ADITYA
COLLEGE OF ENGINEERING

Aditya Nagar, ADB Road, Surampalem - 533437

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

LABORATORY RECORD

NAME	:										
ROLL NO.	:										
YEAR	:										
SEMESTER	:										
LAB	:										

ADITYA
COLLEGE OF ENGINEERING
Aditya Nagar, ADB Road, Surampalem - 533437

Department of

Name : **Roll No. :**

Certified that this is the bonafide record of practical work done by

Mr. /Ms. ....

a student of ....with PIN No. ....

in the .... Laboratory during the year ....

No. of Practicals Conducted : ☐ No. of Practicals Attended : ☐

Signature - Faculty Incharge **Signature - Head of the Department**

Submitted for the Practical examinationheld on ....

EXAMINER - 1 **EXAMINER - 2**

VISION & MISSION OF THE INSTITUTE

VISION

To induce higher planes of learning by imparting technical education with

- International standards
- Applied research
- Creative Ability
- Value based instruction and to emerge as a premiere institute.

MISSION

Achieving academic excellence by providing globally acceptable technical education by forecasting technology through

- Innovative Research and development
- Industry Institute Interaction
- Empowered Manpower

VISION & MISSION OF THE DEPARTMENT

VISION

To be a recognized Computer Science and Engineering hub striving to meet the growing needs of the Industry and Society.

MISSION

M1: Imparting Quality Education through state-of-the-art infrastructure with industry Collaboration

M2: Enhance Teaching Learning Process to disseminate knowledge.

M3: Organize Skill based, Industrial and Societal Events for overall Development.

ADITYA COLLEGE OF ENGINEERING
An AUTONOMOUS Institution
Approved by AICTE, Permanently Affiliated to JNTUK, Accredited by NBA & NAAC
Recognized by UGC under Sections 2(f) and 12(B) of UGC Act, 1956
Aditya Nagar, ADB Road, Surampalem - 533 437, E.G.Dist., Ph: 99631 76662.

Department of Computer Science and Engineering

Operating Systems LAB

Academic Year 2023-24

II B.Tech I Semester

S.No	Date	Name of the Experiment	Page No.	Remarks
1		a) Study of Unix/Linux general purpose utility command list: man,who,cat, cd, cp, ps, ls, mv, rm, mkdir, rmdir, echo, more, date, time, kill, history, chmod, chown, finger, pwd, cal, logout, shutdown. b) Study of vi editor c) Study of Bash shell, Bourne shell and C shell in Unix/Linux operating system d) Study of Unix/Linux file system (tree structure) e) Study of .bashrc, /etc/bashrc and Environment variables.	1	
2		Write a C program that makes a copy of a file using standard I/O, and system calls	22	
3		Write a C program to emulate the UNIX ls –l command.	25	
4		Write a C program that illustrates how to execute two commands concurrently with a command pipe. Ex: - ls –l | sort	27	
5		Simulate the following CPU scheduling algorithms: (a) Round Robin (b) SJF (c) FCFS (d) Priority	30	
6		Multiprogramming-Memory management-Implementation of fork (), wait (), exec() and exit (), System calls	42	
7		Simulate the following: a) Multiprogramming with a fixed number of tasks (MFT) b) Multiprogramming with a variable number of tasks (MVT)	44	
8		Simulate Bankers Algorithm for Dead Lock Avoidance	49	
9		Simulate Bankers Algorithm for Dead Lock Prevention.	54	
10		Simulate the following page replacement algorithms: a) FIFO b) LRU c) LFU	60	
11		Simulate the following File allocation strategies (a) Sequenced (b) Indexed (c) Linked	69	
12		Write a C program that illustrates two processes communicating using shared memory	75	

Experiment -1

1. a) Study of Unix/Linux general purpose utility command list man,who,cat, cd, cp, ps, ls, mv, rm, mkdir, rmdir, echo, more, date, time,kill, history,chmod, chown, finger, pwd, cal, logout, shutdown.

man Command

Short for "manual," man allows a user to format and display the user manual built into Linux distributions, which documents commands and other aspects of the system.

Syntax: man [option(s)] keyword(s)

Example man ls

who Command

Identifies the users currently logged in. The "who" command lets you display the users that are currently logged into your UNIX computer system. The following information is displayed: login name, workstation name, date and time of login. Entering who am i or who am I displays your login name, workstation name, date and time you logged in.

Syntax: who [OPTION]... [FILE | ARG1 ARG2]

Example who

cat Command

cat is used to display the contents of the file. cat is used to create file.

Syntax:

cat >filename	used to create file.
cat file name	used to display content of file.

Examples:

```
$ cat>my
Balji
Navya
Rukmini
```

```
Shanmugam
^d        //Press Ctrl   D

$ cat my
Balji
Navya
Rukmini
Shanmugam
```

cd Command

The cd command, which stands for "change directory", changes the shell's current working directory.

Syntax: cd directory name

Example: cd new

```
 cd .
cd ..
```

cp Command:

Used if user wants to copy files from one location to another. Copied file is inherited from existing source file whatever permission provided in original transfer to duplicate file

Syntax: cp [option] source file destination file

Options:

-f	Forces to copy,does not ask user choice
-i	Interactive copy,asks for user choice if yes then only proceed.
-r	Recursively copies files.
-R	Recursively copy & copying of device nodes and named pipes.
-p	Transfer the characteristics of time of last modification & access, userid & Groupid,File permission.

Example:

$cp my newmy

The cp command can be used to copy a file my from current working directory to it with different *name newmy.*

ps Command

Report a snapshot of the current processes. ps displays information about a selection of the active processes.

Syntax:

ps [options]

Options:

-e	To see every process on the system.
-ejH	To print a process tree
-a	Displays all processes on a terminal, with the exception of group leaders.
-l	Displays a long listing.

ls Command

List contents of directories.

ls,l,lc,lf,lr,lx

Syntax:ls [options] [file-list]

Description

L	List Files with long information including symbolic links.
lc	List files in column
lf	List files indicating directories, executable and symbolic links.
lr	List files recursively listing any subdirectories encounterd.

mv Command

move (rename) files

Syntax: mv [OPTION]... SOURCE... DIRECTORY

Options; -b like --backup but does not accept an argument.

-f, --force do not prompt before overwriting.

-i, --interactive prompt before overwrite.

<u>rm Command</u>

Remove files or directories

Syntax: rm [OPTION]... FILE...

Options: -f, --force ignore nonexistent files, never prompt

-i prompt before every removal.

-I prompt once before removing more than three files, or when removing recursively. Less intrusive than -i, while still giving protection against most mistakes.

<u>mkdir Command</u>

make directories

Syntax: mkdir [OPTION]... DIRECTORY...

Options: -m, --mode=MODE set file mode (as in chmod), not a=rwx - umask

-p, --parents no error if existing, make parent directories as needed

-v, --verbose print a message for each created directory.

<u>rmdir Command</u>

- remove empty directories

Syntax: rmdir [OPTION]... DIRECTORY...

Options: -p, --parents remove DIRECTORY and its ancestors; e.g., `rmdir -p a/b/c' is similar to `rmdir a/b/c a/b a'

-v, --verbose output a diagnostic for every directory processed

--help display this help and exit.

<u>echo Command</u>

- display a line of text

Syntax: echo [SHORT-OPTION]... [STRING]...

echo LONG-OPTION

Options: -n do not output the trailing newline

-e enable interpretation of backslash escapes

-E disable interpretation of backslash escapes (default)

--help display this help and exit

--version output version information and exit

more Command

File perusal filter for crt viewing.more is a filter for paging through text one screenful at a time. This version is especially primitive. Users should realize that less(1) provides more(1) emulation plus extensive enhancements.

Syntax: more [-dlfpcsu] [-num] [+/pattern] [+linenum] [file ...]

Options: **-**d display help instead of ring bell

-f count logical, rather than screen lines

-l suppress pause after form feed

-p suppress scroll, clean screen and display text

-c suppress scroll, display text and clean line ends

-u suppresses underlining

-s squeezes multiple blank lines into one

-NUM specify the number of lines per screenful.

+NUM display file beginning from line number NUM

+/STRING display file beginning from search string match

-V output version information and exit

date Command

print or set the system date and time

Syntax: date [OPTION]... [+FORMAT]

date [-u|--utc|--universal] [MMDDhhmm[[CC]YY][.ss]]

Options: -d, --date=STRING display time described by STRING, not `now'

-f, --file=DATEFILE like --date once for each line of DATEFILE

-r, --reference=FILE display the last modification time of FILE

Examples:

$ date --date='@2147483647' :Convert seconds since the epoch (1970-01-01 UTC) to a date.

$ TZ='America/Los_Angeles' date :Show the time on the west coast of the US (use tzselect(1) to find TZ).

$ date --date='TZ="America/Los_Angeles" 09:00 next Fri' :Show the local time for 9AM next Friday on the west coast of the US.

<u>time Command</u>

Run programs and summarize system resource usage

Syntax: time [-apqvV] [-f FORMAT]

Options: [-o FILE]-o FILE, --output=FILE

Write the resource use statistics to FILE instead of to the standard error stream. By default, this overwrites the file, destroying the file's previous contents. This option is useful for collecting information on interactive programs and programs that produce output on the standard error stream.

-a, --append

Append the resource use information to the output file instead of overwriting it. This option is only useful with the `-o' or `--output' option.

