# ANNA UNIVERSITY

# JAYA ENGINEERING COLLEGE

**(AN ISO 9001:2000 Certified Institution)**
**THIRUNINRAVUR – 602 024**
**Email: info@jec.ac.in**
**Website: www.jec.ac.in**


## DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

## CS8461 –OPERATING SYSTEM LABORATORY

IV SEMESTER – C.S.E

**Practical Record**

## LIST OF EXPERIMENTS

**EX.NO:1** **BASIC UNIX COMMANDS**

**AIM:**

To study and to execute the basic shell commands in UNIX.

## GENERAL PURPOSE COMMANDS

### 1. The date command

The date command is used to display the current date with day of week, month, day, time ( 24 Hours clock) and the year.

**Syntax:** $ date

Thu Jan 22 09:41:40 IST 2009

The date command can also be used with following format specifications.

The format specification must be preceded by a (+) symbol followed by (%) operator and a single character.

| Format | Purpose | Example | Result |
|---|---|---|---|
| +%m | To display only month | $date +%m | 01 |
| +%h | To display month name | $date +%h | jan |
| +%d | To display day of month | $date +%d | 22 |
| +%y | To display last two digits of the year | $date +%y | 09 |
| +%H | To display only Hours | $date +%H | 09 |
| +%M | To display only Minutes | $date +%M | 53 |
| +%S | display only seconds | $date +%S | 52 |

### 2. The echo and banner commands

The echo command is used to print the message on the screen, whatever we type on the line.

**Syntax : $ echo text**

**E.g. :** $ echo Basic Unix commands

Basic Unix commands

The banner command prints our message in large letters to give the impression of a banner.

**Eg: $ banner UNIX**

### 3. The Unix Calendar : cal

The cal command helps us to keep track of our days. I.e., It displays the specified month or year calendar.

$ cal 2009 → Prints the calendar for the entire year

$ cal 1 2009 → Prints the calendar for the month of January

**4. Unix calculator : bc**

Unix offers an online calculator and can be invoked by the command bc.

This calculator is programmable and has complex functions.

**Syntax:** $ bc

To start up the desk calculator, type the following:

$ bc

It doesn't display command prompt and simply waits for you. Type the necessary calculations.

**Eg:** 4+5+6+3+2

20

bc continues this process until you enter CTRL + D to terminate.

**5. The who command**

The who command is used to display data about all the users, who are currently logged into the system.

**Syntax :** $ who

**Eg :** indira@rmk0459:~$ who

indira tty7 2009-01-22 08:46 (:0)

indira pts/0 2009-01-22 09:41 (:0.0)

It displays the output format as login name of the user, terminal line, login date and time.

**6. The Who am i command**

This command displays a single line of output pertaining to the login details of the user.

**E.g. :** who am i

indira pts/0 2009-01-22 09:41 (:0.0)

This command identifies the user and lists the user name, terminal line, the date and time of login.

Several command line options of who command:

| Option | Description |
|---|---|
| -a | Displays everything about all users. |
| -b | Displays the date & time that the system was last rebooted. |
| -d | Displays all dead processes. |
| -H | Displays verbose column headings. |
| -m | Displays only your own statistics. |
| -q | Gets only the number of users and their login names. |

**7. The finger command**

The finger command gathers and displays the information about the users, which includes login name, Home directory etc.,

**Syntax:** $ finger indira

Login: indira Name:

Directory: /nhome/staff/indira Shell: /bin/bash

On since Thu Jan 22 08:46 (IST) on tty7 from :0

On since Thu Jan 22 09:41 (IST) on pts/0 from :0.0

No mail.

No Plan.

To display the information about all users who are currently logged on.

**Eg :** $ finger

| Login | Name | Tty | Idle | Login Time | Office | Office Phone |
|---|---|---|---|---|---|---|
| indira | | tty7 | | Jan 22 08:46 (:0) | | |
| indira | | pts/0 | | Jan 22 09:41 (:0.0) | | |

**8. The id command**

The id command is used to display the num,erical value that corresponds to our login name I.e., every valid UNIX user is assigned a login name, a user id and a group-id

**Syntax :** $ id

It displays login name, user-id and and group-id.

**E.g. :** $ id

uid=8789(indira) gid=506(staff) groups=506(staff)

**9. The tty command**

The tty (teletype) command is used to know the terminal name that we are using.

**Syntax :** $ tty

**E.g. :** $ tty

/dev/pts/0

**COMMAND GROUPING**

This means to execute number of commands in a single line. This can be accomplished by using the following command grouping options.

**1. The semicolon (;)**

UNIX has a limitation to execute only one command at a time, the semicolon operator (;) overcomes this limitation and can be used to separate multiple commands at the command line.

Syntax : $ command1; command2;.....; command n

**Eg :** ~$ who; date

indira tty7 2009-01-22 08:46 (:0)

indira pts/0 2009-01-22 09:41 (:0.0)

Thu Jan 22 11:51:48 IST 2009

It executes the both commands at a time.

**2. The && operator**

The '&&' operator signifies the logical AND operation appears in between two or more valid Unix commands. It means, that only if the first command is successfully executed, then the next command will be executed.

**Syntax :** $ command1 && command2 &&.....&& command n

**E.g. :** $ who && date

indira tty7 2009-01-22 08:46 (:0)

indira pts/0 2009-01-22 09:41 (:0.0)

Thu Jan 22 11:57:07 IST 2009

It executes first the who command and after successful execution of who command the date command will be executed.

**3. The '||' operator**

Signifies the logical OR operation appears in between two or more valid Unix commands. It means, that only if the first command happens to be unsuccessful , it will continue to execute next command.

**Syntax :** $ command1 || command2 ||..... || command n

**E.g. :** $ ls || date

The above first command is used to list the files and no files are there, it will continue with next command execution.

## COMMANDS FOR WORKING WITH DIRECTORY

**1. pwd** (Print Working Directory)

Shows current working directory path.

**Syntax:** $pwd <Enter>

**2. mkdir** (Make Directory)

Makes a sub-directory named "dirname" in the current directory.

**Syntax:** $ mkdir dirname

**Eg:** $ mkdir exec

**3. cd** (Change Directory)

Change current directory. Without a "dirname", it will return you to your home directory. Otherwise, it takes you to the directory named. "cd /" will take you to the root directory.

**Syntax :** $ cd [dirname]

**Eg:** $cd /user

$cd ..

**4. rmdir** (Remove Directory)

Removes the directory "dirname"

**Syntax :** $ rmdir dirname

**5. ls** {directory}

Shows directory listing. If no "directory" is specified, "ls" prints the names of the files in the current directory.

1. **Syntax:** $ ls [option]... [File]...

**Options**

-a list all files including files that start with “.”

-s list size of files(in kilobytes).

-l long list, shows ownership, permissions and links.

-l –g lists the group of each file or directories when used with –l.

-t lists files chronologically.

-u list files using time of last access instead of times of last modification.

$ ls > output filename Command is used to send the output to the file.

**FILE SYSTEM MANIPULATION**

**1. cat** {filename}

**a.** Prints out ( to the screen ) the contents of the named file.

**Syntax**: $cat > filename

**E.g.** $ cat > hello

Type something

^d

**b.** It is also used to view the contents in the file.

**Syntax**: $cat filename

**E.g.:** $ cat hello

**c.** It is also used to concatenate files into a single file.

**Syntax**: $cat file1 file2 >file3

**2. cp** {filename(s)}{path}

Copies files from one directory/filename to another.

**Syntax:**

$ cp *source-filename destination-filename* To copy a file into another file.

$ cp *source-filename destination-directory* To copy a file into another directory.

**E.g.:**

$ cp f1 f2 makes a file "f2" identical to "f1".

$ cp *.c src/ copies all files that end in ".c" into the "src" subdirectory.

**3. mv** filename path

Moves "filename" to "path". This might consist of a simple renaming of the file.

**Syntax:**

$ **mv** *present-filename new-filename* To rename a file

$ mv source-filename destination-directory To move a file into another directory.

**E.g.:** $ mv file1 file2 renaming.

$ mv file1 /tmp/ or mv file1 /tmp/file2 moving the file to a new directory

**4. ln** -s {source} {dest}

Creates a symbolic link from {source} to {dest}. {Source} can be a directory or a file. Allows to move around with ease instead of using long and complicated path names.

**Syntax:**

$ ln source-filename destination-filename

To create another name for a same file called a link or alias name.

**Eg:**

$ ln f1 f2 creates another name for f1

**5. rm** filename(s)

Removes files. Careful with this one - it is irreversible. It is usually aliased ( in a user"s .cshrc

file ) to "rm -i" which insures that "rm" asks you if you are sure that you want to remove the named file.

**Syntax:** $ r**m** filename to remove a file

**E.g.:** $ rm f1

**6. cmp** {file1} {file2}

Compares the contents of two files from each other. Reports the first different character found, and the line number.

**Syntax:** $ cmp file1 file2

**E.g.**

To compare doc1 and doc2, having the following data.

| Doc1 | doc2 |
|---|---|
| This a document | This is document |
| For internal calculation | For internal calculation. |

$ cmp doc1 doc2

output will be

doc1 and doc 2 differ: byte 6, line1.

**7. diff** {options}{file1} {file2}

Displays *all* the differences between two files or directories to the screen.

**Options**

-b ignores trailing blanks and other strings

-h used for files of unlimited length.

**Syntax:** $ diff file1 file2

**8. comm** {options} file1 file2

Displays common lines in the two files

**Options**

-1 suppresses the display of the first column in the output.

-2 suppresses the display of the second column in the output.

-3 suppresses the display of the third column in the output.

**Syntax:** $ comm. {-{1}{2}{3}} file1 file2

**E.g.:**

1. $ comm. emp1 emp2

| emp1 | emp2 |
|---|---|
| Navneet | Ritu |
| Ritu | Rachana |

Will produce the following output

Ritu

Rachana

Navneet

2. $ comm. –12 emp1 emp2

the output will be

Ritu

**9. uniq** {option}{file}

Displays the unique lines.

**Options**

-u display only the unique lines

-d display only the duplicate lines

-c display all lines, each processed by a count of the records.

**Syntax:** $ uniq {option}{file name}

**Eg**

$ uniq emp.dat

-display only the unique lines of the file.

**10. chmod** {options}

Changes the permission modes of a file.

2. Syntax

**$ chmod [***who op permission***]** *filename*

*who* can be any combination of: **u** (user)

**g** (group)

**o** (other)

**a** (all) (i.e. **ugo** )

*op* adds or takes away permission, and can be:+ (add permission),- (remove permission), = (set to exactly this permission) *permission* can be any combination of **r** (read), **w** (write), **x** (execute)

**Eg: $ chmod a+x**

*filename* (makes *filename* executable by ~ home directory tilde)

If you type "ls -l" in a directory, you might get something like this:

drwx ------3 ertle 512 Jul 16 13:38 LaTeX/

drwxr-xr-- 2 ertle 512 Jun22 12:26 X/

drwxr-xr-x 3 ertle 512 Jul 13 16:29 Xroff/

-rw-r--r-- 1 ertle 373 Oct 3 1992 o.me

-rw-r--r-- 1 ertle 747 Nov 21 1992 profile

-rwxr-xr-x 1 ertle 244 Jul 16 23:44 zap*

The first part of the line tells you the file"s permissions.

