List of experiments :

S.No	Name of the Experiment
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
2.	Multiprogramming-Memory management- Implementation of fork (), wait (), exec() and exit (), System calls
3.	Write a C program that makes a copy of a file using standard I/O, and system calls
4.	Write a C program to emulate the UNIX ls –l command
5.	Simulate the following CPU scheduling algorithms a) Round Robin b) SJF c) FCFS d) Priority
6.	Simulate the following a) Multiprogramming with a fixed number of tasks (MFT) b) Multiprogramming with a variable number of tasks (MVT)
7.	Write a C program that illustrates how to execute two commands concurrently with a command pipe. Ex: - ls –l | sort
8.	Simulate Bankers Algorithm for Dead Lock Avoidance
9..	Simulate Bankers Algorithm for Dead Lock Prevention.
10.	Write a C program that illustrates two processes communicating using shared memory
11.	Simulate the following page replacement algorithms. a) FIFO b) LRU c) LFU

12.	Write a C program to simulate producer and consumer problem using semaphores
13.	Simulate the following File allocation strategies a) Sequenced b) Indexed c) Linked .
14.	Write C program to create a thread using pthreads library and let it run its function
15.	Write a C program to illustrate concurrent execution of threads using pthreads library.

EXERCISE-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.

(i)man
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

(ii)who:
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.
Synopsys
who [OPTION]... [FILE | ARG1 ARG2]
Example
who am i

(iii) cat:
concatenate or display files
Synopsys
cat [- q] [- s] [- S] [- u] [- n[- b]] [- v [- [- t]] [- | File ...]
The cat command reads each File parameter in sequence and writes it to standard output. If you do not specify a file name, the cat command reads from standard input. You can also specify a file name of – (minus) for standard input.
Exit Status
This command returns the following exit values:
0 All input files were output successfully.
>0 An error occurred.
Examples
1. To display a file at the workstation, enter:
cat notes
2. To concatenate several files, enter:
cat section1.1 section1.2 section1.3 >section1
3. To suppress error messages about files that do not exist, enter:

cat -q section2.1 section2.2 section2.3 >section2
4. To append one file to the end of another, enter:

cat section1.4 >>section1
The >> appends a copy of section1.4 to the end of section1. If
you want to replace the file, use the >.
5. To add text to the end of a file, enter:
cat >>notes
Get milk on the way home
Ctrl-D
6. To concatenate several files with text entered from the keyboard,
cat section3.1 - section3.3 >section3
7. To concatenate several files with output from another command,
li | cat section4.1 - >section4

(iv) cd:
The cd command, which stands for "change directory", changes the shell's current working directory.
Syntax
cd directory
Example
cd new

(v) cp:

Copy files and directories.
Syantax:
cp [option] source destination
Example:
cp file1 file2

(vi) ps:
Report a snapshot of the current processes.
Syatax:
ps [options]

(vii) ls
List directory contents
Syntax:
ls [option]

(viii) mv
Move or rename files
Syantax
mv [option] soure dest

Example:
mv file1 file2

(ix) rm
Remove files or directories
Syantax:
rm [option] filename
Example
rm file1
cat file1
cat: file1:No such file or directory.

(x) mkdir
Make directories
Syantax
mkdir [option] directoryname
Example:
mkdir cse
cd cse
~/cse$_

(xi) rmdir
Remove empty directories
Syntax:
rmdir [option] directoryname
Example:
rmdir cse
cd cse
bash:cd:cse: No such file or directory

(xii) echo
Display a line of text
Syntax:
echo [short_option] String
Example
echo "This is CSE World"
This is CSE World

(xiii) more
File perusal filter or crt viewing
Syntax:
more [-difpcsu] [-num] [+/pattern] [+linenumber]
Example
cat file1

BVCITS
This is CSE class
more +2 file1
This is CSE class

(xiv) date
Print or set the system date or time.
Syntax:
date [option] [format]
Example
date
Mon Jan 1 00:30:20 PST 2001

(xv) time
Run programs and summarize system resource usage.
Syntax:
time [format] [file]

(xvi) kill
Send a signal to a process
Syntax:
kill [-signal | -s signal] pid…
Example
Kill -1

(xvii) history
GNU history library.
Example:
history
vi 4ex.sh
sh 4b.sh
man cat…

(xviii) chmod
Change file mode
Syntax:
chmod [option] mode file
Example
Chmod 652 file1
-rw-r-x-w- bvcits bvcits 21 2001-01-01 00:37 file1

(xix) chown
Change the file owner and group
Syntax:
chown [option] [owner] [:[group]] file

Example

```
chown cse bvc
```

(xx) finger

The finger display information about the system user

Syntax:

```
Finger [-l] [-m] [-p] [-s] [username]
```

Example

```
finger abc
login:abc    name:(null)
directory: /home/abc  shell: /bin/bash
on since Mon Nov 1 18:45 (IST)
```