<u>kill Command</u>

Send a signal to a process.The default signal for kill is SIGTERM

Syntax: kill [-signal | -s signal] pid ...

kill [-L | -V, --version]

kill -l [signal]

Examples kill -9 -1

Kill all processes you can kill.

kill -l 11

Translate number 11 into a signal name.

kill -L

List the available signal choices in a nice table.

kill 123 543 2341 3453 11

Send the default signal, SIGTERM, to all those processes.

Name	**Num**	**Action**	**Description**
0	0	n/a	exit code indicates if a signal may be sent
ALRM	14	exit	
HUP	1	exit	
INT	2	exit	
KILL	9	exit	cannot be blocked
PIPE	13	exit	
POLL		exit	
PROF		exit	
TERM	15	exit	
USR1		exit	
USR2		exit	

history Command

It displays commands executed by users

Syntax: history

Example: history

finger Command

User information lookup program. The **finger** displays information about the system users.

Syntax:

finger [-**lmsp**] [*user* ...] [*user@host* ...]

Options:

-s Display's the user's login name,real name,terminal name & write status,idle time,login time,office location & office phone number

chmod Command

The **chmod** command changes the access mode of one file or multiple files.

Syntax:

```
chmod  [option]  mode  files
```

Options:

-R	Descend directory arguments recursively while setting modes.

Mode

Mode	Description
Who	u=user,g=group,o=other,a=all(default)
opcode	+ means add permission
	- means remove permission
	= means assign permission & remove the permission of unspecified fields.

Example:

```
chmod 751 tech
chmod u=rwx,g=rx,o=x tech
chmod=r tech
```

chown Command

Change files owner and group. **chown** changes the user and/or group ownership of each given file.

Syntax:

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

Options:

Change the owner and/or group of each FILE to OWNER and/or GROUP. With **--reference**, change the owner and group of each FILE to those of RFILE.

-c, --changes

like verbose but report only when a change is made

--no-preserve-root

do not treat `/' specially (the default)

--preserve-root

fail to operate recursively on `/'

-f, --silent, --quiet

Suppress most error messages

-R, --recursive

Operate on files and directories recursively

-v, --verbose

Output a diagnostic for every file processed

<u>pwd Command</u>

Print name of current/working directory

Syntax: pwd [OPTION]...

Example: pwd -L

Options:-L, --logical use PWD from environment, even if it contains symlinks.

-P, --physical avoid all symlinks

--help display this help and exit

--version output version information and exit

<u>cal Command</u>

Displays a calendar and the date of Easter. If arguments are not specified, the current month is displayed.

Syntax: cal [-hjy] [-A number] [-B number] [[month] year]

Examples: cal -hj 09 2015

cal -hjy

Options: -A number

Display the number of months after the current month.

-B number

Display the number of months before the current month.

-C Switch to cal mode.

-y Display a calendar for the specified year.

-h Turns off highlighting of today.

-J Display Julian Calendar, if combined with the -e option, display date of Easter according to the Julian Calendar.

login, logout Command

write utmp and wtmp entries.

The utmp file records who is currently using the system. The wtmp file records all logins and logouts.

The function login() takes the supplied struct utmp, ut, and writes it to both the utmp and the wtmp file. The function logout() clears the entry in the utmp file again.

shutdown Command

Bring the system down

Syntax: shutdown [OPTION]... TIME [MESSAGE]

Example: shutdown -h 17:25 shutdown -r +5

Options: -r Requests that the system be rebooted after it has been brought down.

-h Requests that the system be either halted or powered off after it has been brought down, with the choice as to which left up to the system.

-c Cancels a running shutdown. TIME is not specified with this option, the first argument is MESSAGE.

1.b) Study of vi editor.

The vi editor is available on almost all Unix systems. vi can be used from any type of terminal because it does not depend on arrow keys and function keys--it uses the standard alphabetic keys for commands.

vi (pronounced "vee-eye") is short for "vi"sual editor. It displays a window into the file being edited that shows 24 lines of text. vi is a text editor, not a "what you see is what you get" word processor. vi lets you add, change, and delete text, but does not provide such formatting capabilities as centering lines or indenting paragraphs.

This help note explains the basics of vi:

- opening and closing a file
- moving around in a file
- elementary editing

===== Starting vi =====

You may use vi to open an already existing file by typing

vi filename

where "filename" is the name of the existing file. If the file is not in your current directory, you must use the full pathname. Or you may create a new file by typing vi newname where "newname" is the name you wish to give the new file.

To open a new file called "testvi," enter vi testvi

On-screen, you will see blank lines, each with a tilde (~) at the left, and a line at the

bottom giving the name and status of the new file:

```
~

~

"testvi" [New file]
```

===== vi Modes =====

vi has two modes:

- command mode
- insert mode

In command mode, the letters of the keyboard perform editing functions (like moving the cursor, deleting text, etc.). To enter command mode, press the escape &<Esc> key.

In insert mode, the letters you type form words and sentences. Unlike many word Processors, vi starts up in command mode.

===== Entering Text =====

In order to begin entering text in this empty file, you must change from command mode to insert mode. To do this, type i nothing appears to change, but you are now in insert mode and can begin typing text. In general, vi's commands do not display on the screen and do not require the Return key to be pressed.

Type a few short lines and press &<Return> at the end of each line. If you type a long line, you will notice the vi does not word wrap, it merely breaks the line unceremoniously at the edge of the screen. If you make a mistake, pressing <Backspace> or <Delete> may remove the error, depending on your terminal type.

===== Moving the Cursor =====

To move the cursor to another position, you must be in command mode. If you have just

finished typing text, you are still in insert mode. Go back to command mode by pressing

<Esc>.

If you are not sure which mode you are in, press <Esc> once or twice until you hear a beep.

When you hear the beep, you are in command mode.

The cursor is controlled with four keys: h, j, k, l.

Key Cursor Movement

h left one space

j down one line

k up one line

l right one space

When you have gone as far as possible in one direction, the cursor stops moving and you hear a beep. For example, you cannot use l to move right and wrap around to the next line, you must use j to move down a line. See the section entitled "Moving Around in a File" for ways to move more quickly through a file.

Basic Editing

Editing commands require that you be command mode. Many of the editing commands have a different function depending on whether they are typed as upper- or lowercase. Often, editing commands can be preceded by a number to indicate a repetition of the command.

Deleting Characters

To delete a character from a file, move the cursor until it is on the incorrect letter, then

type x’

The character under the cursor disappears. To remove four characters (the one under the cursor and the next three) type 4x

To delete the character before the cursor, type X (uppercase)

Deleting Words

To delete a word, move the cursor to the first letter of the word, and type dw

This command deletes the word and the space following it. To delete three words type 3dw

Deleting Lines

To delete a whole line, type dd

The cursor does not have to be at the beginning of the line. Typing dd deletes the entire line containing the cursor and places the cursor at the start of the next line. To delete two lines, type 2dd. To delete from the cursor position to the end of the line, type D (uppercase)

Replacing Characters

To replace one character with another:

1. Move the cursor to the character to be replaced.

2. Type r

3. Type the replacement character.

The new character will appear, and you will still be in command mode.

Replacing Words

To replace one word with another, move to the start of the incorrect word and type cw

The last letter of the word to be replaced will turn into a $. You are now in insert mode and may type the replacement. The new text does not need to be the same length as the original.

Press <Esc> to get back to command mode. To replace three words, type 3cw

Replacing Lines

To change text from the cursor position to the end of the line:

1. Type C (uppercase).

2. Type the replacement text.

3. Press <Esc>.

Inserting Text

To insert text in a line:

1. Position the cursor where the new text should go.

2. Type i

3. Enter the new text. The text is inserted BEFORE the cursor.

4. Press <Esc> to get back to command mode.

Appending Text

To add text to the end of a line:

1. Position the cursor on the last letter of the line.

2. Type a

3. Enter the new text. This adds text AFTER the cursor.

4. Press <Esc> to get back to command mode.

Opening a Blank Line

To insert a blank line below the current line, type o (lowercase)

To insert a blank line above the current line, type O (uppercase)

Joining Lines

To join two lines together:

1. Put the cursor on the first line to be joined.

2. Type J

To join three lines together:

1. Put the cursor on the first line to be joined.

2. Type 3J

===== Undoing =====

To undo your most recent edit, type u

To undo all the edits on a single line, type U (uppercase)

Undoing all edits on a single line only works as long as the cursor stays on that line.

Once you move the cursor off a line, you cannot use U to restore the line.

===== Moving Around in a File =====

There are shortcuts to move more quickly though a file. All these work in command mode.

Key Movement

w forward word by word

b backward word by word

$ to end of line

0 (zero) to beginning of line

H to top line of screen

M to middle line of screen

L to last line of screen

G to last line of file

1G to first line of file

<Control>f scroll forward one screen

<Control>b scroll backward one screen

<Control>d scroll down one-half screen

<Control>u scroll up one-half screen

===== Moving by Searching =====

To move quickly by searching for text, while in command mode:

1. Type / (slash).
2. Enter the text to search for.
3. Press <Return>.

The cursor moves to the first occurrence of that text.

To repeat the search in a forward direction, type n

To repeat the search in a backward direction, type N

===== Closing and Saving a File =====

With vi, you edit a copy of the file, rather than the original file. Changes are made to the Original only when you save your edits.

To save the file and quit vi, type ZZ

The vi editor is built on an earlier Unix text editor called ex. ex commands can be used with in vi. ex commands begin with a : (colon) and end with a <Return>. The command is displayed on the status line as you type. Some ex commands are useful when saving and closing files.