For example, the "X" file permissions start with a "d" which tells that it is a directory. The next three characters, "rwx" show that the owner has read, write, and execute permissions on this file. The next three characters, "r-x" shows that people in the same group have read and execute permission on the file. Finally, the last three characters "r-" show that everyone else only has read permission on that

file ( To be able to enter a directory, you need read AND execute permission ). Users can use "chmod" to change these permissions. If the user didn"t want anybody else to be able to enter the "X" directory, they would change the permissions to look like those of the LaTeX directory, like this : "chmod og-rx X" - this means remove the read ("r" ) and execute ("x") permissions from the group ("g") and others ("o").

**11. chown** {options}

Reassigns the ownership of a file from one user to another.

**Syntax:** $chown ownername filename

**12. chgrp** {options}

Performs the same function for the group that owns the file.

**Syntax:** $ chgrp groupname filename

**13. Metacharacters**: are special characters.

* - specifies number of characters

? – specifies a single character

| **option** | **purpose** |
|---|---|
| [ ] – range | used to match a whole set of filenames at a command line |

**INPUT/ OUTPUT REDIRECTION**

**1. Input Redirection**

Changing the default input source

The input redirection operator: “<”

**Syntax:**$ command < filename

Standard input is reassigned to the default devices.

**Eg:** $ cat < emp.dat

cat will read from the standard input which is redirected to the emp.dat file by the shell

**2. Output Redirection**

Changing the default destination of output.

The output redirection operator: ">"

**Syntax:** $ command > filename

Standard output is reassigned to the default devices.

**Eg:** $ cat emp.dat > emp.out

will read from the file emp.dat and stores the output into the emp.out file.

**3. Standard Error Redirection**

Standard error is used to display error messages.

By default standard error is assigned to the terminal

File descriptor for standard files:

0 is assigned to the standard input.

1 is assigned to the standard output.

2 is assigned to the standard error.

**Syntax:** $ command 2>err_file

**4. Pipes**

It is ued to connect two commands together .A pipe is a mechanism by which the output of one command can be channeled into the input of another command.A pipe is effected by the character (|) and is placed between the two ommands.

**Example**:

$who | wc –l

$ ls | sort |wc –l

**5. tee**

It is used to save the output that is produced in the middle of the pipe.

**Syntax :** $ command | tee file

**FILTERS**

A filter is device file that reads from the std input and writes to the std output in particular format. Filters are the control tools of UNIX.

**1.head**

Default displays the first ten lines of a file.

**Syntax :** $ head {-n} filename

-n displays first n lines of a file.

**2. tail**

Default displays 10 lines of a file from the end of a file.

**Syntax :** $ tail {+/-n} filename

-n display last n lines of a file.

+n displays lines from the nth line till end of the file.

**3. pg**(page)

Displays the contents page by page.The user has to strike the "enter" key for scrolling.

**Syntax:** $ pg filename1 filename2 ….

**Options**

-e not to pause at the end of each file.

-s prints all messages and prompts in standard output mode.

**E.g.** $ ls –l | pg

**4. more**

It is similar to page command.the difference is :the user has ti strike "space bar" key instead of "enter" key.It can also works with multiple files.

**Syntax :** $ more filename

| Command | Purpose |
|---|---|
| Spacebar | Scrolls on screenful forward |
| F | Scrolls on screenful forward |
| B | Scrolls on screenful backward |
| J | Scrolls on line forward |
| K | Scrolls on line backward |
| 100G | Goes to line number 100 |
| G | Goes to last line of the file |

**5. grep**

It is used to search and print specified patterns from a file.It is abbreviation of "global regular expression and print".

**Syntax :** $ grep [option] pattern file(s)

**Options**

-n prints line numbers

-v the reverse search criterion

-c display only a count of matching pattern

**6.sort**

It is used to sort the contents of a file.

**Syntax :** $ sort filename

**Options available in sort:**

| Command | Purpose |
|---|---|
| -r | Sorts and displays the contents in reverse order |
| -c | Checks if the file is sorted |
| -n | Sort numerically |
| -u | Removes duplicate records |
| -m list | Merges sorted files in list |

**7. nl**(no. of lines)

It adds file number to a file and it displays the file and not provides to access to edit.

**Syntax :** $ nl filename

**8. cut**

It is used to select the specified fields from a line of text.

**Syntax :** $ cut {options} filename

**Options**

c selects columns specified by list

f selects fields specified by list

d field delimiter (default is tab)

**9. paste**

Merge lines of files.

**Syntax :** $ paste {options} filename1 filename2

**Options**

- d delimiters

-s serial (paste one file at a time instead of in parallel)

**10. wc (word count)**

Counts the number of words, characters and lines present in a file. The output consists of 3 numbers which are no. of lines, no. of words and the no. of characters in that order.

**Syntax**

$ wc {options} filename

**Options**

-c character count only.

-w word count only

-l for line count only.

**11. tr(translate)**

This command is used to squeeze the repetitive characters from the input and translate the input to some other form.

**Syntax:**$ tr str newstr <filename

where newstr is the string to replaced with every occurrence of str in the input.\

**TRANSFER DATA BETWEEN DEVICES / USER**

**1. mesg:**

It is used send message to another user's terminal.

**Syntax :**$ mesg y

Where mesg is the command used to give permission

y specifies yes,to communicate.

N specifies no to communicate

**2. write loginname**

Send a message to another user. Each line will be sent to the other person as you hit the carriage-

return. Press <CTRL>-D to end the message. Write won"t work if the other user has typed "mesg n".

**Syntax :** $ write usename

Where write is the command used to communicate

Username is the name if the user to whom u want to communicate

3. **wall:**

It is used to send message to all users those who currently logged in using the UNIX server.

**Syntax :** $ wall message

4.**news:**

It is used to read messages published on the system administrator.

**Syntax :** $ news

5. **mail** {login-name}

Read or send mail messages. If no "login-name" is specified, "mail" checks to see if you have any mail in your mail box. With a "login-name", "mail" will let you type in a message to send to that person. For more advanced mail processing, you might try "elm" or "pine" at the command line, or "M-x mail" in emacs.

1. **Syntax:** $ mail {options} login-name

**Options**

q returns the undeleted messages to the mail file and exit mail program

p displays the previous message again

s saves the message in a file

**Eg:**

$ mail username

Body of the mail

(ctrl +d)

**RESULT:**

Thus the basic shell commands have been studied and executed in UNIX.

**EX.NO.1.1 UNIX EDITOR**

**AIM:**

To study about the VI editor and perform all editing options

**INTRODUCTION**

An Editor is a program that allows us to see a portion of a file on the screen and to modify characters and lines by simply typing at the cursor position. There are a number of editors that may be included with UNIX system, including ed, ex, vi, EMACS. The latter two use the entire screen, which is a big advantage, and both are powerful editors. We are focusing on vi because it is easier and perhaps more importantly it's guaranteed to always be part of UNIX.

**THE VI EDITOR**

vi stands for visual. Vi is a full screen editor that allows the user to edit the entire document at the same time. vi has no menus but instead uses combinations of keystrokes in order to accomplish commands. The vi editor was written in the university of California at Berkeley by Bill joy, who is one of the co-founder of Sun-Micro systems.

**STARTING WITH VI**

There are different ways to start vi editor

$vi : opens an empty editor

$vi filename : opens an editor with specified file name

eg.,$ vi myfile

**VI MODES**

vi has two modes: the command mode and the insert mode.

**Command mode:**

In this mode, all the keys pressed by the user are interpreted to be editor commands. No text is displayed on the screen,even if the corresponding key is pressed on keyboard.

**Insert mode:**

Insert mode permits to insert new text, editing and replacement of existing text. In the insert mode letters typed in at the keyboard are echoed on the screen.

When the editor is opened it is in command mode. If you want to switch to 'insert mode' press **i** to enter the insert mode of vi editor. If you wish to leave insert mode and return to the command

mode, hit the **ESC** key. If you are not sure which mode you are, hit **ESC** a couple of times and that will put you back in command mode.

**MOVING THE CURSOR**

The cursor movement commands are

| command | action |
|---|---|
| h or Backspace | Left one character |
| l or Space bar | Right one character |
| k or - | Up one line |
| j or + | Down one line |
| w | Moves forward a word |
| #b | Moves back a word |
| #e | Moves to last character in a word |
| f[character] | Moves right to specified character in a line |
| F[character] | Moves left to the specified character in a line |
| t[character] | Moves right and places it one character before the specified characters |
| T[character] | Moves left and places it one character before the specified character |
| 0(zero) | Moves to the beginning of the line |
| $ | Moves to the end of the line |
| L | Moves the cursor to the last line in the screen |
| #G | Moves the cursor to the end of the file/moves to the specified line number |

Where # specifies the number proceeding them

eg: 3w moves three words forward

vi editor must be in command mode to perform all cursor movements.

**EDITING THE FILE**

commands to insert a text in a file

| command | purpose |
|---|---|
| i | Insert text to the left of the cursor |
| I | Insert text at beginning of the line |
| a | Appends text to the right of the cursor |
| A | Appends text at end of line |
| o | Appends a new line below |
| O | Appends a line above |

To do all the above insert commands, first position the cursor appropriately by using cursor movement commands.

Commands to delete a text from a file

| command | purpose |
|---|---|
| x | Deletes one character |
| nx | Deletes n characters, where n is the number of characters |
| #x | Deletes n characters at the cursor position |
| #N | Deletes n characters before the cursor position |
| D | Deletes a line from cursor position to the end of the line |
| d0 | Deletes from the cursor position to the starting of the line |
| #dd | Deletes the current line where the cursor is positioned |
| #dw | Deletes the word from the cursor position to the end of the word |

Before using the above commands the vi editor must be in command mode. To do all the above delete commands, first position the cursor appropriately by using cursor movement commands.

**Commands for undo**

vi has powerful undo features. Suppose some changes that has been made to the file has to be unaffected we use undo options.

| command | Purpose |
|---|---|
| u | To undo the most recent change |
| U | To undo all the changes in the current line |

**SAVING TEXT**

commands to save a file

| command | purpose |
|---|---|
| :w | Save file and remains in edit mode |
| :x | Save file and quits from edit mode |
| :wq | Save file and quits edit mode |
| :w new-filename | Save file under new filename |
| :q! | Quit without changes from edit mode |
| :sh | Escape to the unix shells |

**QUITTING FROM VI**

After making all the changes in the document,save the document and quit using ':wq' in command mode. If we want to quit without saving the changes made,type':q!' in command mode.