(xxi) pwd

Print name of the current working directory

Syntax:

```
pwd [option]
```

Example

```
pwd
/home/bvcits
```

(xxii)cal

Cal, ncal- displays a calendar and the date of easter

Syntax:

```
cal [-a number] [-b numer] [[mont] year]
```

Example

```
June 2018
Su      Mo      Tu      We      Th      Fr      Sa
                1       2       3       4       5
6       7       8       9       10      11      12
13      14      15      16      17      18      19
20      21      22      23      24      25      26
27      28      29      30      31
```

(xxiii) logout

Login,logout-write utmp and wtmp entries

Syntax:

```
#include<utmp.h>
Void login(const struct utmp *ut);
Int logout(const char *utline);
Link with –lutil
```

(xxiv) shutdown
Bring the system down
Syntax:
Shutdown [option]   time[message]

b) Study of vi editor.
How to Use the 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 within 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>.

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.

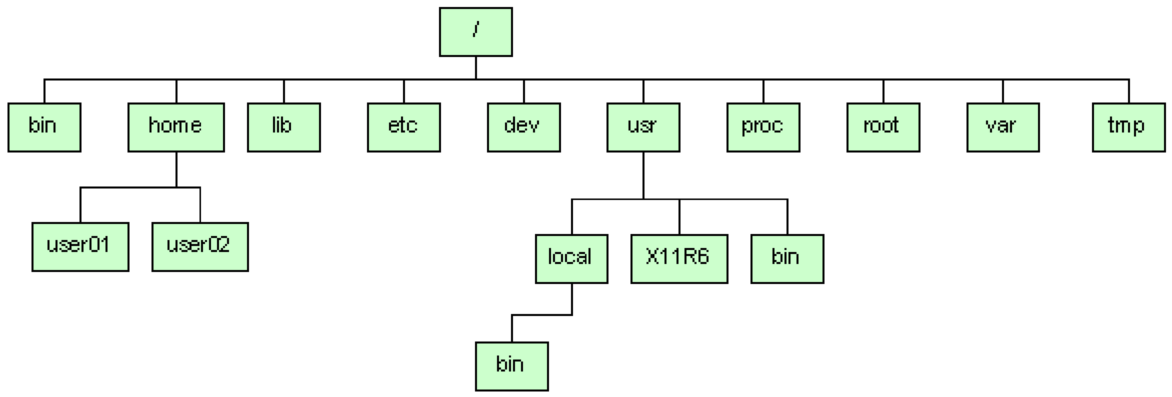
The Bash shell
The popularity of sh motivated programmers to develop a shell that was compatible with it, but with several enhancements. Linux systems still offer the sh shell, but "bash" -- the "Bourne-again Shell," based on sh -- has become the new default standard. One attractive feature of bash is 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.

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 filesystem 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 −

Directory	**Description**
/	This is the root directory which should contain only the directories needed at the top level of the file structure.
/bin	This is where the executable files are located. They are available to all user.
/dev	These are device drivers.
/etc	Supervisor directory commands, configuration files, disk configuration files, valid user lists, groups, ethernet, hosts, where to send critical messages
/lib	Contains shared library files and sometimes other kernel related files.
/boot	Contains files for booting the system.
/home	Contains the home directory for users and other accounts.
/mnt	Used to mount other temporary file systems, such as cdrom and floppy for the CDROM drive and floppy diskette drive, respectively
/proc	Contains all processes marked as a file by process number or other information that is dynamic to the system.
/tmp	Holds temporary files used between system boots
/usr	Used for miscellaneous purposes, or can be used by many users. Includes administrative commands, shared files, library files, and others
/var	Typically contains variable length files such as log and print files and any other type of file that may contain a variable amount of data
/sbin	Contains binary (executable) files, usually for system administration. For example *fdisk* and *ifconfig* utlities.
/kernel	Contains kernel files

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.
PATH	Indicates search path for commands. It is a colon
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
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.

EXERCISE-2

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

Program:

```
#include<fcntl.h>
#include<unistd.h>
int main(int argc, char **argv)
{
        int n,size,fd1,fd2;
        char c;
        fd1 = open(argv[1],O_RDONLY);
        fd2 = open(argv[2],O_WRONLY);
        size = lseek(fd1, -1, SEEK_END);
        n = lseek(fd1, 0, SEEK_SET);
        while(n++ < size)
        {
            read(fd1, &c,1);
            write(fd2, &c,1);
        }
}
```

Output:

```
$cat >file1
First UNIX Program
Welcome to Unix
$cat >file2
$cc exe1.c
$./a.out file1 file2
$ cat file2
First UNIX Program
Welcome to Unix
```