To save the edits you have made, but leave vi running and your file open:

1. Press <Esc>.

2. Type :w

3. Press <Return>.

To quit vi, and discard any changes your have made since last saving:

1. Press <Esc>.

2. Type :q!

3. Press <Return>.

1. c) Study of Bash shell, Bourne shell and C shell in Unix/Linux operating system.

Types of Shells in Linux

In addition to graphical user interfaces like Gnome, KDE and MATE, the Linux operating system also offers several shells. These command-line interfaces provide powerful environments for software development and system maintenance. Though shells have many commands in common, each type has unique features. Over time, individual programmers come to prefer one type of shell over another; some develop new, enhanced shells based on previous ones. UNIX also has an ecosystem of different shells; Linux carries this practice into the open-source software arena.

The Bourne shell

The Bourne shell, called "sh," is one of the original shells, developed for Unix computers by Stephen Bourne at AT&T's Bell Labs in 1977. Its long history of use means many software developers are familiar with it. It offers features such as input and output redirection, shell scripting with string and integer variables, and condition testing and looping.

The Bash shell

The popularity of sh motivated programmers to develop a shell that was compatible with it, but with several enhancements. Linux systems still offer the sh shell, but "bash" – the "Bourne-again Shell," based on sh -- has become the new default standard. One attractive feature of bash is its ability to run sh shell scripts unchanged. Shell scripts are complex sets of commands that automate programming and maintenance chores; being able to reuse these scripts saves programmers time. Conveniences not present with the original Bourne shell include command completion and a command history.

C Shell

Developers have written large parts of the Linux operating system in the C and C++languages.

Using C syntax as a model, Bill Joy at Berkeley University developed the "C-shell," csh,in 1978. Ken Greer, working at Carnegie-Mellon University, took csh concepts a step forward with a new shell, tcsh, which Linux systems now offer. Tcsh fixed problems in csh and added command completion, in which the shell makes educated "guesses" as you type, based on your system's directory structure and files. Tcsh does not run bash scripts, as the two have substantial differences.

The Korn shell

David Korn developed the Korn shell, or ksh, about the time tcsh was introduced. Ksh is compatible with sh and bash. Ksh improves on the Bourne shell by adding floating-point arithmetic, job control, and command aliasing and command completion. AT&T held proprietary rights to ksh until 2000, when it became open source.

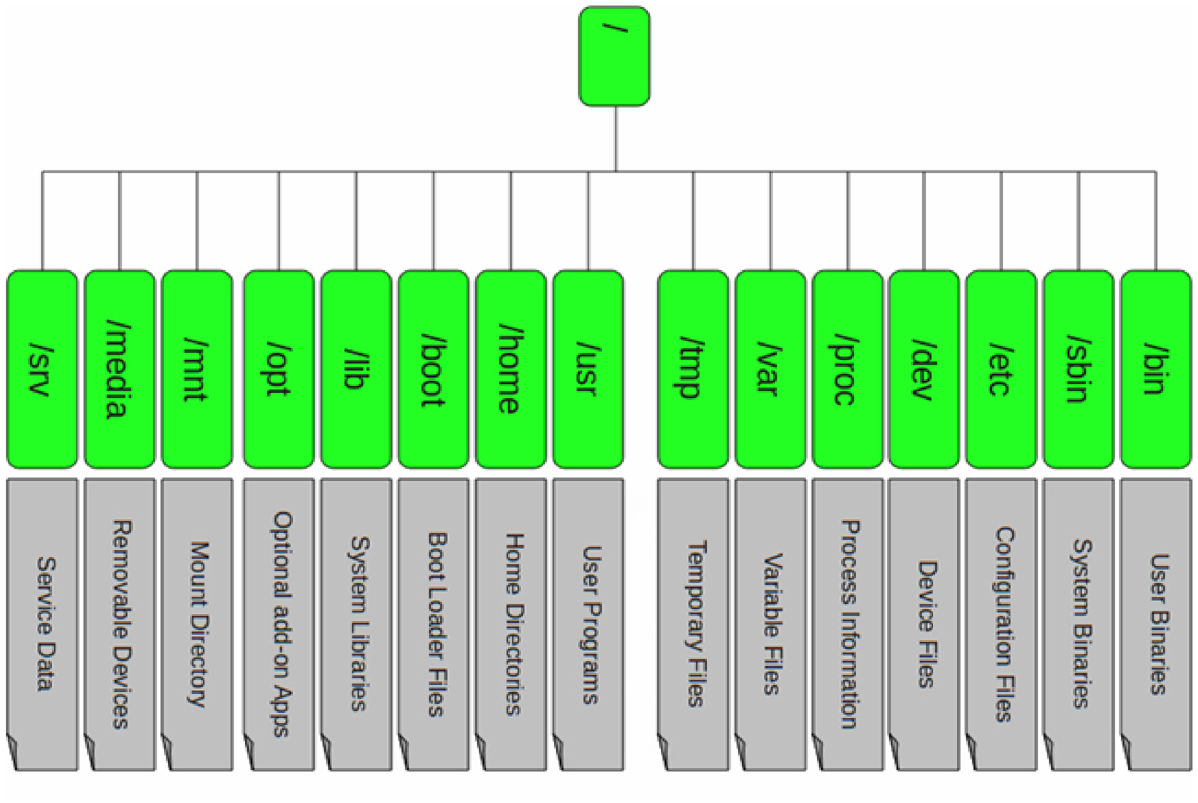
1. d) Study of Unix/Linux file system (tree structure).

A file system is a logical collection of files on a partition or disk. UNIX uses a hierarchical file system structure, much like an upside-down tree, with root (/) at the base of the file system and all other directories spreading from there.

A UNIX file system is a collection of files and directories that has the following properties −

It has a root directory (/) that contains other files and directories.

- Each file or directory is uniquely identified by its name, the directory in which it resides, and a unique identifier, typically called an inode.
- By convention, the root directory has an inode number of 2 and the lost+found directory has an inode number of 3. Inode numbers 0 and 1 are not used. File inode numbers can be seen by specifying the -i option to ls command.
- It is self-contained. There are no dependencies between one filesystem and any other. The directories have specific purposes and generally hold the same types of information for easily locating files. Following are the directories that exist on the major versions of Unix −



1. /bin

The /bin directory is for User Binaries. It is where many of the most common Linux commands are stored. Specifically, this is where the single user mode binaries are stored.

2. /sbin

This directory is almost exactly like the /bin directory, with one exception. The binaries here are primarily used by Administrators for system maintenance.

3. /etc

The configuration files for your programs and operating system are stored in /etc.

4. /dev

This is where all of the device files are located. For example, this is the directory that you would call to in order to mount a drive with a command like: mount /dev/sda2 /mnt/backup

5. /proc

The /proc directory is one of the most interesting in the whole Linux File System. It is actually its own virtual file system with a massive amount of text information about system processes.

6. /var

This is where all of the variable files are stored. Most commonly, this is where log files and web server files are stored.

7. /tmp

These are simply temporary files.

8. /usr

Programs installed by single users get stored here.

9. /home

This is where all of the user home directories are except for the root user's home directory which is /root.

10. /boot

The files that make up the boot loader go in /boot. Everything from boot loader menus, to the actual kernel files are stored here.

11. /lib

All of the binary files that are located in /bin and /sbin are supported by the library files located in /lib.

12. /opt

/opt is short for “optional”. It is the directory where individual vendors can install optional add-on software for the operating system.

13. /mnt

The /mnt directory is the mount point that system administrators can use to mount file systems temporarily.

14. /media

The /media directory serves the same purpose as the /mnt directory except it is specifically for removable devices and can be used by non-administrators.

15. /srv

The /srv directory contains server specific service files.

1.e) Study of .bashrc, /etc/bashrc and Environment variables.

Following is the partial list of important environment variables.

Variable	**Description**
DISPLAY	Contains the identifier for the display that X11 programs should use by default.
HOME	Indicates the home directory of the current user: the default argument for the cd built-in command.
IFS	Indicates the Internal Field Separator that is used by the parser for word splitting after expansion.
LANG	LANG expands to the default system locale; LC_ALL can be used to override this. For example, if its value is pt_BR, then the languageis set to (Brazilian) Portuguese and the locale to Brazil.
LD_LIBRARY_PATH	On many Unix systems with a dynamic linker, contains a colon separated list of directories that the dynamic linker should search for shared objects when building a process image after exec, before searching in any other directories.
PATH	Indicates search path for commands. It is a colon-separated list of directories in which the shell looks for commands.
PWD	Indicates the current working directory as set by the cd command.
RANDOM	Generates a random integer between 0 and 32,767 each time it is referenced.
SHLVL	Increments by one each time an instance of bash is started. This variable is useful for determining whether the built-in exit command ends the current session.
TERM	Refers to the display type
TZ	Refers to Time zone. It can take values like GMT, AST, etc.
UID	Expands to the numeric user ID of the current user, initialized at shell startup.