**RESULT:**

Thus the VI editor has been studied and all the editing options were performed

**Ex No:2** **PROCESS CREATION USING SYSTEM CALLS**

**AIM**

To write programs using system calls of unix operating system

**A)PROCESS:**

**PROGRAM:**

```
main()
{
if(fork()==0)
{
system("clear");
printf("CHILD:I am child & my ID is :%d \n",getpid());
printf("CHILD: My parent is :%d \n",getppid());
}
}
```

**OUTPUT:**

```
CHILD:I am child & my ID is :2663
CHILD: My parent is :1
```

**b) PARENT-CHILD PROCESS:**

**PROGRAM:**

```
main()
{
if(fork()==0)
```

```
{
system("clear");
printf("CHILD:I am child & my ID is :%d \n",getpid());
printf("CHILD: My parent is :%d \n",getppid());
}
else
{
printf("PARENT:I am parent:%d \n",getpid());
sleep(2);
printf("PARENT: My parent is :%d \n",getppid());
}
}
```

**OUTPUT:**

```
PARENT:I am parent:2682
CHILD:I am child & my ID is :2683
CHILD: My parent is :2682
PARENT: My parent is :2434
```

**c) WAITING PROCESS:**

**PROGRAM:**

```
main()
{
int pid,dip,cpid;
```

```
pid=fork();
if(pid==0)
{
printf("1st Child Process ID is %d\n",getpid());
printf("1st Child Process TERMINATING FROM THE MEMORY\n");
}
else
{
dip=fork();
if(dip==0)
{
printf("2nd Child Process ID is %d\n",getpid());
printf("2nd Child Process TERMINATING FROM THE MEMORY\n");
}
else
{
cpid=wait(0);
printf("Child with pid:%d is dead\n",cpid);
cpid=wait(0);
printf("Child with pid:%d is dead\n",cpid);
}


}
}
```

**OUTPUT:**

1st Child Process ID is 2699

1st Child Process TERMINATING FROM THE MEMORY

2nd Child Process ID is 2700

2nd Child Process TERMINATING FROM THE MEMORY

Child with pid:2699 is dead

Child with pid:2700 is dead

**d) ZOMBIE PROCESS:**

**PROGRAM:**

```
main()
{
int pid=fork();
system("clear");
printf("My ID is %d\n",getpid());
if(pid>0)
{
printf("My parent ID is %d\n",getpid());
sleep(05);

}
}
```

**OUTPUT:**

My ID is 2716

My parent ID is 2716

My ID is 2717

**RESULT**

Thus the programs using system calls of unix operating system was written and executed successfully

**EX NO.3 SIMULATION OF UNIX COMMANDS**

**DEMONSTRATION OF I/O SYSTEM CALLS**

**AIM**

To write c program to simulate unix commands

**PROGRAM 1:**

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

int main()
{
   size_t filedesc = open("test.txt", O_WRONLY | O_APPEND);
   if(filedesc < 0)
      return 1;

   if(write(filedesc,"This will be output to test.txt\n", 32) != 32)
   {
      write(2,"There was an error writing to test.txt\n",39);
      return 1;
   }

   return 0;
}
```

**PROGRAM 2:**

```
#include  <stdio.h>
#include  <fcntl.h>
#include <stdlib.h>
#define BUFFERSIZE 1024
int main(void)
{
      char sbuf[BUFFERSIZE];
```

```
    int fd;
    if((fd=open("test.txt",0660))==-1)
    {
        printf("Cannot open file!\n");
        exit(1);
    }
    read(fd,sbuf,BUFFERSIZE);
    printf("%s\n",sbuf);
    close(fd);
}
```

**OUTPUT:**

```
]$ cc iosys1.c
]$ ./a.out test.txt
]$ cat test.txt
This will be output to test.txt
]$cc iosys2.c
]$./a.out
This will be output to test.txt
```

### b) SIMULATION OF ls COMMAND

**PROGRAM:**

```
#include<stdio.h>
#include<stdlib.h>
#if defined (BSD) && !POSIX_SOURCE
#include<sys/dir.h>
typedef struct dirent Dirent;
#else
```

```
#include<dirent.h>
typedef struct dirent Dirent;
#endif
int main(int argc, char *argv[])
{
DIR *dir;
Dirent *direntry;
dir=opendir(*++argv);
while((direntry=readdir(dir))!=NULL)
printf("%s\n",direntry->d_name);
closedir(dir);
exit(0);
}
```

**OUTPUT**

```
./a.out /home/cse/cse2011/cs045/list
abc.c
even.sh
.io.c.swo
.gnome
.bashrc
```

## c) SIMULATION OF cd COMMAND

**PROGRAM:**

```
#include<sys/types.h>
#include<stdio.h>
#include<stdlib.h>
#define PATH_LENGTH 200
int main(int argc, char *argv[])
{
char olddir[PATH_LENGTH +1];
char newdir[PATH_LENGTH+1];
if(getcwd(olddir,PATH_LENGTH)==-1)
perror("getcwd");
printf("\n present working directory :%s",olddir);
chdir(argv[1]);
printf("\n change directory:%s",argv[1]);
getcwd(newdir,PATH_LENGTH);
printf("\n present working directory :%s",newdir);
exit(0);     }
```

**OUTPUT**

```
./a.out  student1
Present woring directory:/home/cse/cse2011/cs045
Change directory:student1
Present working directory : /home/cse/cse/cse2011/cs045/student1
```

**d) SIMULATION OF mv COMMAND**

**PROGRAM:**

```
#include<stdio.h>
#include<unistd.h>
#include<string.h>
int main(int argc, char *argv[])
{
if(argc!=3||!strcmp(argv[1],argv[2]))
printf("\n working input");
else
if(link(argv[1],argv[2])==0)
return unlink(argv[1]);
return -1;
}
```

**OUTPUT**

./a.out abc.c qq.c

We find that the file abc.c no longer exists. It has been renamed as qq.c

**RESULT**

Thus the programs to simulate unix command has been executed successfully

**EX.NO: 4** **SHELL PROGRAMMING**

**EX.NO: 4.1** **AREA AND CIRCUMFERENCE OF CIRCLE**

**AIM:**

To write a shell program for finding the area and circumference of the circle.

**ALGORITHM:**

**Step 1:** Start the process.

**Step 2:** Read the radius r.

**Step 3:** Calculate area of the circle using the formula, area= Π*r*r

**Step 4:** Calculate circumference of the circle using the formula circumference= 2*Π*r

**Step 5:** Print area and circumference of the circle.

**Step 6:** Stop the process.

**PROGRAM:**

```
#shell program to calculate the area and circumference of the circle
echo enter the radius
read r
area=`expr 22 / 7 \* $r \* $r`
circumference=`expr 2 \* 22 / 7 \* $r`
echo area= $area
echo circumference= $circumference
```

**OUTPUT :**

```
algin@ubuntu:~/shell$ sh circle.sh
enter the radius
3
area= 27
circumference= 18
```

**RESULT :**

Thus the shell program for finding the area and circumference of the circle has been executed and verified.

**EX.NO: 4.2 SWAP TWO NUMBERS**

**AIM:**

To write a shell program to swap two numbers using a temporary variable.

**ALGORITHM:**

**Step 1:** Start the process.

**Step 2:** Read two variables 'a' and 'b'.

**Step 3:** Print the values of the variable before swapping.

**Step 4:** Assign 'a' value to temporary variable 'c', 'b' value to 'a' and 'c' value to 'b'.

**Step 5**: Print the values after swapping.

**Step 6:** Stop the process.

**PROGRAM:**

```
#shell program to swap two numbers using a temporary variable
echo "enter the two numbers for swapping"
read a b
echo Before swapping
echo A=$a and B=$b
c=$a
a=$b
b=$c
echo After swapping
echo A=$a and B=$b
```

**OUTPUT:**

```
algin@ubuntu:~/shell$ sh swap.sh
enter the two numbers for swapping
23 45
Before swapping
A=23 and B=45
After  swapping
A=45 and B=23
```

**RESULT :**

Thus the shell program to swap two numbers using a temporary variable has been executed and verified

**EX.NO : 4.3** **CALCULATE THE GROSS SALARY**

**AIM :**

To write a Shell program to find the gross salary.

**ALGORITHM :**

**Step 1:** Start the process.

**Step 2:** Read name and salary of the employee.

**Step 3:** Calculate da=s*47/100.

**Step 4:** Calculate hra=s*12/100.

**Step 5:** Calculate cca=s*3/100.

**Step 6:** Calculate gross=s+hra+cca+da.

**Step 7:** Print gross salary of the employee.

**PROGRAM:**

```
#shell program to find the gross salary
echo Enter the employee name
read name
echo enter the basic salary
read s
da=`expr $s \* 47 / 100`
hra=`expr $s \* 12 / 100`
cca=`expr $s \* 3 / 100`
gross=`expr $s + $hra + $cca + $da`
echo The gross salary of $name is $gross
```

**OUTPUT :**

```
algin@ubuntu:~/shell$ sh gross.sh
Enter the employee name
Algin
enter the basic salary
12413
The gross salary of Algin is 20108
```

**RESULT:**

Thus the Shell program to find the gross salary has been executed and verified.

**EX.NO : 4.4 GREATEST OF TWO NUMBERS**

**AIM :**

To write a Shell program to find the greatest of two numbers.

**ALGORITHM :**

**Step 1:** Start the process.

**Step 2:** Read two variables 'a' and 'b'.

**Step 3:** Assign 'a' value to big.

**Step 4:** Check if(b>big) then assign 'b' value to big and print “big is the greatest”.

**Step 5:** Stop the process.

**PROGRAM:**

```
#shell script to find the greatest of two numbers
echo Enter the two numbers
read a b
big=$a
if [ $b -gt $big ]
then big=$b
fi
echo $big is the greatest
```

**OUTPUT:**

```
algin@ubuntu:~/shell$ sh grea_2no.sh
Enter the two numbers
21 79
79 is the greatest
```

**RESULT :**

Thus the shell program to find the greatest of two numbers has been executed and verified.

**EX.NO: 4.5 CHECK ADD OR EVEN**

**AIM :**

To write a Shell program to check whether a given number is odd or even.

**ALGORITHM :**

**Step 1:** Start the process.

**Step 2:** Read the input value n.

**Step 3:** Find n %2 and store the result in r.

**Step 4:** Check if r=0 then print " The given no is even" else print "The given no is odd".

**Step 5:** Stop the process.

**PROGRAM :**

```
#shell program to check whether a given number is odd or even
echo Enter a number
read n
r=`expr $n % 2`
if [ $r -eq 0 ]
then echo The given number $n is even
else
echo The given number $n is odd
fi
```

**OUTPUT:**

```
algin@ubuntu:~/shell$ sh odd.sh
Enter a number
4
The given number 4 is even
algin@ubuntu:~/shell$ sh odd.sh
Enter a number
9
The given number 9 is odd
```

**RESULT:**

Thus the shell program to check whether a given number is odd or even has been executed and verified.

**EX.NO : 4.6 CHECK POSITIVE , NEGATIVE OR ZERO**

**AIM :**

To write a shell program to check whether a given number is positive, negative or zero.

**ALGORITHM :**

**Step 1:** Start the process.

**Step 2:** Read the input value n.

**Step 3:** Check if n > 0 then print “The given no is Positive” and go to step 6 else go to 4.

**Step 4:** Check if n< 0 then print “The given no is Negative” and go to step 6 else go to 5.

**Step 5:** Print “The given no is Zero”.

**Step 6:** Stop the process.

**PROGRAM :**

```
#shell script to check whether a given number is positive, negative or zero
echo enter a number
read n
if [ $n -gt 0 ]
then echo The given number $n is a positive number
elif [ $n -lt 0 ]
then echo The given number $n is a negative number
else
echo The given number is zero
 fi
```

**OUTPUT :**