EXERCISE-3

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

Program:

```
#include <unistd.h>
#include <stdio.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <stdlib.h>
int main()
{
	int pid;
	pid = fork();
	if(pid<0)
	{
		printf("\n Child process creation failed");
		exit(-1);
	}
	else if(pid == 0)
	{
		execlp("/bin/ls", "ls", '-l", NULL);
	}
	else
	{
		wait(NULL);
		printf("\n child process completed");
		exit(0);
	}
}
```

Output:

```
$cc exe3.c
$./a.out
Total 160
-rw-r—r—1 bvcits bvcits 385 2001-01-01 01:36 155a1
drwxr-xrx1 bvcits bvcits 4096 2001-01-01 00:51 csea
-rw-r—r—1 bvcits bvcits 335 2001-01-01 00:12 exe2.c
-rw-r—r—1 bvcits bvcits 397 2001-01-01 00:23 exe3.c
```

EXERCISE-4

4. Write a C program that illustrates how to execute two commands concurrently with a command pipe.

Program:

```
#include<stdio.h>
#include<stdlib.h>
int main(int argc,char *argv[])
{
int fd[2],pid,k;

k=pipe(fd);
if(k==-1)
{
perror("pipe");
exit(1);
}
pid=fork();
if(pid==0)
{
close(fd[0]);
dup2(fd[1],1);
close(fd[1]);
execlp(argv[1],argv[1],NULL);
perror("execl");
}
else
{
wait(2);
close(fd[1]);
dup2(fd[0],0);
close(fd[0]);
execlp(argv[2],argv[2],NULL);
perror("execl");
}
}
```

Output:

```
$cc exe4.c
$./a.out ls sort
a.out
ls_sort.c
named.
pip
```

EXERCISE-5

Simulate the following CPU scheduling algorithms

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

AIM: Simulate the following CPU scheduling algorithms

a) ROUND ROBIN:

DESCRIPTION:

- Round Robin is the preemptive process scheduling algorithm.
- Each process is provided a fix time to execute, it is called a **quantum**.
- Once a process is executed for a given time period, it is preempted and other process executes for a giventime period.
- Context switching is used to save states of preempted processes.

PROGRAM:

/* C Program to implement Round Robin CPU Scheduling Algorithm */

```
#include<stdio
 .h>int main()
{
  int count,j,n,time,remain,flag=0,time_quantum;
  int
  wait_time=0,turnaround_time=0,at[10],bt[10],rt[
  10];printf("Enter Total Process:\t ");
  scanf("%d",&
  n);remain=n;
  for(count=0;count<n;count++)
  {
    printf("Enter Arrival Time and Burst Time for Process Process Number %d :",count+1);
    scanf("%d",&at[count]);
    scanf("%d",&bt[count]);
    rt[count]=bt[count];
  }
  printf("Enter Time Quantum:\t");
  scanf("%d",&time_quantum);
  printf("\n\nProcess\t|Turnaround Time|Waiting
  Time\n\n");for(time=0,count=0;remain!=0;)
  {
    if(rt[count]<=time_quantum && rt[count]>0)

    {
     time+=rt[cou
     nt];
     rt[count]=0;
     flag=1;
    }

    else if(rt[count]>0)
    {
     rt[count]-=time_quantum;
     time+=time_quantum;
    }
    if(rt[count]==0 && flag==1)
    {
     remain--;
     printf("P[%d]\t|\t%d\t|\t%d\n",count+1,time-at[count],time-
     at[count]-bt[count]); wait_time+=time-at[count]-bt[count];
     turnaround_time+=time-
     at[count];flag=0;
    }
    if(count==n
     -1)
     count=0;
    else
     if(at[count+1]<=tim
     e)count++;
    else
     count=0;
  }
  printf("\nAverage Waiting Time=
  %f\n",wait_time*1.0/n); printf("Avg Turnaround
  Time = %f",turnaround_time*1.0/n);
  return 0;
}
```

OUTPUT:

```
Enter Total Process: 4

Enter Arrival Time and Burst Time for Process
Process Number  1 : 09

Enter Arrival Time and Burst Time for Process
Process Number  2 : 15

Enter Arrival Time and Burst Time for Process
Process Number 3 : 23

Enter Arrival Time and Burst Time for Process
Process Number  4 : 34

Enter Time Quantum:   5


Proce    Turnaround    Waiting
ss          Time       Time

P[2]         9            4

P[3]         11           8

P[4]         14           10
P[1]         21          12

Average Waiting Time=
8.500000Avg Turnaround
Time = 13.70000
```

b) SJF:

DESCRIPTION:

- This is also known as **shortest job first**, or SJF
- This is a non-preemptive, pre-emptive scheduling algorithm.
- Best approach to minimize waiting time.
- Easy to implement in Batch systems where required CPU time is known in advance.
- Impossible to implement in interactive systems where required CPU time is not known.

- The processer should know in advance how much time process will take.