Experiment -2

2. Write a C program that makes a copy of a file using a) standard I/O, and b) system calls

a) Using Standard I/O

```
#include <fcntl.h>
#include <sys/stat.h>
#include <stdio.h>
int main(int argc, char **argv)
 {
   FILE *fp,*fp1;
   int n,count=0;
   char buf;
   fp=fopen(argv[1],"r");
   fp1=fopen(argv[2],"w");
   while(!feof(fp))
   {
   buf=fgetc(fp);
   fputc(buf,fp1);
   }
   fclose(fp);
   fclose(fp1);
    return (0);
}
```

OUTPUT:

```
 [satyakumari@localhost linux]$ ls
2         cp1.c        f2           pipe.c       shared.c   z1
2.c       cp.c         pipedif.c    shm1.c                  z2
a.out     cpsys.c      lsl.c        pipenew.c    shm2.c     z3
```

```
[satyakumari@localhost linux]$ cat z1
THIS IS LINUX LAB
THIS IS LAB3
[satyakumari@localhost linux]$ cc cpsys.c
[satyakumari@localhost linux]$ ./a.out z1 zz
[satyakumari@localhost linux]$ cat zz
THIS IS LINUX LAB
THIS IS LAB3
```

b) Using System calls

```
#include<fcntl.h>
#include<stdio.h>
#include<unistd.h>
#include<sys/stat.h>
int main(int argc, char *argv[])
{
int fd1,fd2;
int n;
char buf;
fd1=open(argv[1],O_RDONLY);
fd2=creat(argv[2],S_IWUSR|S_IRUSR);
while((n=read(fd1,&buf,1))>0)
{
write(fd2,&buf,1);
}
return (0);
}
```

OUTPUT:

```
[satyakumari@localhost linux]$ ls
2        cp1.c        f2                      pipe.c          shared.c    z1
2.c      cp.c        pipedif.c            shm1.c                          z2
a.out   cpsys.c  lsl.c                      pipenew.c     shm2.c      z3
 [satyakumari@localhost linux]$ cat z1
THIS IS LINUX LAB
THIS IS LAB3
[satyakumari@localhost linux]$ cc cp.c
[satyakumari@localhost linux]$ ./a.out z1 zz
[satyakumari@localhost linux]$ cat zz
THIS IS LINUX LAB
THIS IS LAB3
```

Experiment -3

3. Write a C program to emulate the UNIX ls –l command.

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <stdlib.h>
int main()
{
int pid;            //process id
pid=fork();     //create another process
if ( pid<0 )
{                              //fail
printf("Fork failed");
exit (-1);
}
else if ( pid == 0 )
{                    //child
execlp ( "/bin/ls", "ls", "-l", NULL );  //execute ls
}
else
{                           //parent
wait (NULL);              //wait for child
printf("child complete");
exit (0);
}
}
```

OUTPUT:

```
[satyakumari@localhost linux]$ cc lsl.c
[satyakumari@localhost linux]$ ./a.out
total 92
-rw-rw-r-- 1 satyakumari satyakumari    0 Jul 23 11:54 2
-rw------- 1 satyakumari satyakumari  275 Jul  6  2015 2.c
-rwxrwxr-x 1 satyakumari satyakumari 5121 Sep 12 11:48 a.out
-rw-rw-r-- 1 satyakumari satyakumari  368 Jul  6  2015 cat.c
-rw-rw-r-- 1 satyakumari satyakumari  267 Jul  6  2015 cp1.c
-rw-rw-r-- 1 satyakumari satyakumari  275 Jul  6  2015 cp.c
-rw-rw-r-- 1 satyakumari satyakumari  314 Jul 27 11:01 cpsys.c
-rw-rw-r-- 1 satyakumari satyakumari   31 Jul  6  2015 f1
-rw------- 1 satyakumari satyakumari   31 Jul  6  2015 f2
-rw------- 1 satyakumari satyakumari    0 Sep 12 11:37 HOSTNAME=localhost.localdomain
-rw-rw-r-- 1 satyakumari satyakumari  515 Aug 17 11:38 lsl.c
-rw-rw-r-- 1 satyakumari satyakumari  254 Aug 17 11:45 pipe1.c
-rw-rw-r-- 1 satyakumari satyakumari  405 Aug 17 11:43 pipe.c
-rw-rw-r-- 1 satyakumari satyakumari  341 Aug 17 12:18 pipedif.c
-rw-rw-r-- 1 satyakumari satyakumari  401 Aug 17 11:47 pipenew.c
-rw-rw-r-- 1 satyakumari satyakumari  258 Aug 18 11:41 pipeorg.c
-rw-rw-r-- 1 satyakumari satyakumari 1256 Aug 19 11:54 shared.c
-rw-rw-r-- 1 satyakumari satyakumari  932 Aug 19 11:39 shm1.c
-rw-rw-r-- 1 satyakumari satyakumari  696 Aug 19 11:58 shm2.c
-rw-rw-r-- 1 satyakumari satyakumari 1256 Aug 19 11:51 shmemory.c
-rw-rw-r-- 1 satyakumari satyakumari   32 Aug 17 11:01 z1
-rw-rw-r-- 1 satyakumari satyakumari   32 Jul 27 10:30 z2
-rw-rw-r-- 1 satyakumari satyakumari   33 Jul 27 10:33 z3
-rw-rw-r-- 1 satyakumari satyakumari   32 Sep 12 11:43 zz
child complete
```

Experiment -4

4. Write a C program that illustrates how to execute two commands concurrently with a command pipe. Ex: - ls –l | sort

Description:

A pipe is created by calling a pipe() function.

int pipe(int filedesc[2]);

It returns a pair of file descriptors filedesc[0] is open for reading and filedesc[1] is open for writing. This function returns a 0 if ok & -1 on error.

```
#include<stdio.h>
#include<fcntl.h>
main()
{
int pfd[2],p;
pipe(pfd);
p=fork();
if(p==0)//for child
{
close(pfd[0]);
close(1);
dup(pfd[1]);
execlp("ls","ls","-l",(char *)0);
}
else
{
close(pfd[1]);
close(0);
dup(pfd[0]);
execlp("sort","sort",(char *)0);
}
}
```

OUTPUT:

```
[satyakumari@localhost linux]$ cc pipeorg.c

 [satyakumari@localhost linux]$ ./a.out

drwxrwxr-x 2 satyakumari satyakumari 4096 Sep 12 11:51 a

-rw------- 1 satyakumari satyakumari    0 Sep 12 11:37 HOSTNAME=localhost.localdomain

-rw------- 1 satyakumari satyakumari  275 Jul  6  2015 2.c

-rw------- 1 satyakumari satyakumari   31 Jul  6  2015 f2

-rw-rw-r-- 1 satyakumari satyakumari    0 Jul 23 11:54 2

-rw-rw-r-- 1 satyakumari satyakumari 1256 Aug 19 11:51 shmemory.c

-rw-rw-r-- 1 satyakumari satyakumari 1256 Aug 19 11:54 shared.c

-rw-rw-r-- 1 satyakumari satyakumari  254 Aug 17 11:45 pipe1.c

-rw-rw-r-- 1 satyakumari satyakumari  258 Aug 18 11:41 pipeorg.c

-rw-rw-r-- 1 satyakumari satyakumari  267 Jul  6  2015 cp1.c

-rw-rw-r-- 1 satyakumari satyakumari  275 Jul  6  2015 cp.c

-rw-rw-r-- 1 satyakumari satyakumari  314 Jul 27 11:01 cpsys.c

-rw-rw-r-- 1 satyakumari satyakumari   31 Jul  6  2015 f1

-rw-rw-r-- 1 satyakumari satyakumari   32 Aug 17 11:01 z1

-rw-rw-r-- 1 satyakumari satyakumari   32 Jul 27 10:30 z2

-rw-rw-r-- 1 satyakumari satyakumari   32 Sep 12 11:43 zz

-rw-rw-r-- 1 satyakumari satyakumari   33 Jul 27 10:33 z3

-rw-rw-r-- 1 satyakumari satyakumari  341 Aug 17 12:18 pipedif.c

-rw-rw-r-- 1 satyakumari satyakumari  368 Jul  6  2015 cat.c
```

```
-rw-rw-r-- 1 satyakumari satyakumari  401 Aug 17 11:47 pipenew.c

-rw-rw-r-- 1 satyakumari satyakumari  405 Aug 17 11:43 pipe.c

-rw-rw-r-- 1 satyakumari satyakumari  538 Sep 12 11:54 lsl.c

-rw-rw-r-- 1 satyakumari satyakumari  696 Aug 19 11:58 shm2.c

-rw-rw-r-- 1 satyakumari satyakumari  932 Aug 19 11:39 shm1.c

-rwxrwxr-x 1 satyakumari satyakumari 5139 Sep 12 11:57 a.out

total 96
```

Experiment -5

Simulate the following CPU scheduling algorithms

a) FCFS b) SJF c) Priority d) Round Robin

a) AIM: To write a 'C' Program to simulate First Come First Served CPU Scheduling (FCFS) Algorithm.