```
algin@ubuntu:~/shell$ sh pos_neg_zero.sh
enter a number 68
The given number 68 is a positive number
algin@ubuntu:~/shell$ sh pos_neg_zero.sh
enter a number -21
The given number -21 is a negative number
```

**RESULT :**

Thus the shell program to check whether a given number is positive, negative or zero has been executed and verified

**EX.NO.4 .7 BASIC ARITHMETIC OPERATIONS**

**AIM :**

To write a shell program to perform the basic arithmetic operations.

**ALGORITHM :**

**Step 1:** Start the process.

**Step 2:** Read the input numbers 'a' and 'b' and the choice ,

1. Add 2. Subtract 3.Multiply 4.Divide 5. Exit.

**Step 3:** If choice = 1 then find sum=a+b and print the sum.

**Step 4:** If choice = 2 then find difference=a-b and print the difference.

**Step 5:** If choice = 3 then find product=a*b and print the product.

**Step 6:** If choice = 4 then find quotient=a/b and print the quotient.

**Step 7:** If choice = 5 then stop the process.

**PROGRAM :**

```
#shell program to perform the basic arithmetic operations
echo Enter the two numbers
read a b
echo "Menu\n 1.Add\n 2.Subtract\n 3.Multiply\n 4.Divide\n 5.Exit\n"
echo Enter your choice:
read choice
case $choice in
1) sum=`expr $a + $b`
echo The sum of $a and $b is $sum;;
2) difference=`expr $a - $b`
echo The differenceof $a and $b is $difference;;
3) product=`expr $a \* $b`
echo The product of $a and $b is $product;;
4) quotient=`expr $a / $b`
echo The quotient when $a is divided by $b is $quotient;;
5) echo bye
exit;;
esac
```

**OUTPUT :**

```
algin@ubuntu:~/shell$ sh arithmetic.sh
Enter the two numbers
5 6
Menu
 1.Add
 2.Subtract
 3.Multiply
 4.Divide
 5.Exit
Enter your choice:
1
The sum of 5 and 6 is 11
algin@ubuntu:~/shell$ sh arithmetic.sh
Enter the two numbers
23 15
Menu
 1.Add
 2.Subtract
 3.Multiply
 4.Divide
 5.Exit
Enter your choice:
2
The difference of 23 and 15 is 8
algin@ubuntu:~/shell$ sh arithmetic.sh
Enter the two numbers
20 2
Menu
 1.Add
 2.Subtract
 3.Multiply
```

4.Divide

5.Exit

Enter your choice:

3

The product of 20 and 2 is 40

algin@ubuntu:~/shell$ sh arithmetic.sh

Enter the two numbers

21 3

Menu

1.Add

2.Subtract

3.Multiply

4.Divide

5.Exit

Enter your choice:

4

The quotient when 21 is divided by 3 is 7

**RESULT :**

Thus the shell program to perform the basic arithmetic operations has been executed and verified.

**EX.NO.4.8 SUM OF FIRST FIVE NATURAL NUMBER**

**AIM :**

To write shell script to calculate the sum and to print the first five natural numbers.

**ALGORITHM:**

**Step 1:** Start the process.

**Step 2:** Assign sum=0.

**Step 3:** From 1 to 5 add the numbers with sum.

**Step 4:** Print the natural numbers from 1 to 5 and print their sum.

**Step 5:** Stop the process.

**PROGRAM :**

```
# shell program to calculate the sum and to print the first 5 natural numbers
sum=0
echo The first five natural numbers:
for i in 1 2 3 4 5
do
sum=`expr $sum + $i`
echo $i
done
echo The sum of first 5 natural numbers is $sum
```

**OUTPUT :**

```
algin@ubuntu:~/shell$ sh sumn.sh
The first five natural numbers:
1
2
3
4
5
The sum of first 5 natural numbers is 15
```

**RESULT :**

Thus the shell program to calculate the sum and to print the first five natural numbers has been executed and verified.

**EX.NO : 4.9 SUM OF ODD NUMBERS UP TO N**

**AIM :**

To write a Shell program to print and to calculate the sum of odd numbers up to n using until loop.

**ALGORITHM :**

**Step 1:** Start the process.

**Step 2:** Read the input value n.

**Step 3:** Initialize i value as 1, sum as 0 and j as 0.

**Step 4:** Do until i>n then go to step 5 else go to step 6.

**Step 5:** Calculate sum=sum+i, j=j+1, i= i+2 and go to step 4.

**Step 6:** Print the odd numbers up to n and print sum of them.

**Step 7:** Stop the process.

**PROGRAM:**

```
#shell program to print and to calculate the sum of odd numbers upto n using until loop
echo "Enter the n value\t:"
read n
i=1
sum=0
j=0
until [ $i -gt $n ]
do
echo " $i\t"
sum=`expr $sum + $i`
j=`expr $j + 1`
i=`expr $i + 2`
done
echo The sum of the first $j odd numbers upto $n is $sum
```

**OUTPUT :**

```
algin@ubuntu:~/shell$ sh sumodd.sh
Enter the n value      : 10
 1
 3
 5
 7
 9
The sum of the first 5 odd numbers upto 10 is 25
```

**RESULT :**

Thus the shell program to print and to calculate the sum of odd numbers up to n using until loop has been executed and verified.

**EX.NO:4.10 SUM OF EVEN NUMBERS UPTO N**

**AIM :**

To write a shell program to print and to calculate the sum of even numbers upto n using while loop.

**ALGORITHM:**

**Step 1:** Start the process.

**Step 2:** Read the input value n.

**Step 3:** Initialize i value as 2, sum as 0 and j as 0.

**Step 4:** Check if i<n then go to step 5 else go to step 6.

**Step 5:** Calculate sum=sum+i, j=j+1, i= i+2 and go to step 4.

**Step 6:** Print the even numbers up to n and print sum of them

**Step 7:** Stop the process.

**PROGRAM:**

```
#shell program to print and to calculate the sum of even numbers upto n using while loop
echo "Enter the n value\t:"
read n
i=2
sum=0
j=0
while [ $i -le $n ]
do
echo " $i\t"
sum=`expr $sum + $i`
j=`expr $j + 1`
i=`expr $i + 2`
done
echo "\nThe sum of first $j even numbers upto $n is $sum"
```

**OUTPUT :**

```
algin@ubuntu:~/shell$ sh sumeven.sh
Enter the n value      :    10
 2
 4
 6
 8
 10
The sum of first 5 even numbers upto 10 is 30
```

**RESULT :**

Thus the shell program to print and to calculate the sum of even numbers up to n using while loop has been executed and verified.

**EX.NO: 4.11 FACTORIAL OF A GIVEN NUMBER**

**AIM :**

To write a shell program to find factorial of a given number using while loop.

**ALGORITHM :**

**Step 1:** Start the process.

**Step 2:** Read the input value n.

**Step 3:** Initialize i value as 1, fact as 1.

**Step 4:** Check if i<n then go to step 5 else go to step 6.

**Step 5:** Calculate fact=fact*i and i=i+1 and go to step 4.

**Step 6:** Print the factorial of given number n.

**Step 7:** Stop the process.

**PROGRAM :**

```
#shell program to find factorial of a given number using while loop
echo Enter a number
read n
i=1
fact=1
while [ $i -le $n ]
do
fact=`expr $fact \* $i`
i=`expr $i + 1`
done
echo Factorial value of $n is $fact
```

**OUTPUT :**

```
algin@ubuntu:~/shell$ sh fact.sh
Enter a number
5
Factorial value of 5 is 120
```

**RESULT :**

Thus the shell program to find factorial of a given number using while loop has been executed and verified.

**EX.NO : 4.12 CHECK ARMSTRONG NUMBER OR NOT**

**AIM :**

To write Shell script to check whether a given number is armstrong or not.

**ALGORITHM :**

**Step 1:** Start the process.

**Step 2:** Read the input value n.

**Step 3:** Initialize sum as 0 and x as n.

**Step 4:** Check if n>0 then go to step 5 else go to step 6.

**Step 5:** Calculate y=n%10, z=y*y*y, sum=sum+z, n=n/10 and go to step 4.

**Step 6:** Check if x=sum then Print "The given number is Armstrong" and go to step 7 else print " The given number is not Armstrong" and go to step 7.

**Step 7:** Stop the process.

**PROGRAM :**

```
#Shell program to check whether a given number is armstrong or not.
echo enter a number
read n
x=$n
sum=0
while [ $n -gt 0 ]
do
y=`expr $n % 10`
z=`expr $y \* $y \* $y`
sum=`expr $sum + $z`
n=`expr $n / 10`
done
if [ $x -eq $sum ]
then
echo $x is an armstrong number
else
echo $x is not an armstrong number
fi
```

**OUTPUT :**

algin@ubuntu:~/shell$ sh arm.sh

enter a number 123

123 is not an armstrong number

algin@ubuntu:~/shell$ sh arm.sh

enter a number 153

153 is an armstrong number

**RESULT :** Thus the shell program to check whether a given number is Armstrong or not has been executed and verified.

**EX.NO.5** **CPU SCHEDULING**

## 1. FIRST COME FIRST SERVE SCHEDULING ALGORITHM

**AIM**

To schedule the process based on FCFS Scheduling algorithm.

**ALGORITHM**

1. Initialise the variables .
2. Get the no of processess.
3. Get the burst times.
4. Calculate turn around time and waiting time
5. Find the average turn around time and waiting time and print them.

**PROGRAM**

```
#include<stdio.h>
#include<string.h>
main()
{
        int i,j,np,b[100],wt[100],tat[100],twt=0,temp,l,ttat=0;
        float awt=0.0;
        char p[10][10],tem[10];
        printf("ENTER NO OF PROCESS:");
        scanf("%d",&np);
        printf("ENTER PROCESS NAME:");
        for(i=0;i<np;i++)
        {
                scanf("%s",p[i]);
        }
        for(i=0;i<np;i++)
        {
                printf("ENTER BURST TIME FOR %s:",p[i]);
                scanf("%d",&b[i]);
        }
        wt[0]=0;
```

```
for(i=1;i<np;i++)
{
        wt[i]=wt[i-1]+b[i-1];
        twt=twt+wt[i];
}
for(i=0;i<np;i++)
{
        tat[i]=wt[i]+b[i];
        ttat=ttat+tat[i];
}
awt=(float)twt/(float)np;
printf("\nPROCESS\t BURST TIME\tWAITING TIME\tTURN AROUND
TIME:\n");
for(i=0;i<np;i++)
{
        printf("%s\t d\t\t%d\t\t%d\n",p[i],b[i],wt[i],tat[i]);
}
printf("TOTAL WAITING TIME:%d",twt);
printf("\nAVERAGE WAITING TIME:%f",awt);
printf("\ngantt chart:\n");
l=strlen(p[1]);
for(i=0;i<=tat[np-1]+(np*l);i++)
{
        printf("-");
}
printf("\n|");
j=0;
for(i=1;i<=tat[np-1]+(np*l);i++)
{
        if(i==tat[j])
        {
        printf("%s|",p[j]);
        j++;
        }
```

```
                else
                        printf(" ");
        }
        printf("\n");
        for(i=0;i<=tat[np-1]+(np*l);i++)
        {
                printf("-");
        }
        printf("\n0");
        j=0;
        for(i=1;i<tat[np-1]+(np*l);i++)
        {
                if(i==tat[j]+l*(j))
                {
                printf("%d",tat[j]);
                j++;
                }
                else
                        printf(" ");
        }
}
```

**OUTPUT:**

