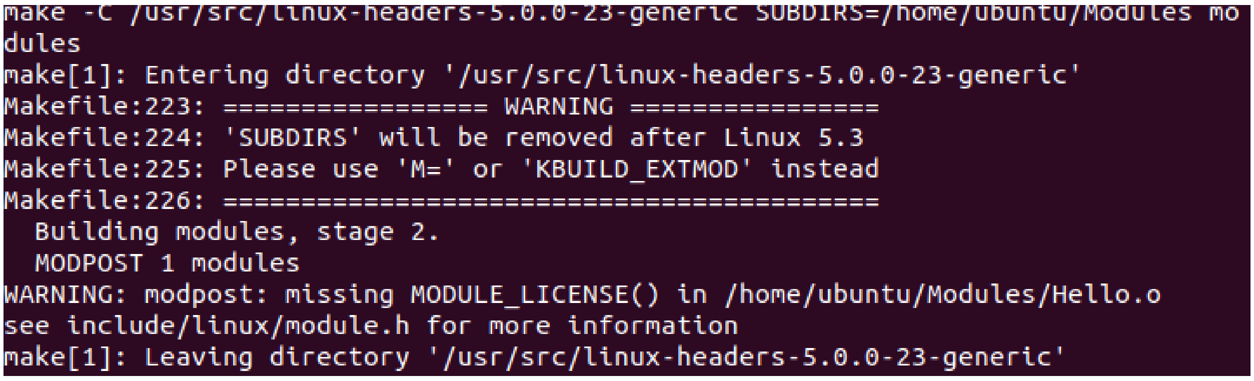
MakeFile:-

```
obj-m += Hello.o

KDIR = /usr/src/linux-headers-4.18.0-10-generic

all:

        $(MAKE) -C $(KDIR) SUBDIRS= $(PWD) modules
clean:
        rm -rf *.o *.ko *.mod* *.symvers *.order
```

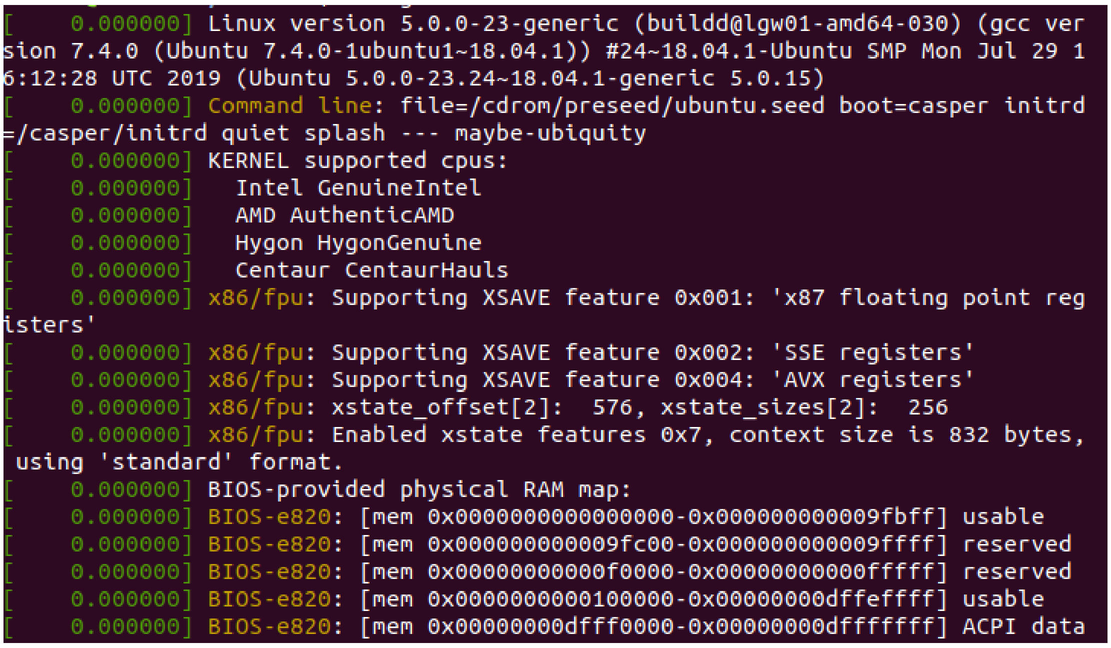
HELLO.c :-

```
#include<linux/init.h>
#include<linux/module.h>

static int hello_init(void)
{
   printk(KERN_ALERT,"HELLO WORKD\n");
   return 0;
}

static void hello_exit(void)
{
   printk(KERN_ALERT,"GOOD BYE");
}

module_init(myinit);
module_exit(hello_exit);
```

OUTPUT:



```
make -C /usr/src/linux-headers-5.0.0-23-generic SUBDIRS=/home/ubuntu/Modules mo
dules
make[1]: Entering directory '/usr/src/linux-headers-5.0.0-23-generic'
Makefile:223: ================= WARNING ================
Makefile:224: 'SUBDIRS' will be removed after Linux 5.3
Makefile:225: Please use 'M=' or 'KBUILD_EXTMOD' instead
Makefile:226: ==========================================
  Building modules, stage 2.
  MODPOST 1 modules
WARNING: modpost: missing MODULE_LICENSE() in /home/ubuntu/Modules/Hello.o
see include/linux/module.h for more information
make[1]: Leaving directory '/usr/src/linux-headers-5.0.0-23-generic'
```



```
[    0.000000] Linux version 5.0.0-23-generic (buildd@lgw01-amd64-030) (gcc ver
sion 7.4.0 (Ubuntu 7.4.0-1ubuntu1~18.04.1)) #24~18.04.1-Ubuntu SMP Mon Jul 29 1
6:12:28 UTC 2019 (Ubuntu 5.0.0-23.24~18.04.1-generic 5.0.15)
[    0.000000] Command line: file=/cdrom/preseed/ubuntu.seed boot=casper initrd
=/casper/initrd quiet splash --- maybe-ubiquity
[    0.000000] KERNEL supported cpus:
[    0.000000]   Intel GenuineIntel
[    0.000000]   AMD AuthenticAMD
[    0.000000]   Hygon HygonGenuine
[    0.000000]   Centaur CentaurHauls
[    0.000000] x86/fpu: Supporting XSAVE feature 0x001: 'x87 floating point reg
isters'
[    0.000000] x86/fpu: Supporting XSAVE feature 0x002: 'SSE registers'
[    0.000000] x86/fpu: Supporting XSAVE feature 0x004: 'AVX registers'
[    0.000000] x86/fpu: xstate_offset[2]:  576, xstate_sizes[2]:  256
[    0.000000] x86/fpu: Enabled xstate features 0x7, context size is 832 bytes,
 using 'standard' format.
[    0.000000] BIOS-provided physical RAM map:
[    0.000000] BIOS-e820: [mem 0x0000000000000000-0x000000000009fbff] usable
[    0.000000] BIOS-e820: [mem 0x000000000009fc00-0x000000000009ffff] reserved
[    0.000000] BIOS-e820: [mem 0x00000000000f0000-0x00000000000fffff] reserved
[    0.000000] BIOS-e820: [mem 0x0000000000100000-0x00000000dffeffff] usable
[    0.000000] BIOS-e820: [mem 0x00000000dfff0000-0x00000000dfffffff] ACPI data
```