```
main()
{
int n,i,twt,ttt;
char p[20][20];
int b[20],w[20],tt[20];
float awt,att;
printf("Enter no.of process:");
scanf("%d",&n);
for(i=1;i<=n;i++)
{
 printf("Enter process:");
 scanf("%s",p[i]);
 printf("Enter burst time:");
 scanf("%d",&b[i]);
 }
 w[1]=0;
 tt[1]=w[1]+b[1];
 twt=0;
 ttt=tt[1];
 for(i=2;i<=n;i++)
  {
  w[i]=w[i-1]+b[i-1];
  tt[i]=w[i]+b[i];
  twt=w[i]+twt;
  ttt=tt[i]+ttt;
  }
  printf("sno\tprocess name \tburst time \twaiting time \tturn around time\n");
  for(i=1;i<=n;i++)
  {
   printf("%d\t%s\t\t%d\t\t%d\t\t%d\t\n",i,p[i],b[i],w[i],tt[i]);
   }
   printf("Total waiting time: %d\n",twt);
   printf("Total turn around time: %d\n",ttt);
   printf("Avg waiting time: %f\n",(float)twt/n);
   printf("Avg turn around time: %f\n",(float)ttt/n);
     }
```

OUTPUT:

```
Enter no.of process:3
Enter process:a1
Enter burst time:3
Enter process:a2
Enter burst time:4
Enter process:a3
Enter burst time:6
s.no     process name    burst time      waiting time    turn around time
1              a1                  3                 0                  3
2              a2                  4                 3                  7
3              a3                  6                 7                  13
Total waiting time: 10
Total turn around time: 23
Avg waiting time: 3.333333
Avg turn around time: 7.666667
```

b) AIM: To write a 'C' Program to simulate Shortest Job First CPU scheduling (SJF) Algorithm.

```
main()
{
int n,i,twt,ttt,k,j,temp;
char p[20][20],c[20];
int b[20],w[20],tat[20];
float awt,att;
printf("Enter no.of process:");
scanf("%d",&n);
for(i=1;i<=n;i++)
{
 printf("Enter process:");
 scanf("%s",p[i]);
 printf("Enter burst time:");
 scanf("%d",&b[i]);
 }
 for(k=1;k<=n-1;k++)
 {
 for(j=1;j<=n-k;j++)
 {
 if(b[j]>b[j+1])
 {
 temp=b[j];
 b[j]=b[j+1];
 b[j+1]=temp;
 strcpy(c,p[j]);
 strcpy(p[j],p[j+1]);
 strcpy(p[j+1],c);
 }
 }
 }
 w[1]=0;
 tat[1]=w[1]+b[1];
 twt=0;
 ttt=tat[1];
 for(i=2;i<=n;i++)
  {
  w[i]=w[i-1]+b[i-1];
  tat[i]=w[i]+b[i];
  twt=w[i]+twt;
  ttt=tat[i]+ttt;
  }
  printf("s.no\tprocessname\tburst\ttwait\ttturn around\n");
  for(i=1;i<=n;i++)
  {
   printf("%d\t%s\t\t%d\t%d\t%d\t\n",i,p[i],b[i],w[i],tat[i]);
   }
   printf("Total waiting time: %d\n",twt);
   printf("Total turn around time: %d\n",ttt);
   printf("Avg waiting time: %f\n",(float)twt/n);
   printf("Avg turn around time: %f\n",(float)ttt/n);
   }
```

OUTPUT:

```
Enter no.of process:3
Enter process:a1
Enter burst time:5
Enter process:a2
Enter burst time:3
Enter process:a3
Enter burst time:6
s.no    processname     burst   twait   tturn around
1           a2                 3        0         3
2           a1                 5        3          8
3          a3                  6       8             14
Total waiting time: 11
Total turn around time: 25
Avg. waiting time: 3.666667
Avg. turn around time: 8.333333
```

c) AIM: To write a ‘C’ Program to simulate Priority CPU scheduling Algorithm.

```
#include<stdio.h>
#include<string.h>
void main()
{
int n,i,twt,ttt,k,j,temp;
char p[20][20],c[20];
int b[20],w[20],tat[20],pr[20];
float awt,att;
printf("Enter no.of process:");
scanf("%d",&n);
for(i=1;i<=n;i++)
{
 printf("Enter process:");
 scanf("%s",p[i]);
 printf("Enter burst time:");
 scanf("%d",&b[i]);
 printf("Enter priority:");
 scanf("%d",&pr[i]);
 }
 for(k=1;k<=n-1;k++)
 {
 for(j=1;j<=n-k;j++)
 {
 if(pr[j]>pr[j+1])
 {
 temp=pr[j];
 pr[j]=pr[j+1];
 pr[j+1]=temp;
 temp=b[j];
 b[j]=b[j+1];
 b[j+1]=temp;
 strcpy(c,p[j]);
 strcpy(p[j],p[j+1]);
 strcpy(p[j+1],c);
 }
 }
 }
 w[1]=0;
 tat[1]=w[1]+b[1];
 twt=0;
 ttt=tat[1];
 for(i=2;i<=n;i++)
  {
  w[i]=w[i-1]+b[i-1];
  tat[i]=w[i]+b[i];
  twt=w[i]+twt;
  ttt=tat[i]+ttt;
  }
  printf("s.no\tpname\tburst\tpriority\ttwait\ttturn around\n");
  for(i=1;i<=n;i++)
  {
   printf("%d\t%s\t%d\t\t%d\t%d\t%d\n",i,p[i],b[i],pr[i],w[i],tat[i]);
   }
   printf("Total waiting time: %d\n",twt);
   printf("Total turn around time: %d\n",ttt);
   printf("Avg waiting time: %f\n",(float)twt/n);
   printf("Avg turn around time: %f\n",(float)ttt/n);
   }
```

OUTPUT:

```
Enter no.of process: 3
Enter process:A1
Enter burst time:5
Enter priority:3
Enter process:A2
Enter burst time:4
Enter priority:1
Enter process:A3
Enter burst time:3
Enter priority:2
s.no   pname   burst     priority        twait   tturn around
1        A2       4           1               0      4
2        A3      3            2               4      7
3        A1      5            3               7      12
Total waiting time 11
Total turn around time 23
Avg. waiting time 3.666667
Avg. turn around time 7.666667
```

d) AIM: To write a ‘C’ Program to simulate Round-Robin (RR) CPU scheduling Algorithm.

```
#include<stdio.h>

#include<string.h>

 int main()

{

 char p[10][5];

 int et[10],wt[10],timer,count,pt[25],rt,i,j,totwt=0,n,found=0,m,tat[10],ttt=0;

 float avgwt;

 printf("Enter the no of  process:\n");

 scanf("%d",&n);

 printf("Enter time slice:\n");

 scanf("%d",&timer);

 for(i=0;i<n;i++)

 {

  printf("Enter the process name:\n");

  scanf("%s",&p[i]);

  printf("Enter the process time:\n");

  scanf("%d",&pt[i]);

 }

 m=n;

 wt[0]=0;

 i=0;

 do

 {
```

```
 if(pt[i]>timer)
 {
  rt=pt[i]-timer;
  strcpy(p[n],p[i]);
  pt[n]=rt;
  et[i]=timer;
  n++;
 }
else
 {
  et[i]=pt[i];
 }
i++;
wt[i]=wt[i-1]+et[i-1];
}
while(i<n);
count=0;
for(i=0;i<n;i++)
{
 for(j=i+1;j<=n;j++)
 {
  if(strcmp(p[i],p[j])==0)
   {
    count++;
    found=j;
   }
  if(found!=0)
  {
   wt[i]=wt[found]-(count*timer);
   count=0;
   found=0;
  }
 }
}
 for(i=0;i<m;i++)
 {
  totwt+=wt[i];
  tat[i]=wt[i]+pt[i];
  ttt=ttt+tat[i];
 }
 avgwt=(float)totwt/m;
 printf("pname \t ptime \t wtime \tturnaroundtime\n");
 for(i=0;i<m;i++)
 {
  printf("\n %s \t %d \t %d \t\t %d",p[i],pt[i],wt[i],tat[i]);
 }
 printf("\n Total waiting time: %d \n",totwt);
 printf("\n Avg wating time: %f",avgwt);
 printf("\n Total turnaroundtime: %d",ttt);
 printf("\n Avg turnaround time: %f",(float)ttt/m);
}
```

OUTPUT:

```
Enter the number of Processes:3

Enter time slice:4

Enter the process name:A1

Enter the process time:4

Enter the process name:A2

Enter the process time:6

Enter the process name:A3

Enter the process time:3

pname    ptime   wtime  turnaroundtime

 A1        4        0          4

 A2        6        7         13

 A3        3        8         11

 Total waiting time: 15

 Avg.wating time: 5.000000

Total turnaroundtime: 28

 Avg.turnaround time: 9.333333
```

Experiment -6

Multiprogramming-Memory management- Implementation of fork (), wait (), exec() and exit (), System calls

```
#include <stdio.h>

#include <stdlib.h>

#include <unistd.h> /* for fork */

#include <sys/types.h> /* for pid_t */

#include <sys/wait.h> /* for wait */

int main(int argc,char** argv)

{

/*Spawn a child to run the program.*/

 pid_t pid=fork();

 if (pid==0)

 { /* child process */

 execv("/bin/ls",argv);

 exit(127); /* only if execv fails */

 }

 else

 { /* pid!=0; parent process */

printf("\nWaiting Child process to finish");

 //waitpid(pid,0,0); /* wait for child to exit */

 wait(NULL);

 }

 printf("\nExiting main process\n");
return 0;

}
```

Output:

```
ui

dsfg

gfh

tyrert

yuk

sdg

cat p

ui

dsfg

gfh

tyrert

yuk

sdg
```

Experiment -7

Simulate the following

a) Multiprogramming with a fixed number of tasks (MFT)

b) Multiprogramming with a variable number of tasks (MVT)

a) AIM: To write a ‘C’ Program to simulate Multiprogramming with Fixed number of tasks.

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int ms,i,ps[20],n,size,p[20],s,intr=0;
        printf("Enter size of memory:");
        scanf("%d",&ms);
        printf("Enter memory for OS:");
        scanf("%d",&s);
        ms-=s;
        printf("Enter no.of partitions to be divided:");
        scanf("%d",&n);
        size=ms/n;
        for(i=0;i<n;i++)
        {
                printf("Enter process and process size:");
                scanf("%d%d",&p[i],&ps[i]);
                if(ps[i]<=size)
                {
                        intr=intr+size-ps[i];
                        printf("process%d is allocated\n",p[i]);
                }
                else
                        printf("process%d is blocked\n",p[i]);
        }
        printf("\nTotal fragmentation is: %d",intr);
}
```

OUTPUT:

```
Enter size of memory: 50
Enter memory for OS: 10
Enter no.of partitions to be divided: 4
Enter process and process size: 1 6
process1 is allocated
Enter process and process size:2 10
process2 is allocated
Enter process and process size:3 12
process3 is blocked
Enter process and process size:4 5
process4 is allocated

Total fragmentation is: 9
```

b) AIM: To write a ‘C’ Program to simulate Multiprogramming with variable number of tasks.

```
#include<stdio.h>
#include<conio.h>
main()
{
        int i,m,n,tot,s[20];
        clrscr();
        printf("Enter total memory size:");
        scanf("%d",&tot);
        printf("Enter no. of pages:");
        scanf("%d",&n);
        printf("Enter memory for OS:");
        scanf("%d",&m);
        for(i=0;i<n;i++)
        {
                printf("Enter size of page%d:",i+1);
                scanf("%d",&s[i]);
        }
        tot=tot-m;
        for(i=0;i<n;i++)
        {
                if(tot>=s[i])
                {
                        printf("Allocate page %d\n",i+1);
```

```
                    tot=tot-s[i];
                }
                else
                    printf("process p%d is blocked\n",i+1);
            }
            printf("External Fragmentation is=%d",tot);
}
```

OUTPUT:

```
Enter total memory size:50
Enter no. of pages:10
Enter memory for OS:10
Enter size of page1:5
Enter size of page2:6
Enter size of page3:3
Enter size of page4:6
Enter size of page5:2
Enter size of page6:7
Enter size of page7:3
Enter size of page8:4
Enter size of page9:2
Enter size of page10:5
Allocate page 1
Allocate page 2
Allocate page 3
Allocate page 4

Allocate page 5

Allocate page 6

Allocate page 7

Allocate page 8

Allocate page 9

process p10 is blocked

External Fragmentation is=2
```

Experiment -8

Simulate Bankers Algorithm for Dead Lock Avoidance

AIM: Write a C program to Simulate Bankers Algorithm for Dead Lock Avoidance

```
#include<stdio.h>

int max[100][100];

int alloc[100][100];

int need[100][100];

int avail[100];

int n,r;

void input();

void show();

void cal();

int main()

{

        int i,j;

        printf("********** Banker's Algorithm ************\n");

        input();

        show();

        cal();

}

void input()

{

        int i,j;

        printf("Enter the no of Processes\t");

        scanf("%d",&n);

        printf("Enter the no of resources instances\t");

        scanf("%d",&r);

        printf("Enter the Max Matrix\n");
        for(i=0;i<n;i++)
                for(j=0;j<r;j++)
                        scanf("%d",&max[i][j]);
        printf("Enter the Allocation Matrix\n");
        for(i=0;i<n;i++)
                for(j=0;j<r;j++)
                        scanf("%d",&alloc[i][j]);
        printf("Enter the available Resources\n");
        for(j=0;j<r;j++)
                scanf("%d",&avail[j]);
}
void show()
{
        int i,j;
        printf("Process\t Allocation\t Max\t Available\t");
        for(i=0;i<n;i++)
        {
                printf("\nP%d\t ",i);
                for(j=0;j<r;j++)
                        printf("%d ",alloc[i][j]);
                printf("\t\t");
                for(j=0;j<r;j++)
                        printf("%d ",max[i][j]);
                printf("\t");
                if(i==0)
                        for(j=0;j<r;j++)
                                printf("%d ",avail[j]);
        }
```

```
}
void cal()
{
        int finish[100],temp,need[100][100],flag=1,k,c1=0;
        int safe[100];
        int i,j;
        for(i=0;i<n;i++)
                finish[i]=0;
        //find need matrix
        for(i=0;i<n;i++)
                for(j=0;j<r;j++)
                        need[i][j]=max[i][j]-alloc[i][j];
        printf("\n");
        while(flag)
        {
                flag=0;
                for(i=0;i<n;i++)
                {
                        int c=0;
                        for(j=0;j<r;j++)
                        {
                                if((finish[i]==0)&&(need[i][j]<=avail[j]))
                                {
                                        c++;
                                        if(c==r)
                                        {
                                                for(k=0;k<r;k++)
                                                    avail[k]+=alloc[i][k];
                                                finish[i]=1;
                                                flag=1;
                                                printf("P%d->",i);
                                        }
                                }
                        }
                }
        }
        for(i=0;i<n;i++)
        {
                if(finish[i]==1)
                        c1++;
                else
                        printf("P%d->",i);
        }
        if(c1==n)
                printf("\n The system is in safe state\n");
        else
        {
                printf("\n Process are in dead lock");
                printf("\n System is in unsafe state");
        }
}
```

OUTPUT:

```
Enter the no of processes 5
Enter the no of resources instances 3
Enter the max matrix
7 5 3
```

```
3 2 2

9 0 2

2 2 2

4 3 3

Enter the allocation matrix

0 1 0

2 0 0

3 0 2

2 1 1

0 0 2

Enter available resources 3 2 2

P1->p3->p4->p2->p0->

The system is in safe state.
```

Experiment -9

Simulate Bankers Algorithm for Dead Lock Prevention

AIM: Write a C program to Simulate Bankers Algorithm for Dead Lock Prevention

```
#include<stdio.h>
#include<conio.h>
void main()
{
 int allocated[15][15],max[15][15],need[15][15],avail[15],tres[15],work[15],flag[15];
 int pno,rno,i,j,prc,count,t,total;
 count=0;
 printf("\n Enter number of process:");
 scanf("%d",&pno);
 printf("\n Enter number of resources:");
 scanf("%d",&rno);
 for(i=1;i<=pno;i++)
 {
  flag[i]=0;
 }
 printf("\n Enter total numbers of each resources:");
 for(i=1;i<=rno;i++)
  scanf("%d",&tres[i]);
 printf("\n Enter Max resources for each process:");
 for(i=1;i<=pno;i++)
 {
  printf("\n for process %d:",i);
  for(j=1;j<=rno;j++)
   scanf("%d",&max[i][j]);
 }
 printf("\n Enter allocated resources for each process:");
```

```
for(i=1;i<=pno;i++)
{
 printf("\n for process %d:",i);
 for(j=1;j<=rno;j++)
  scanf("%d",&allocated[i][j]);
}
printf("\n available resources:\n");
for(j=1;j<=rno;j++)
{
 avail[j]=0;
 total=0;
 for(i=1;i<=pno;i++)
 {
  total+=allocated[i][j];
 }
 avail[j]=tres[j]-total;
 work[j]=avail[j];
 printf("%d \t",work[j]);
}
do
{
for(i=1;i<=pno;i++)
{
 for(j=1;j<=rno;j++)
 {
  need[i][j]=max[i][j]-allocated[i][j];
 }
}
printf("\n Allocated matrix        Max      need");
for(i=1;i<=pno;i++)
{
 printf("\n");
 for(j=1;j<=rno;j++)
 {
  printf("%4d",allocated[i][j]);
 }
 printf("|");
 for(j=1;j<=rno;j++)
 {
  printf("%4d",max[i][j]);
 }
  printf("|");
 for(j=1;j<= rno;j++)
 {
  printf("%4d",need[i][j]);
 }
}
 prc=0;
 for(i=1;i<=pno;i++)
 {
  if(flag[i]==0)
  {
   prc=i;
   for(j=1;j<=rno;j++)
   {
    if(work[j]<need[i][j])
    {
      prc=0;
      break;
    }
   }
  }
  if(prc!=0)
  break;
 }
 if(prc!=0)
 {
  printf("\n Process %d completed",i);
  count++;
  printf("\n Available matrix:");
  for(j=1;j<=rno;j++)
{
   work[j]+=allocated[prc][j];
   allocated[prc][j]=0;
   max[prc][j]=0;
   flag[prc]=1;
   printf("%d\t",work[j]);
  }
 }
}
while(count!=pno&&prc!=0);
 if(count==pno)
  printf("\nThe system is in a safe state!!");
 else
  printf("\nThe system is in an unsafe state!!");
 }
```