```
ENTER NO OF PROCESS:5
ENTER PROCESS NAME:p1
p2
p3
p4
p5
ENTER BURST TIME FOR p1:3
ENTER BURST TIME FOR p2:2
ENTER BURST TIME FOR p3:5
ENTER BURST TIME FOR p4:7
ENTER BURST TIME FOR p5:1
```

| PROCESS | BURST TIME | WAITING TIME | TURN AROUNDTIME: |
|---|---|---|---|
| p1 | 3 | 0 | 3 |
| p2 | 2 | 3 | 5 |
| p3 | 5 | 5 | 10 |
| p4 | 7 | 10 | 17 |
| p5 | 1 | 17 | 18 |

TOTAL WAITING TIME:35

AVERAGE WAITING TIME:7.000000

**Gantt chart:**

```
-----------------------------------
| p1| p2|   p3|    p4|p5|
-----------------------------------
0  3   5     10      17 18
```

## 5.2. SHORTEST JOB FIRST SCHEDULING ALGORITHM

**AIM**

To schedule the process based on SJF Scheduling algorithm.

**ALGORITHM**

1. Initialise the variables.
2. Get the no of processes.
3. Get the burst times.
4. Arrange the jobs based on their burst times with the job with the shortest burst time first.
5. Calculate turn around time and waiting time
6. Find the average turn around time and waiting time and print them.

**PROGRAM**

```
#include<stdio.h>
#include<string.h>
main()
```

```
{
        int i,j,np,b[100],wt[100],tat[100],twt=0,temp,l;
        float awt=0.0;
        char p[10][10],tem[10];
        printf("ENTER NO OF PROCESS:");
        scanf("%d",&np);
        printf("ENTER PROCESS NAME:");
        for(i=0;i<np;i++)
        {
                scanf("%s",p[i]);
        }
        for(i=0;i<np;i++)
        {
                printf("ENTER BURST TIME: %s:",p[i]);
                scanf("%d",&b[i]);
        }
        for(i=0;i<np-1;i++)
        {
                for(j=i+1;j<np;j++)
                {
                        if(b[i]>b[j])
                        {
                        temp=b[i];
                        b[i]=b[j];
                        b[j]=temp;
                        strcpy(tem,p[j]);
                        strcpy(p[j],p[i]);
                        strcpy(p[i],tem);
                        }
                }
        }
        wt[0]=0;
        for(i=1;i<np;i++)
        {
```

```
        wt[i]=wt[i-1]+b[i-1];
        twt=twt+wt[i];
}
for(i=0;i<np;i++)
{
        tat[i]=wt[i]+b[i];
}
awt=(float)twt/(float)np;
printf("\nPROCESS\t BURST TIME\tWAITING TIME
                        \tTURN AROUND TIME:\n");
for(i=0;i<np;i++)
{
        printf("%s\t %d\t\t%d\t\t%d\n",p[i],b[i],
                                wt[i],tat[i]);
}
printf("TOTAL WAITING TIME:%d",twt);
printf("\nAVERAGE WAITING TIME:%f",awt);
printf("\ngantt chart:\n");
l=strlen(p[1]);
for(i=0;i<=tat[np-1]+(np*l);i++)
{
        printf("-");
}
printf("\n|");
j=0;
for(i=1;i<=tat[np-1]+(np*l);i++)
{
        if(i==tat[j])
        {
                printf("%s|",p[j]);
                j++;
        }
        else
                printf(" ");
```

```
        }
        printf("\n");
        for(i=0;i<=tat[np-1]+(np*l);i++)
        {
                printf("-");
        }
        printf("\n0");
        j=0;
        for(i=1;i<tat[np-1]+(np*l);i++)
        {
                if(i==tat[j]+l*(j))
                {
                        printf("%d",tat[j]);
                        j++;
                }
                else
                printf(" ");
        }
}
```

**OUTPUT:**

```
ENTER NO OF PROCESS:5
ENTER PROCESS NAME:p1
p2
p3
p4
p5
ENTER BURST TIME: p1:4
ENTER BURST TIME: p2:2
ENTER BURST TIME: p3:7
ENTER BURST TIME: p4:1
ENTER BURST TIME: p5:3
PROCESS BURSTTIME WAITING TIME TURNAROUNDTIME:
p4              1               0               1
p2              2               1               3
```

p5 3 3 6

p1 4 6 10

p3 7 10 17

TOTAL WAITING TIME:20

AVERAGE WAITING TIME:4.000000

**Gantt chart:**

**----------**

**|p4| p2| p5| p1| p3|**

**----------**

**01 3 6 10 17**

## 5.3. PRIORITY SCHEDULING ALGORITHM

**AIM**

To schedule the process based on Priority Scheduling algorithm.

**ALGORITHM**

1. Initialize the variables.
2. Get the no of processes.
3. Get their burst times.
4. Get their priority.
5. Arrange the jobs based on their priority.
6. Calculate turn around time and waiting time.
7. Find the average turn around time and waiting time and print them.

**PROGRAM**

```
#include<stdio.h>
#include<string.h>
main()
{
    int i,j,np,b[100],wt[100],tat[100],twt=0,temp,l,pr[10],t;
    float awt=0.0;
    char p[10][10],tem[10];
    printf("ENTER NO OF PROCESS:");
    scanf("%d",&np);
    printf("ENTER PROCESS NAME:");
```

```
for(i=0;i<np;i++)
{
        scanf("%s",p[i]);
}
for(i=0;i<np;i++)
{
        printf("ENTER BURST TIME %s:",p[i]);
        scanf("%d",&b[i]);
}
for(i=0;i<np;i++)
{
        printf("ENTER %s PRIORITY:",p[i]);
        scanf("%d",&pr[i]);
}
for(i=0;i<np-1;i++)
{
        for(j=i+1;j<np;j++)
        {
                if(pr[i]>pr[j])
                {
                temp=b[i];
                b[i]=b[j];
                b[j]=temp;
                strcpy(tem,p[j]);
                strcpy(p[j],p[i]);
                strcpy(p[i],tem);
                t=pr[i];
                pr[i]=pr[j];
                pr[j]=t;
                }
        }
}
wt[0]=0;
for(i=1;i<np;i++)
```

```
{
        wt[i]=wt[i-1]+b[i-1];
        twt=twt+wt[i];
}
for(i=0;i<np;i++)
{
        tat[i]=wt[i]+b[i];
}
awt=(float)twt/(float)np;
printf("\nPROCESS\t BURST TIME\tWAITING TIME\tTURN AROUND
TIME:\n");
for(i=0;i<np;i++)
{
        printf("%s\t %d\t\t%d\t\t%d\n",p[i],b[i],wt[i],tat[i]);
}
printf("TOTAL WAITING TIME:%d",twt);
printf("\nAVERAGE WAITING TIME:%f",awt);
printf("\ngantt chart:\n");
l=strlen(p[1]);
for(i=0;i<=tat[np-1]+(np*l);i++)
{
        printf("-");
}
printf("\n|");
j=0;
for(i=1;i<=tat[np-1]+(np*l);i++)
{
        if(i==tat[j])
        {
        printf("%s|",p[j]);
        j++;
        }
        else
                printf(" ");
```

```
        }
        printf("\n");
        for(i=0;i<=tat[np-1]+(np*l);i++)
        {
                printf("-");
        }
        printf("\n0");
        j=0;
        for(i=1;i<tat[np-1]+(np*l);i++)
        {
                if(i==tat[j]+l*(j))
                {
                printf("%d",tat[j]);
                j++;
                }
                else
                        printf(" ");
        }
}
```

**OUTPUT:**

```
ENTER NO OF PROCESS:5
ENTER PROCESS NAME:p1
p2
p3
p4
p5
ENTER BURST TIME p1:3
ENTER BURST TIME p2:5
ENTER BURST TIME p3:6
ENTER BURST TIME p4:4
ENTER BURST TIME p5:1
ENTER p1 PRIORITY:3
ENTER p2 PRIORITY:1
ENTER p3 PRIORITY:4
```

ENTER p4 PRIORITY:5

ENTER p5 PRIORITY:2

| PROCESS | BURST TIME | WAITING TIME | TURN AROUND TIME: |
|---|---|---|---|
| p2 | 5 | 0 | 5 |
| p5 | 1 | 5 | 6 |
| p1 | 3 | 6 | 9 |
| p3 | 6 | 9 | 15 |
| p4 | 4 | 15 | 19 |

TOTAL WAITING TIME:35

AVERAGE WAITING TIME:7.000000

**Gantt chart:**

**----------**

**| p2|p5| p1| p3| p4|**

**----------**

**0 5 6 9 15 19 */**

## 5.4. ROUND ROBIN SCHEDULING

**AIM**

To schedule the process based on Round Robin Scheduling algorithm.

**ALGORITHM**

1. Initialise the variables .
2. Get the no of jobs.
3. Get the burst times.
4. Get the time slice.
5. Calculate turn around time and waiting time.
6. Find the average turn around time and waiting time and print them.

**PROGRAM**

#include<stdio.h>

```
main()
{
int i,n,process[10],burst[10], arrival[10], temp_comp[10], wait[10], turn[10], finish[10],
count, ctime=0, tslice, tot_wait=0, tot_turn=0;
boolean flag[10];
printf("\n Enter the no. of processe \n");
scanf("%d", &n);
count=n;

printf("\n Enter the time slice\n");
scanf("%d", &tslice);
for (i=1;i<=n;i++)
{
        printf("\n Enter the process id and burst time for the %d process",i);
        scanf("%d%d", &process[i],&burst[i]);
        arrival[i]=0;
        temp_comp[i]=0;
        wait[i]=0;
        turn[i]=0;
        flag[i]=false;
}
while(count>0)
{
        for (i=1;i<=n;i++)
        {
                if (flag[i]==false)
                {
                        wait[i]=wait[i]+(ctime-temp_comp[i]);
                        if (burst[i]<=tslice)
                        {
                                ctime=ctime+burst[i];
                                burst[i]=0;
                                finish[i]=ctime;
                        }
```

```
                else
                {
                        ctime=ctime+tslice;
                        burst[i]=burst[i]-tslice;
                }
                temp_comp[i]=ctime;
                if (burst[i]==0)
                {
                        flag[i]=true;
                        count=count-1;
                }
        }
    }
}
printf("\n Process\tBurstTime\t WaitingTime\t TurnAroundTime");
for(i=1;i<=n;i++)
{
    turn[i]=finish[i]-arrival[i];
    tot_wait=tot_wait+wait[i];
    tot_turn=tot_turn+turn[i];
    printf("\t%d\t%d\t%d\t%d",process[i], burst[i], wait[i], turn[i]);
}
printf("\n Average Waiting time is %d \n Average Turn around time is %d", (tot_wait/n),
(tot_turn/n));
}
```

**OUTPUT**

$ ./a.out

Enter the number of jobs 5

Enter the 0 th burst time 6

Enter the 1 th burst time 9

Enter the 2 th burst time 7

Enter the 3 th burst time 3

Enter the 4 th burst time 7

Enter the time slice 2

| job | burst | AT | start | finish | WT | TAT |
|---|---|---|---|---|---|---|
| 0 | 9 | 0 | 0 | 9 | 0 | 9 |
| 1 | 6 | 0 | 9 | 15 | 9 | 15 |
| 3 | 3 | 0 | 18 | 15 | 18 | 18 |
| 4 | 7 | 0 | 18 | 25 | 18 | 25 |
| 2 | 7 | 0 | 25 | 32 | 25 | 32 |

Avg turn around = 19

Avg waiting time = 13

**RESULT:**

Thus the CPU scheduling algorithms are implemented successfully.

**EX.NO:6** **IMPLEMENT SEMAPHORES**

**AIM:**

To write a C program to implement the semphores.

**PROGRAM:**

**//Implement the Producer – Consumer problem using semaphores**

```
#include<stdio.h>
#include<stdlib.h>
int mutex=1,full=0,empty=3,x=0;
void main()
{
int n;
void producer();
void consumer();
int wait(int);
int signal(int);
printf("\n1.PRODUCER\n2.CONSUMER\n3.EXIT\n");
while(1)
{
printf("\nENTER YOUR CHOICE\n");
scanf("%d",&n);
switch(n)
{
case 1:
if((mutex==1)&&(empty!=0))
producer();
else
printf("BUFFER IS FULL");
break;
case 2:
if((mutex==1)&&(full!=0))
consumer();
else
```

```
printf("BUFFER IS EMPTY"); break;
case 3:
exit(0);
break;
}
}
}
int wait(int s)
{
return(--s);
}
int signal(int s)
{
return(++s);
}
void producer()
{
 mutex=wait(mutex);
 full=signal(full);
 empty=wait(empty);
 x++;
printf("\nproducer produces the item%d",x);
 mutex=signal(mutex);
}
void consumer()
{
 mutex=wait(mutex);
full=wait(full);
empty=signal(empty);
 printf("\n consumer consumes item%d",x);
x--;
mutex=signal(mutex);
}
```

**OUTPUT:**

```
c3104068@rmk-desktop:~$ g++ semaphore.c
c3104068@rmk-desktop:~$ ./a.out
1.PRODUCER
2. CONSUMER
3. EXIT
ENTER YOUR CHOICE: 1
producer produces the item1
ENTER YOUR CHOICE:1
producer produces the item2
ENTER YOUR CHOICE:1
producer produces the item3
ENTER YOUR CHOICE:2
 consumer consumes item3
ENTER YOUR CHOICE:2
 consumer consumes item2
ENTER YOUR CHOICE:2
 consumer consumes item1
ENTER YOUR CHOICE:1
producer produces the item1
ENTER YOUR CHOICE:2
 consumer consumes item1
ENTER YOUR CHOICE:2
BUFFER IS EMPTY
ENTER YOUR CHOICE:3
```

**RESULT:**

Thus the semaphore program is executed successfully.

**EX.NO:7** **IMPLEMENT SHARED MEMORY AND IPC**

**AIM:**

To implement the Shared memory and IPC using C program.

**IPC:(INTER PROCESS COMMUNICATION)**

**PROGRAM:**

```
//Developing Application using Inter Process communication(using pipes)

#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
int main()
{
int pid,pfd[2],n,a,b,c;
if(pipe(pfd)==-1)
{
printf("\nError in pipe connection\n");
exit(1);
}
pid=fork();
if(pid>0)
{
printf("\nParent Process");
printf("\n\n\tFibonacci Series");
printf("\nEnter the limit for the series:");
scanf("%d",&n);
close(pfd[0]);
write(pfd[1],&n,sizeof(n));
close(pfd[1]);
exit(0);
}
```

```
else
{
close(pfd[1]);
read(pfd[0],&n,sizeof(n));
printf("\nChild Process");
a=0;
b=1;
close(pfd[0]);
printf("\nFibonacci Series is:");
printf("\n\n%d\n%d",a,b);
while(n>2)
{
c=a+b;
printf("\n%d",c); a=b; b=c; n--;
}
}
printf(“\n”);
}
```

**OUTPUT:**

```
c3104068@rmk-desktop:~$ g++ ipc.c

c3104068@rmk-desktop:~$ ./a.out
Parent Process
Fibonacci Series
Enter the limit for the series:8
Child Process
Fibonacci Series is: 0 1 1 2 3 5 8 13
```

**RESULT:**

Thus the implementation of the Shared memory and IPC program is executed successfully.

**EX.NO:8 IMPLEMENT BANKERS ALGORITHM FOR DEAD LOCK AVOIDANCE**

**AIM:**

To implement the Bankers Algorithm for Dead Lock Avoidance

**PROGRAM:**

```
#include<stdio.h>
#include<string.h>
main()
{
    int i,j,k,l,ava[100],maxi[100][100],allo[100][100],need[100][100],np,n,work[100];
    char p[10][10],tem[10],ar[10][10];
    int fin[100];
    printf("\nENTER NO OF PROCESS:");
    scanf("%d",&np);
    printf("\nENTER PROCESS NAME:");
    for(i=0;i<np;i++)
    {
        scanf("%s",p[i]);
    }
    printf("\nEnter no.of Resources:");
    scanf("%d",&n);
    printf("\nEnter Resources name:");
    for(i=0;i<n;i++)
    {
        scanf("%s",ar[i]);
    }
    printf("\nEnter Available:");
    for(i=0;i<n;i++)
    {
        scanf("%d",&ava[i]);
    }
    printf("\nEnter Allocation:");
```

```
for(i=0;i<np;i++)
{
	printf("\nEnter %s :",p[i]);
	for(j=0;j<n;j++)
	{
		scanf("%d",&allo[i][j]);
	}
}
printf("\nEnter Maximum:");
for(i=0;i<np;i++)
{
	printf("\nEnter %s :",p[i]);
	for(j=0;j<n;j++)
	{
		scanf("%d",&maxi[i][j]);
	}
}
for(i=0;i<np;i++)
{
	for(j=0;j<n;j++)
	{
	need[i][j]=maxi[i][j]-allo[i][j];
	}
}
printf(" ");
for(i=0;i<n;i++)
{
	printf("%s\t",ar[i]);
}
for(i=0;i<np;i++)
{
	printf("\n%s:",p[i]);
	for(j=0;j<n;j++)
	{
```

```
            printf("%d\t",need[i][j]);
        }
    }
    for(i=0;i<n;i++)
    {
        work[i]=ava[i];
    }
    for(i=0;i<n;i++)
    {
        fin[i]=0;
    }
    for(l=0;l<np;l++)
    {
        for(i=0;i<np;i++)
        {       if(!fin[i])
            {
                int f=1;
                for(j=0;j<n;j++)
                    {
                        if(work[j]<need[i][j])
                        f=0;
                    }
                if(f)
                {
                    fin[i]=1;
                    for(k=0;k<n;k++)
                    {
                        work[k]+=allo[i][k];
                    }
                }
            }
        }
    }
    int f1=1;
```

```
        for(i=0;i<np;i++)
        {
                if(fin[i]==0)
                {
                        printf("\nNot Safe\n");
                        f1=0;
                        break;
                }
        }
        if(f1)
                printf("\nSystem is Safe\n");

}
```

**OUTPUT:**

```
c3104068@rmk-desktop:~$ g++ bank.c

c3104068@rmk-desktop:~$ ./a.out
ENTER NO OF PROCESS:5
ENTER PROCESS NAME:
p0

p1

p2

p3

p4
Enter no.of Resources:4
Enter Resources name:
A
B
```

C

D

Enter Available:1 5 2 0

Enter Allocation:

Enter p0 :0 0 1 2

Enter p1 :1 0 0 0

Enter p2 :1 3 5 4

Enter p3 :0 6 3 2

Enter p4 :0 0 1 4

Enter Maximum:

Enter p0 :0 0 1 2

Enter p1 :1 7 5 0

Enter p2 :2 3 5 6

Enter p3 :0 6 5 2

Enter p4 :0 6 5 6

| | A | B | C | D |
|---|---|---|---|---|
| p0: | 0 | 0 | 0 | 0 |
| p1: | 0 | 7 | 5 | 0 |
| p2: | 1 | 0 | 0 | 2 |
| p3: | 0 | 0 | 2 | 0 |
| p4: | 0 | 6 | 4 | 2 |

System is Safe

**RESULT:**

Thus the implementation of Bankers Algorithm for Dead Lock Avoidance is executed successfully.

**EX.NO:9 IMPLEMENT DEADLOCK DETECTION ALGORITHM**

**AIM:**

To implement the Deadlock Detection Algorithm

**PROGRAM:**

```
#include  <stdio.h>
#include <conio.h>
void main()
{
int found,flag,l,p[4][5],tp,c[4][5],i,j,k=1,m[5],r[5],a[5],temp[5],sum=0;
clrscr();
printf("enter total no of processes");
scanf("%d",&amp;tp);
printf("enter clain matrix");
for(i=1;i&lt;=4;i++)
for(j=1;j&lt;=5;j++)
{
scanf("%d",&amp;c[i][j]);
}
printf("enter allocation matrix");
for(i=1;i&lt;=4;i++)
for(j=1;j&lt;=5;j++)
{
scanf("%d",&amp;p[i][j]);
}
printf("enter resource vector:\n");
for(i=1;i&lt;=5;i++)
{
scanf("%d",&amp;r[i]);
}
printf("enter availability vector:\n");
for(i=1;i&lt;=5;i++)
{
scanf("%d",&amp;a[i]);
```

```
temp[i]=a[i];
}
for(i=1;i&lt;=4;i++)
{
sum=0;
for(j=1;j&lt;=5;j++)
{
sum+=p[i][j];
}
if(sum==0)
{
m[k]=i;
k++;
} }
for(i=1;i&lt;=4;i++)
{
for(l=1;l&lt;k;l++)
if(i!=m[l])
{
flag=1;
for(j=1;j&lt;=5;j++)
if(c[i][j]&gt;temp[j])
{
flag=0;
break;
} }
if(flag==1)
{
m[k]=i;
k++;
for(j=1;j&lt;=5;j++)
temp[j]+=p[i][j];
} }
printf("deadlock causing processes are:");
```

```
for(j=1;j&lt;=tp;j++)
{
found=0;
for(i=1;i&lt;k;i++)
{
if(j==m[i])
found=1;
}
if(found==0)
printf("%d\t",j);
}
getch();
}
```

**OUTPUT:**

**INPUT:**

enter total no. of processes : 4

enter claim matrix :

0 1 0 0 1

0 0 1 0 1

0 0 0 0 1

1 0 1 0 1

enter allocation matrix :

1 0 1 1 0

1 1 0 0 0

0 0 0 1 0

0 0 0 0 0

enter resource vector :

2 1 1 2 1

enter the availability vector :

0 0 0 0 1

OUTPUT :

deadlock causing processes are : 1 2

**Result:**

Thus the deadlock detection algorithm is implemented successfully.

**Ex No 10 MULTITHREADING AND SYNCHRONIZATION**

**AIM**

To write a C program to implement multithreading and synchronization

**PROGRAM**

```
#include <unistd.h>

#include <sys/types.h>

#include <errno.h>

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <string.h>

#include <semaphore.h>

void handler ( void *ptr );

pthread_barrier_t barrier;

int worker = 2; // number of workers

int job = 4; // number of jobs for each worker

int main()

{

   int i = 0;

   pthread_t thread_a;

   pthread_barrier_init(&barrier, NULL, worker);

   for (i; i < worker; i++)