OUTPUT:

```
 Enter number of process: 5

 Enter number of resources: 3

 Enter total numbers of each resources: 10 5 7

 Enter Max resources for each process:
```

```
for process 1:7 5 3

for process 2:3 2 2

for process 3:9 0 2

for process 4:2 2 2

for process 5:4 3 3

Enter allocated resources for each process:
for process 1:0 1 0

for process 2:3 0 2

for process 3:3 0 2

for process 4:2 1 1

for process 5:0 0 2

available resources:
2         3         0
Allocated matrix        Max      need
  0   1   0|   7   5   3|   7   4   3
  3   0   2|   3   2   2|   0   2   0
  3   0   2|   9   0   2|   6   0   0
  2   1   1|   2   2   2|   0   1   1
  0   0   2|   4   3   3|   4   3   1
Process 2 completed
Available matrix: 5 3 2
Allocated matrix        Max      need
  0   1   0|   7   5   3|   7   4   3
  0   0   0|   0   0   0|   0   0   0
  3   0   2|   9   0   2|   6   0   0
  2   1   1|   2   2   2|   0   1   1
  0   0   2|   4   3   3|   4   3   1
Process 4 completed
Available matrix: 7 4 3
Allocated matrix        Max      need
  0   1   0|   7   5   3|   7   4   3
  0   0   0|   0   0   0|   0   0   0
  3   0   2|   9   0   2|   6   0   0
  0   0   0|   0   0   0|   0   0   0
  0   0   2|   4   3   3|   4   3   1
Process 1 completed
```

```
Available matrix: 7 5 3
Allocated matrix        Max      need
  0   0   0|   0   0   0|   0   0   0
  0   0   0|   0   0   0|   0   0   0
  3   0   2|   9   0   2|   6   0   0
  0   0   0|   0   0   0|   0   0   0
  0   0   2|   4   3   3|   4   3   1
Process 3 completed
Available matrix: 10 5 5
Allocated matrix        Max      need
  0   0   0|   0   0   0|   0   0   0
  0   0   0|   0   0   0|   0   0   0
  0   0   0|   0   0   0|   0   0   0
  0   0   0|   0   0   0|   0   0   0
  0   0   2|   4   3   3|   4   3   1

Process 5 completed

Available matrix: 10 5 7

The system is in a safe state!!
```

Experiment -10

Simulate the following page replacement algorithms.
a) FIFO b) LRU c) LFU

a) AIM: To write a 'C' Program to simulate First in First out (FIFO) Page Replacement Algorithm

```
#include<stdio.h>
void main( )
{
int a [20],c[10],i,k,ps,nop,npf=0,nps=0;
printf ("\n\tEnter no of pages :" ) ;
scanf ("%d",& nop) ;
printf ("\n\tEnter pages :" ) ;
for (i=0;i<nop;i++)
{
scanf("%d",&a[i]);
}
printf("\n \t Enter no of  page frames :") ;
scanf("%d",&ps);
for(i=0;i<nop;i++)
{
for(k=0;c[k]!=a[i]&&k<nps;k++);
        if(k==nps)
        {       npf++;
                if(nps==ps)
                for(k=0;k<nps-1;k++)
                c[k]=c[k+1];
                c[k]=a[i];
                if(nps<ps)
                nps++;
         }
printf("\n");
for(k=0;k<nps;k++)
printf("%d",c[k]);
}
printf("\n \tNo of page faults occurred are %d",npf) ;
 }
```

OUTPUT:

```
Enter no of pages: 20
Enter pages: 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
Enter no of page frames: 3
7
70
701
012
012
123
230
304
042
423
230
230
230
301
012
012
012
127
270
701
No of page faults occurred are 15
```

b) AIM: To write a ‘C’ Program to simulate Least Recently Used (LRU) Page Replacement Algorithm

```
#include<stdio.h>
void main ( )
{
int a[50],c[10],i,j,k,ps,nop,npf=0,nps=0;
printf("\n\t Enter no of pages :");
scanf("%d",&nop);
printf("\n\tEnter pages :");
for(i=0;i<nop;i++)
{
scanf("%d",&a[i]);
}
printf("\n\tEnter no of  page frames : ");
scanf ("%d",&ps);
for(i=0;i<nop;i++)
{
for(k=0;c[k]!=a[i]&&k<nps;k++);
        if(k==nps)
        {
                npf++;
                if(nps==ps)
                for(k=0;k<nps-1;k++)
                c[k]=c[k+1];
                c[k]=a[i];
                if(nps<ps)
                nps++;
        }
else
        {
```

```
   for (j=k;j<nps-1;j++)
                c[j]=c[j+1];
                c[j]=a[i];
}
printf("\n") ;
        for(k=0;k<nps;k++)
        printf("%d" ,c[k]) ;
}
printf (" \n\tNo of page faults occurred are %d",npf);
}
```

OUTPUT:

```
 Enter no of pages: 20

 Enter pages: 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

 Enter no of page frames: 3

7

70

701

012

120

203

230

304

042

423

230

203

032
```

```
321

312

120

201

017

170

701

No of page faults occurred are 12
```

C) AIM: To write a 'C' Program to simulate Least Frequently Used (LFU) Page Replacement Algorithm

PROGRAM:

```
#include<stdio.h>

#include<conio.h>

int fr[3];

void main()

{

void display();

int p[12]={2,3,2,1,5,2,4,5,3,2,5,2},i,j,fs[3];

int index,k,l,flag1=0,flag2=0,pf=0,frsize=3;

clrscr();

for(i=0;i<3;i++)

{

fr[i]=-1;

}

for(j=0;j<12;j++)

{

flag1=0,flag2=0;

for(i=0;i<3;i++)

{

if(fr[i]==p[j])

{

flag1=1;

flag2=1;

break;
```

```
}
}
if(flag1==0)
{
for(i=0;i<3;i++)
{
if(fr[i]==-1)
{
fr[i]=p[j];
flag2=1;
break;
}
}
}
if(flag2==0)
{
for(i=0;i<3;i++)
fs[i]=0;
for(k=j-1,l=1;l<=frsize-1;l++,k--)
{
for(i=0;i<3;i++)
{
if(fr[i]==p[k])
fs[i]=1;
}
```

```
}
for(i=0;i<3;i++)
{
if(fs[i]==0)
index=i;
}
fr[index]=p[j];
pf++;
}
display();
}
printf("\n no of page faults :%d",pf);
getch();
}
void display()
{
int i;
printf("\n");
for(i=0;i<3;i++)
printf("\t%d",fr[i]);
}
```

OUTPUT :

```
2 -1 -1

2  3 -1
2  3 -1
2  3  1
```

```
2  5  1
2  5  1
2  5  4
2  5  4
3  5  4
3  5  2
3  5  2
3  5  2
no of page faults : 4
```

Experiment-11

Simulate the following File allocation strategies
a) Sequenced b) Indexed c) Linked

a) AIM: To write a 'C' Program to simulate Sequenced File Allocation strategy.

```
#include<stdio.h>
void main()
{
int n,i,j,b[20],sb[20],t[20],x,c[20][20];
printf("Enter no.of files:");
scanf("%d",&n);
for(i=0;i<n;i++)
{
        printf("Enter no. of blocks occupied by file%d:",i+1);
        scanf("%d",&b[i]);
        printf("Enter the starting block of file%d:",i+1);
        scanf("%d",&sb[i]);
        t[i]=sb[i];
        for(j=0;j<b[i];j++)
                c[i][j]=sb[i]++;
}
printf("Filename\tStart block\tlength\n");
for(i=0;i<n;i++)
printf("%d\t    %d   \t%d  \n",i+1,t[i],b[i]);
printf("Enter file name:");
scanf("%d",&x);
printf("\nFile name is:%d",x);
printf("\nlength is:%d",b[x-1]);
printf("\nblocks occupied:");
for(i=0;i<b[x-1];i++)
printf("%4d\t",c[x-1][i]);
}
```

OUTPUT:

```
Enter no. of files: 2
Enter no. of blocks occupied by file1:4
Enter the starting block of file1:2
Enter no. of blocks occupied by file2:10
Enter the starting block of file2:5
Filename        Start block     length
1           2           4
2           5           10
Enter file name:1
File name is:1
length is:4
blocks occupied:   2                   3                   4                   5

[satyakumari@localhost os]$ cc fileseqorg.c
[satyakumari@localhost os]$ ./a.out
Enter no.of files:2
Enter no. of blocks occupied by file1:4
Enter the starting block of file1:2
Enter no. of blocks occupied by file2:10
Enter the starting block of file2:5
Filename        Start block     length
1           2           4
2           5           10
Enter file name:2
File name is:2
length is:10
blocks occupied:   5      6         7         8         9         10        11        12        13        14
```

b) AIM: To write a ‘C’ Program to simulate Indexed File Allocation strategy.

```
#include<stdio.h>
main()
{
 int n,m[20],i,j,sb[20],s[20],b[20][20],x;
 printf("\nEnter no. of files:");
 scanf("%d",&n);
 for(i=0;i<n;i++)
 {
        printf("\nEnter starting block and size of file%d:",i+1);
        scanf("%d%d",&sb[i],&s[i]);
        printf("\nEnter blocks occupied by file%d:",i+1);
        scanf("%d",&m[i]);
        printf("\nenter blocks of file%d:",i+1);
        for(j=0;j<m[i];j++)
        scanf("%d",&b[i][j]);
 }
printf("\nFile\t index\tlength\n");
 for(i=0;i<n;i++)
 {
printf("%d\t%d\t%d\n",i+1,sb[i],m[i]);
 }
printf("\nEnter file name:");
 scanf("%d",&x);
 printf("\nfile name is:%d\n",x);
 i=x-1;
 printf("Index is:%d",sb[i]);
 printf("\nBlock occupied are:");
 for(j=0;j<m[i];j++)
printf("%3d\n",b[i][j]);
}
```