   {
```

```
        int *n_workers = malloc(sizeof(*n_workers));
        *n_workers = i;
        pthread_create (&thread_a, NULL, (void *) &handler, n_workers);
    }
    pthread_join(thread_a, NULL);
    pthread_barrier_destroy(&barrier);
    pthread_exit(0);
}
void handler ( void *ptr )
{
    int x = *((int *) ptr);
    int i = 0;
    for (i; i < job; i++)
    {
        printf("Worker %d: Doing Job %d\n", x, i);
        pthread_barrier_wait(&barrier);
    }
}
```

**OUTPUT**

Worker 0: Doing Job 0

Worker 1: Doing Job 0

Worker 1: Doing Job 1

Worker 0: Doing Job 1

Worker 0: Doing Job 2

Worker 1: Doing Job 2

Worker 0: Doing Job 3

Worker 1: Doing Job 3

**RESULT**

Thus the C program to for multithreading and thread synchronization was executed successfully

**EX NO 11** **MEMORY ALLOCATION**

**AIM**

To write a C program to implement first fit, Worst fit and best fit memory allocation method

**A) FIRST-FIT**

```
#include<stdio.h>
#include<conio.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];
clrscr();
printf("\n\tMemory Management Scheme - Worst Fit");
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]);
getch();
}
```

**INPUT**

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

**OUTPUT**

| File No | File Size | Block No | Block Size | Fragment |
|---|---|---|---|---|
| 1 | 1 | 3 | 7 | 6 |
| 2 | 4 | 1 | 5 | 1 |

**B)WORST-FIT**

**PROGRAM**

```
#include<stdio.h>
#include<conio.h>
#define max 25
void main()
{
int frag[max],b[max],f[max],i,j,nb,nf,temp;
static int bf[max],ff[max];
clrscr();
printf("\n\tMemory Management Scheme - First Fit");
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]);
getch();
}
```

**INPUT**

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

**OUTPUT**

| File No | File Size | Block No | Block Size | Fragment |
|---|---|---|---|---|
| 1 | 1 | 1 | 5 | 4 |
| 2 | 4 | 3 | 7 | 3 |

**C) BEST-FIT**

**PROGRAM**

```
#include<stdio.h>
#include<conio.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];
clrscr();
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]);
getch();
}
```

**INPUT**

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

**OUTPUT**

| File No | File Size | Block No | Block Size | Fragment |
|---|---|---|---|---|
| 1 | 1 | 2 | 2 | 1 |
| 2 | 4 | 1 | 5 | 1 |

**RESULT**

Thus the program for First fit,Worst fit and Best fit memory allocation is implemented successfully

**EX.NO.12 IMPLEMENT PAGING TECHNIQUE OF MEMORY ANAGEMENT**

**AIM:**

To write a C program to implement Paging Technique of memory management.

**PROGRAM:**

```
#include<stdio.h>
#include<conio.h>
main()
{
        int np,ps,i;
        int *sa;
        clrscr();
        printf("Enter how many pages\n");
        scanf("%d",&np);printf("Enter the page size \n");
        scanf("%d",&ps);
        sa=(int*)malloc(2*np);
        for(i=0;i<np;i++)
        {
        sa[i]=(int)malloc(ps);
        printf("page%d\t address %u\n",i+1,sa[i]);
        }
        getch();
}
```

**INPUT:**

Enter how many pages: 5

Enter the page size:4

**OUTPUT:**

Page1 address: 1894

Page2 address: 1902

Page3 address: 1910

Page4 address: 1912

Page5 address: 1926

**RESULT:**

Thus the paging is simulated using memory management scheme.

**EX.NO:13 IMPLEMENT ALL PAGE REPLACEMENT ALGORITHMS**

**AIM:**

To write C program to implement the Page replacement algorithms.

**13 a) FIFO (First In First Out)**

**PROGRAM:**

```
#include<stdio.h>
int m,n,i,j,k,flag,count=0,refer[100],page_frame[100][2],fault=0,min,no_frames;
void replace()
{
    for(i=0;i<n;i++)
    {
        flag=1;
        for(j=0;j<no_frames;j++)
            if(refer[i]==page_frame[j][0])
            {
                m=j;
                flag=0;
            }
        if(flag)
        {
            fault++;
            min=32000;
            for(j=0;j<no_frames;j++)
                if(page_frame[j][1]<min)
                {
                    min=page_frame[j][1];
                    k=j;
                }
            page_frame[k][0]=refer[i];
            page_frame[k][1]=++count;
            for(j=0;j<no_frames;j++)
```

```
                printf("%d\t",page_frame[j][0]);
            printf("\n");
        }
        else
            printf("no page fault\n");
    }
    printf("number of page fault is:%d\n",fault);
}
int main()
{
    printf("\nEnter the number of reference:");
    scanf("%d",&n);
    printf("\nEnter the number of frames:");
    scanf("%d",&no_frames);
    printf("\nEnter the reference string:");
    for(i=0;i<n;i++)
        scanf("%d",&refer[i]);
    printf("\t\t\tFIFO ALGORITHM \n");
    for(i=0;i<no_frames;i++)
    {
        page_frame[i][0]=-1;
        page_frame[i][1]=count;
    }
    replace();
    fault=0;
    count=0;
    return 0;
}
```

**OUTPUT**:

```
$ cc fifo.c
$ ./a.out
Enter the number of reference:10
Enter the number of frames:3
Enter the reference string:7 0 1 2 0 3 0 4 2 6
                    FIFO ALGORITHM
7       -1      -1
7       0       -1
7       0       1
2       0       1
no page fault
2       3       1
2       3       0
4       3       0
4       2       0
4       2       6
number of page fault is:9
```

**RESULT:**

Thus the implementation of the page replacement algorithm based on FIFO is executed successfully.

**13 B) LRU (LEAST RECENTLY USED)**

**AIM:**

To implement the page replacement algorithm based on LRU.

**PROGRAM:**

```
#include<stdio.h>
int m,n,i,j,k,flag,count=0,refer[100],page_frame[100][2],fault=0,min,no_frames;
void replace()
{
    for(i=0;i<n;i++)
    {
        flag=1;
        for(j=0;j<no_frames;j++)
            if(refer[i]==page_frame[j][0])
            {
                m=j;
                flag=0;
            }
        if(flag)
        {
            fault++;
            min=32000;
            for(j=0;j<no_frames;j++)
                if(page_frame[j][1]<min)
                {
                    min=page_frame[j][1];
                    k=j;
                }
            page_frame[k][0]=refer[i];
            page_frame[k][1]=++count;
            for(j=0;j<no_frames;j++)
                printf("%d\t",page_frame[j][0]);
            printf("\n");
```

```
            }
            else
            {
                printf("no page fault\n");
                page_frame[m][1]=++count;
            }
        }
        printf("number of page fault is:%d\n",fault);
}
int main()
{
        printf("\nEnter the number of reference:");
        scanf("%d",&n);
        printf("\nEnter the number of frames:");
        scanf("%d",&no_frames);
        printf("\nEnter the reference string:");
        for(i=0;i<n;i++)
            scanf("%d",&refer[i]);
        fault=0;
        count=0;
        printf("\t\t\tLRU ALGORITHM \n");
        for(i=0;i<no_frames;i++)
        {
        page_frame[i][0]=-1;
        page_frame[i][1]=count;
        }
        replace();
        return 0;
}
```

**OUTPUT:**

```
$ cc lru.c
$ ./a.out

Enter the number of reference:10

Enter the number of frames:3

Enter the reference string:7 0 1 2 0 3 0 4 2 6
                    LRU ALGORITHM
7       -1      -1
7       0       -1
7       0       1
2       0       1
no page fault
2       0       3
no page fault
4       0       3
4       0       2
4       6       2
number of page fault is:8
```

**RESULT:**

Thus the implementation of page replacement algorithm based on LRU.

## 13 C) LFU (LEAST FREQUENTLY USED)

**AIM:**

To implement the page replacement algorithm based on LFU

**PROGRAM:**

```
#include<stdio.h>
int m,n,i,j,k,flag,count=0,refer[100],page_frame[100][2],fault=0,min,no_frames;
void replace()
{
    for(i=0;i<n;i++)
    {
        flag=1;
        for(j=0;j<no_frames;j++)
            if(refer[i]==page_frame[j][0])
            {
                m=j;
                flag=0;
            }
        if(flag)
        {
            fault++;
            min=32000;
            for(j=0;j<no_frames;j++)
                if(page_frame[j][1]<min)
                {
                    min=page_frame[j][1];
                    k=j;
                }
            page_frame[k][0]=refer[i];
            page_frame[k][1]=++count;
            for(j=0;j<no_frames;j++)
                printf("%d\t",page_frame[j][0]);
            printf("\n");
```

```
            }
            else
            {
                printf("no page fault\n");
                page_frame[m][1]=++count;
            }
        }
        printf("number of page fault is:%d\n",fault);
}
int main()
{
        printf("\nEnter the number of reference:");
        scanf("%d",&n);
        printf("\nEnter the number of frames:");
        scanf("%d",&no_frames);
        printf("\nEnter the reference string:");
        for(i=0;i<n;i++)
            scanf("%d",&refer[i]);
        fault=0;
        count=0;
        printf("\t\t\tLRU ALGORITHM \n");
        for(i=0;i<no_frames;i++)
        {
        page_frame[i][0]=-1;
        page_frame[i][1]=count;
        }
        replace();
        return 0;
}
```

**OUTPUT:**

```
$ cc lru.c
$ ./a.out

Enter the number of reference:10

Enter the number of frames:3

Enter the reference string:7 0 1 2 0 3 0 4 2 6
                    LRU ALGORITHM
7       -1      -1
7       0       -1
7       0       1
2       0       1
no page fault
2       0       3
no page fault
4       0       3
4       0       2
4       6       2
number of page fault is:8
```

**RESULT:**

Thus the implementation of the page replacement algorithm based on LRU is executed successfully.

**EX.NO:14 IMPLEMENT ALL FILE ORGANIZATION TECHNIQUES**

**14 A) SINGLE LEVEL DIRECTORY**

**AIM:**

To implement the file organisation technique using single level directory.

**PROGRAM:**

```
#include<stdio.h>
struct directory
{
    char name[30];
    int no;
    char file[50][30];
};
int main()
{
    struct directory dir[50];
    int n,i,j;
    printf("Enter the no. of Directories:");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        printf("Enter the name of the Directory %d:",i+1);
        scanf("%s",dir[i].name);
        printf("Enter the no.of files in the directory %s:",dir[i].name);
        scanf("%d",&dir[i].no);
        if(dir[i].no!=0)
            printf("Enter the file names:\n");
        for(j=0;j<dir[i].no;j++)
            scanf("%s",dir[i].file[j]);
    }
    printf("\nDirectory Structure..\n");
    for(i=0;i<n;i++)
```

```
    {
        printf("%s\n",dir[i].name);
        for(j=0;j<dir[i].no;j++)
            printf("\t%s\n",dir[i].file[j]);
    }
    return 0;
}
```

**OUTPUT:**

```
Enter the no. of Directories:2
Enter the name of the Directory 1:IIyear
Enter the no.of files in the directory IIyear:3
Enter the file names:
mpmc.txt
os.txt
daa.txt
Enter the name of the Directory 2:IIIyear
Enter the no.of files in the directory IIIyear:3
Enter the file names:
npm.txt
cns.txt
ooad.txt
Directory Structure..
IIyear
    mpmc.txt
    os.txt
    daa.txt
IIIyear
    npm.txt
    cns.txt
    ooad.txt
```

**RESULT:**

Thus the implementation of single level directory file organization technique is executed successfully.

**EX.NO:14 B) TWO LEVEL DIRECTORY**

**AIM*:***

To implement the file organisation technique using two level directory.

**PROGRAM:**

```
#include<stdio.h>
struct subdirectory
{
    char name[30];
    int no;
    char file[50][30];
};
struct directory
{
    char name[30];
    int nf,nsd;
    struct subdirectory sd[50];
    char file[50][30];
};
int main()
{
    struct directory dir[50];
    int n,i,j,k;
    printf("Enter the no. of Directories:");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        printf("Enter the name of the Directory %d:",i+1);
        scanf("%s",dir[i].name);
```

```
        printf("Enter the no.of sub directories in the directory %s:",dir[i].name);
        scanf("%d",&dir[i].nsd);
        if(dir[i].nsd!=0)
                printf("Enter the details of sub directories:\n");
        for(j=0;j<dir[i].nsd;j++)
        {
                printf("Enter the name of the Sub Directory %d:",i+1);
                scanf("%s",dir[i].sd[j].name);
                printf("Enter the no.of files in the Sub Directory %s:",dir[i].sd[j].name);
                scanf("%d",&dir[i].sd[j].no);
                if(dir[i].sd[i].no!=0)
                        printf("Enter the file names of the Sub Directory %s:\n",dir[i].sd[j].name);
                for(k=0;k<dir[i].sd[j].no;k++)
                        scanf("%s",dir[i].sd[j].file[k]);
        }
        printf("Enter the no.of files in the directory %s:",dir[i].name);
        scanf("%d",&dir[i].nf);
        if(dir[i].nf!=0)
                printf("Enter the file names:\n");
        for(j=0;j<dir[i].nf;j++)
                scanf("%s",dir[i].file[j]);
}
printf("\nDirectory Structure..\n");
for(i=0;i<n;i++)
{
        printf("%s\n",dir[i].name);
        for(j=0;j<dir[i].nsd;j++)
        {
                printf("\t%s\n",dir[i].sd[j].name);
                for(k=0;k<dir[i].sd[j].no;k++)
                        printf("\t\t%s\n",dir[i].sd[j].file[k]);
        }
        for(j=0;j<dir[i].nf;j++)
                printf("\t%s\n",dir[i].file[j]);
```

```
    }
    return 0;
}
```

**OUTPUT:**

```
Enter the no. of Directories:2
Enter the name of the Directory 1:IT
Enter the no.of sub directories in the directory IT:2
Enter the details of sub directories:
Enter the name of the Sub Directory 1:IIyear
Enter the no.of files in the Sub Directory IIyear:2
Enter the file names of the Sub Directory IIyear:
a.txt
b.txt
Enter the name of the Sub Directory 1:IIIyear
Enter the no.of files in the Sub Directory IIIyear:1
Enter the file names of the Sub Directory IIIyear:
a.txt
Enter the no.of files in the directory IT:1
Enter the file names:
mark.doc
Enter the name of the Directory 2:CSE
Enter the no.of sub directories in the directory CSE:0
Enter the no.of files in the directory CSE:1
Enter the file names:
staff.doc
Directory Structure..
IT
    IIyear
        a.txt
        b.txt
    IIIyear
```

a.txt

mark.doc

CSE

staff.doc

**Result:**Thus the implementation of two level directory file organization technique is executed successfully.

## EX.NO:14 C) HIERARCHICAL

**AIM:**

To implement the file organisation technique using hierarchical level directory.

**PROGRAM:**

```
#include<stdio.h>
#include<stdlib.h>
struct directory;
struct directory
{
    char name[30];
    int nf,nsd;
    struct directory *sub[50];
    char file[50][30];
};
void mkdir(struct directory *dir)
{
    int j;
    printf("Enter the name of the Directory:");
    scanf("%s",dir->name);
    printf("Enter the no.of sub directories in the directory %s:",dir->name);
    scanf("%d",&dir->nsd);
    if(dir->nsd!=0)
        printf("Enter the details of sub directories:\n");
    for(j=0;j<dir->nsd;j++)
```

```
        {
            dir->sub[j]=(struct directory *)malloc(sizeof(struct directory));
            mkdir(dir->sub[j]);
        }
        printf("Enter the no.of files in the directory %s:",dir->name);
        scanf("%d",&dir->nf);
        if(dir->nf!=0)
            printf("Enter the file names:\n");
        for(j=0;j<dir->nf;j++)
            scanf("%s",dir->file[j]);
}
void display(struct directory *dir,int i)
{
        int j,x;
        for(x=0;x<i;x++)
            printf("\t");
        printf("%s\n",dir->name);
        for(j=0;j<dir->nsd;j++)
            display(dir->sub[j],i+1);
        for(j=0;j<dir->nf;j++)
        {
            for(x=0;x<=i;x++)
                printf("\t");
            printf("%s\n",dir->file[j]);
        }
}
int main()
{
        struct directory dir[50];
        int n,i,j,k;
        printf("Enter the no. of Directories:");
        scanf("%d",&n);
        for(i=0;i<n;i++)
            mkdir(&dir[i]);
```

```
        printf("\nDirectory Structure..\n");
        for(i=0;i<n;i++)
                display(&dir[i],0);
        return 0;
}
```

**OUTPUT:**

```
Enter the no. of Directories:2
Enter the name of the Directory:
it
Enter the no.of sub directories in the directory it:2
Enter the details of sub directories:
Enter the name of the Directory:IIyear
Enter the no.of sub directories in the directory IIyear:2
Enter the details of sub directories:
Enter the name of the Directory:Subject
Enter the no.of sub directories in the directory Subject:0
Enter the no.of files in the directory Subject:5
Enter the file names:
PQT.txt
DAA.txt
OS.txt
MPMC.txt
SE.txt
Enter the name of the Directory:Staff
Enter the no.of sub directories in the directory Staff:0
Enter the no.of files in the directory Staff:1
Enter the file names:
staff.doc
Enter the no.of files in the directory IIyear:1
Enter the file names:
stud.doc
Enter the name of the Directory:IIIyear
Enter the no.of sub directories in the directory IIIyear:2
Enter the details of sub directories:
```

Enter the name of the Directory:Subject
Enter the no.of sub directories in the directory Subject:0
Enter the no.of files in the directory Subject:6
Enter the file names:
NM.txt
NPM.txt
CNS.txt
ES.txt
OOAD.txt
WT.txt
Enter the name of the Directory:Staff
Enter the no.of sub directories in the directory Staff:0
Enter the no.of files in the directory Staff:1
Enter the file names:
staff.doc
Enter the no.of files in the directory IIIyear:1
Enter the file names:
stud.doc
Enter the no.of files in the directory it:2
Enter the file names:
markII.xls
markIII.xls
Enter the name of the Directory:cse
Enter the no.of sub directories in the directory cse:1
Enter the details of sub directories:
Enter the name of the Directory:Staff
Enter the no.of sub directories in the directory Staff:0
Enter the no.of files in the directory Staff:1
Enter the file names:
staff.doc
Enter the no.of files in the directory cse:1
Enter the file names:
staff.xls

Directory Structure..

it

IIyear

Subject

PQT.txt

DAA.txt

OS.txt

MPMC.txt

SE.txt

Staff

staff.doc

stud.doc

IIIyear

Subject

NM.txt

NPM.txt

CNS.txt

ES.txt

OOAD.txt

WT.txt

Staff

staff.doc

stud.doc

markII.xls

markIII.xls

cse

Staff

staff.doc

staff.xls

**RESULT:**

Thus the implementation of hierarchical level directory file organization technique is executed successfully.

**EX.NO 15 IMPLEMENTATION OF FILE ALLOCATION STRATEGIES**

**A) SEQUENTIAL ALLOCATION**

**AIM:**

To implement file allocation strategies using sequential allocation

**PROGRAM:**

```
#include<stdio.h>
int main()
{
int f[50],i,st,j,len,c,k;
//clrscr();
for(i=0;i<50;i++)
f[i]=0;
X:
printf("\n Enter the starting block & length of file");
scanf("%d%d",&st,&len);
for(j=st;j<(st+len);j++)
if(f[j]==0)
{
f[j]=1;
printf("\n%d->%d",j,f[j]);
}
else
{
break;
}
if(j==(st+len))
printf("\n the file is allocated to disk");
printf("\n if u want to enter more files?(y-1/n-0)");
scanf("%d",&c);
if(c==1)
goto X;
else
```

```
exit(0);
}
```

**OUTPUT**

```
root@rmk-desktop:~/Desktop# ./a.out
Enter the starting block & length of file4
5
4->1
5->1
6->1
7->1
8->1
 the file is allocated to disk
 if u want to enter more files?(y-1/n-0)1
Enter the starting block & length of file5
2
Block already allocated
 if u want to enter more files?(y-1/n-0)1
Enter the starting block & length of file9
10
9->1
10->1
11->1
12->1
13->1
14->1
15->1
16->1
17->1
18->1
 the file is allocated to disk
 if u want to enter more files?(y-1/n-0)0
```

**RESULT:**

Thus the sequential allocation program is executed successfully.

**B) INDEXED ALLOCATION**

**AIM:**

To implement file allocation strategies using indexed allocation

**PROGRAM:**

```
#include<stdio.h>
int f[50],i,k,j,inde[50],n,c,count=0,p;
int main()
{
for(i=0;i<50;i++)
f[i]=0;
x:
printf("enter index block\t");
scanf("%d",&p);
if(f[p]==0)
{
f[p]=1;
printf("enter no of files on index\t");
scanf("%d",&n);
}
else
{
printf("Block already allocated\n");
goto x;
}
for(i=0;i<n;i++)
scanf("%d",&inde[i]);
for(i=0;i<n;i++)
if(f[inde[i]]==1)
{
printf("Block already allocated");
goto x;
}
for(j=0;j<n;j++)
```

```
f[inde[j]]=1;
printf("\n allocated");
printf("\n file indexed");
for(k=0;k<n;k++)
printf("\n %d->%d:%d",p,inde[k],f[inde[k]]);
printf(" Enter 1 to enter more files and 0 to exit\t");
scanf("%d",&c);
if(c==1)
goto x;
else
exit(0);
}
```

**OUTPUT**

```
enter index block        9
enter no of files on index        3
1
2
3
 allocated
 file indexed
 9->1:1
 9->2:1
 9->3:1 Enter 1 to enter more files and 0 to exit        1
enter index block        10
enter no of files on index        6
2
3
4
5
6
7
Block already allocatedenter index block       2
Block already allocated
enter index block        10
```

Block already allocated

enter index block 11

enter no of files on index 2

1

2

Block already allocatedenter index block 11

Block already allocated

enter index block 14

enter no of files on index 2

45

34

allocated

file indexed

14->45:1

14->34:1 Enter 1 to enter more files and 0 to exit

**RESULT:**

Thus the indexed file allocation program is executed successfully.