OUTPUT:

```
Enter no. of files:2
Enter starting block and size of file1:2 5
Enter blocks occupied by file1:5
enter blocks of file1:2 5 4 6 7
Enter starting block and size of file2:3 4
Enter blocks occupied by file2:5
enter blocks of file2:2 3 4 5 6

File    index  length
1       2      5
2       3      5

Enter file name:1
file name is:1
Index is:2
Block occupied are:  2
  5
  4
  6
  7
```

c) AIM: To write a ‘C’ Program to simulate Linked File Allocation strategy.

```
#include<stdio.h>
struct file
{
 char fname[10];
 int start,size,block[10];
}f[10];
main()
{
 int i,j,n;
 printf("\nEnter no. of files:");
 scanf("%d",&n);
 for(i=0;i<n;i++)
 {
 printf("\nEnter file name:");
 scanf("%s",&f[i].fname);
 printf("\nEnter starting block:");
 scanf("%d",&f[i].start);
 f[i].block[0]=f[i].start;
 printf("\nEnter no.of blocks:");
 scanf("%d",&f[i].size);
 printf("\nEnter block numbers:");
 for(j=1;j<=f[i].size;j++)
 {
        scanf("%d",&f[i].block[j]);
 }
 }
 printf("File\tstart\tsize\tblock\n");
 for(i=0;i<n;i++)
 {
        printf("%s\t%d\t%d\t",f[i].fname,f[i].start,f[i].size);
        for(j=1;j<=f[i].size-1;j++)
                printf("%d--->",f[i].block[j]);
        printf("%d",f[i].block[j]);
        printf("\n");
  }
}
```

OUTPUT:

```
Enter no. of files:2
Enter file name:venkat
Enter starting block:20
Enter no.of blocks:6
Enter block numbers:4   12   15   45   32  25
Enter file name:rajesh
Enter starting block:12
Enter no.of blocks:5
Enter block numbers:6 5 4 3 2
File      start    size    block
venkat     20       6      4--->12--->15--->45--->32--->25
rajesh     12       5      6--->5--->4--->3--->2
```

Experiment-12

Write a C program that illustrates two processes communicating using shared memory

AIM: Write a C program that illustrates two processes communicating using shared memory

sender.c

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/ipc.h>
#include<sys/shm.h>
#include<sys/types.h>
#include<string.h>
#include<unistd.h>
int main()
{
   int i;
    void *shared_memory;
    char buff[100];
    int shmid;
    shmid=shmget((key_t)2345, 1024, 0666|IPC_CREAT);
    printf("Key of shared memory is %d\n",shmid);
    shared_memory=shmat(shmid,NULL,0);
    printf("Process attached at %p\n",shared_memory);
    printf("Enter some data to write to shared memory\n");
    read(0,buff,100); //get some input from user

   strcpy(shared_memory,buff);
   printf("You wrote : %s\n",(char *)shared_memory);
   }
```

receiver.c

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/ipc.h>
#include<sys/shm.h>
#include<sys/types.h>
int main()
{
   int i;
   void *shared_memory;
   char buff[100];
   int shmid;
   shmid=shmget((key_t)2345, 1024, 0666);
   printf("Key of shared memory is %d\n",shmid);
   shared_memory=shmat(shmid,NULL,0);
   printf("Process attached at %p\n",shared_memory);
   printf("Data read from shared memory is : %s\n",(char *)shared_memory);
   }
```

Output:

```
cc sender.c
./a.out
Key of shared memory is 65538
Process attached at 0xb784f000
Enter some data to write to shared memory
abcd
You wrote : abcd
cc receiver.c
./a.out
Key of shared memory is 65538
Process attached at 0xb7772000
Data read from shared memory is : abcd
```

Experiment-13

Write a C program to simulate producer and consumer problem using semaphores

AIM: Write a C program to simulate producer and consumer problem using semaphores

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#define BUFFER_SIZE 5
#define MAX_ITEMS 5
int buffer[BUFFER_SIZE];
int in = 0;
int out = 0;
int produced_count = 0;
int consumed_count = 0;
pthread_mutex_t mutex;
pthread_cond_t full;
pthread_cond_t empty;
void* producer(void* arg) {
  int item = 1;
  while (produced_count < MAX_ITEMS) {
    pthread_mutex_lock(&mutex);

    while (((in + 1) % BUFFER_SIZE) == out) {
      pthread_cond_wait(&empty, &mutex);
    }
    buffer[in] = item;
    printf("Produced: %d", item);
    item++;
    in = (in + 1) % BUFFER_SIZE;
```

```
        produced_count++;
        pthread_cond_signal(&full);
        pthread_mutex_unlock(&mutex);
    }
    pthread_exit(NULL);
}
void* consumer(void* arg) {
    while (consumed_count < MAX_ITEMS) {
        pthread_mutex_lock(&mutex);
        while (in == out) {
            pthread_cond_wait(&full, &mutex);
        }
        int item = buffer[out];
        printf("Consumed: %d", item);
        out = (out + 1) % BUFFER_SIZE;
        consumed_count++;
        pthread_cond_signal(&empty);
        pthread_mutex_unlock(&mutex);
    }
    pthread_exit(NULL);
}
int main() {
    pthread_t producerThread, consumerThread;
    pthread_mutex_init(&mutex, NULL);
    pthread_cond_init(&full, NULL);
    pthread_cond_init(&empty, NULL);
    pthread_create(&producerThread, NULL, producer, NULL);
    pthread_create(&consumerThread, NULL, consumer, NULL);
    pthread_join(producerThread, NULL);
    pthread_join(consumerThread, NULL);
```

```
  pthread_mutex_destroy(&mutex);
  pthread_cond_destroy(&full);
  pthread_cond_destroy(&empty);
  return 0;
}
```

OUTPUT:

```
Produced: 1
Produced: 2
Produced: 3
Produced: 4
Consumed: 1
Consumed: 2
Consumed: 3
Consumed: 4
Produced: 5
Consumed: 5
```

Experiment-14

Write C program to create a thread using pthreads library and let it run its function.

AIM: Write C program to create a thread using pthreads library and let it run its function.

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

void *mythread(void *vargp)
{
  sleep(1);
  printf("welcome to thread creation example·\n");
  return NULL;
}
int main()
{
  pthread_t tid;
  printf("before thread\n");
  pthread_create(&tid,NULL,mythread,NULL);
  pthread_join(tid,NULL);
  exit(0);
}
```

OUTPUT:

```
before thread
welcome to thread creation example·
```

Experiment-15

Write a C program to illustrate concurrent execution of threads using pthreads library.

AIM: Write a C program to illustrate concurrent execution of threads using pthreads library.

```
#include<stdio.h>

#include<stdlib.h>

#include<pthread.h>

void *mythread1(void *vargp)

{

   int i;

   printf("thread1\n");

      for(i=1;i<=10;i++)

     printf("i=%d\n",i);

   printf("exit from thread1\n");

  return NULL;

}

void *mythread2(void *vargp)

{

    int j;

   printf("thread2 \n");

   for(j=1;j<=10;j++)

    printf("j=%d\n",j);

 printf("Exit from thread2\n");

  return NULL;
}
int main()
{
  pthread_t tid;
  printf("before thread\n");
  pthread_create(&tid,NULL,mythread1,NULL);
  pthread_create(&tid,NULL,mythread2,NULL);
  pthread_join(tid,NULL);
  pthread_join(tid,NULL);
  exit(0);
}
```

Output:

```
$cc record7.c– l pthread
$ ./a.out
thread1
i=1
i=2;
i=3
thread2
j=1
j=2
j=3
j=4
j=5
j=6
```

```
j=7

j=8

i=4

i=5

i=6

i=7

i=8

i=9

i=10

exit from thread1

j=9

j=10

exit from thread2
```

Additional Experiments

Experiment-1

AIM: Loading executable programs into memory

Program

```
/* using execvp to execute the contents of argv */
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
int main(int argc, char *argv[])
 {
if(argc!=2)
printf("exec failure, usage: <programfile> <command>\n");
 else
execvp(argv[1], &argv[1]);
return 0;
}
```

Output:

```
$ ./a.out clear
$./a.out ls
a.out loading.c loading.c~ loading.o
```

Experiment-2

AIM: System Call implementation- read (), write (), open () and close ()

Program :

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <errno.h>
#include <sys/types.h>
#include <unistd.h>
#define BUF_SIZE 8192
int main(int argc, char* argv[])
{
int input_fd, output_fd; /* Input and output file descriptors */
ssize_t ret_in, ret_out; /* Number of bytes returned by read() and write() */
char buffer[BUF_SIZE]; /* Character buffer */ /* Are src and dest file name arguments missing
*/
 if(argc != 3)
{
 printf ("Usage: <programfile> <sourcefile> <destfile>\n");
return 1;
} /* Create input file descriptor */
input_fd = open (argv [1], O_RDONLY);
if (input_fd == -1)
{
perror ("open");
return 2;
}
output_fd = open(argv[2], O_WRONLY | O_CREAT, 0644);
if(output_fd == -1)
{
perror("open");
return 3;
} /* Copy process */
while((ret_in = read (input_fd, &buffer, BUF_SIZE)) > 0)
{
ret_out = write (output_fd, &buffer, (ssize_t) ret_in);
if(ret_out != ret_in)
{
/* Write error */ perror("write");
return 4;
} } /* Close file descriptors */
close (input_fd);
close (output_fd);
printf("File Copied successfully\n");
return (EXIT_SUCCESS);
}
```

Output:

```
$cc execute.c
$cat o.txt
My name is uma
$./a.out o.text n.text
File copy is successful
$cat n.txt
My name is uma
if permission denied
$sudo chmod 777 n.txt
```