PROGRAM:

/* C Program to implement SJF CPU Scheduling Algorithm */

```
#include<stdio
.h>void main()
{

int
bt[20],p[20],wt[20],tat[20],i,j,n,total=0,pos,t
emp;float avg_wt,avg_tat;
printf("Enter number of
process:");scanf("%d",&n);
printf("\nEnter Burst
Time:\n");for(i=0;i<n;i++)
{

printf("p%d:",i+1);

scanf("%d",&bt
[i]);p[i]=i+1;
}

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

pos=i;
for(j=i+1;j<n;j++)
{

if(bt[j]<bt[pos])
pos=j;
}

temp=bt[i];
bt[i]=bt[po
s];
bt[pos]=te
mp;
temp=p[i];
p[i]=p[pos]
;
p[pos]=tem
p;
}
wt[0]=
0;
for(i=1;i<n;i++)

{

wt[i]=0;
for(j=0;j<i;j
++)
wt[i]+=bt[j];
total+=wt[i];
}

avg_wt=(float)total
/n;total=0;
printf("\nProcess\t          Burst Time    \tWaiting
Time\tTurnaround Time");for(i=0;i<n;i++)
{

tat[i]=bt[i]+wt
[i];
total+=tat[i];
printf("\np%d\t\t  %d\t\t      %d\t\t\t%d",p[i],bt[i],wt[i],tat[i]);
avg_tat=(float)total/n;
printf("\n\nAverage Waiting
Time=%f",avg_wt); printf("\nAverage
Turnaround Time=%f\n",avg_tat);
}
```

OUTPUT:

```
Enter number of
process: 4Enter Burst
Time:

P1:4
```

```
P2:8

P3:3

P4:7

Process    Burst Time    Waiting        Turnaround
                         Time           Time

P3            3            0               3

P1            4            3               7

P4            7            7              14

P2            8           14              22

Average Waiting
Time=6.000000 Average
Turnaround Time=11.500000
```

c) FCFS:

DESCRIPTION:

- Jobs are executed on first come, first serve basis.
- It is a non-preemptive, pre-emptive scheduling algorithm.
- Easy to understand and implement.
- Its implementation is based on FIFO queue.
- Poor in performance as average wait time is high.

PROGRAM:

/* C Program to implement FCFS CPU Scheduling Algorithm */

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

        int n,a[10],b[10],t[10],w[10],g[10],i,m;
```

```
            float
att=0,awt=0;
for(i=0;i<10;i++)
{

a[i]=0; b[i]=0; w[i]=0; g[i]=0;

}

            printf("enter the number of
process");scanf("%d",&n);
            printf("enter the burst
times");for(i=0;i<n;i++)
scanf("%d",&b[i]);
printf("\nenter the arrival
times");for(i=0;i<n;i++)
scanf("%d",&a[i]);
            g[0]=0;

for(i=0;i<10;i+
+)
g[i+1]=g[i]+b
[i];
for(i=0;i<n;i+
+)
{

            w[i]=g[i]-a[i];

             t[i]=g[i+1]-
             a[i];
             awt=awt+w[
             i];
             att=att+t[i];

}

awt
=awt/n;
att=att/n;
printf("\n\tprocess\twaiting time\tturn arround
time\n");for(i=0;i<n;i++)
{
```

```
printf("\tp%d\t\t%d\t\t%d\n",i,w[i],t[i]);

}

        printf("the average waiting time is
        %f\n",awt); printf("the average turn
        around time is %f\n",att);
}
```

OUTPUT:

```
enter the number of
process 4enter the burst
times
4

9

 8

3

enter the arrival times

0

2

4

3

process    waiting       turn arround
           time          time

p0            0             4

p1            2            11
p2            9            17

p3           18            21

the average waiting time is 7.250000

the average turn around time is 13.250000
```

d)PRIORITY:

DESCRIPTION:

- Priority scheduling is a non-preemptive algorithm and one of the most common scheduling algorithms inbatch systems.
- Each process is assigned a priority. Process with highest priority is to be executed first and so on.
- Processes with same priority are executed on first come first served basis.
- Priority can be decided based on memory requirements, time requirements or any other resource requirement.

PROGRAM:

/* C Program to implement Priority CPU Scheduling Algorithm */

```
int main()

{

int
bt[20],p[20],wt[20],tat[20],pr[20],i,j,n,total=0,pos,temp,avg_wt,avg_t
at;printf("Enter Total Number of Process:");
scanf("%d",&n);



printf("\nEnter Burst Time and
Priority\n");for(i=0;i<n;i++)
{

printf("\nP[%d]\n",i
+1);printf("Burst
Time:");
scanf("%d",&bt[i]);

printf("Priority:");
scanf("%d",&pr[i])
;p[i]=i+1;
}



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

{

pos=i;
for(j=i+1;j<n;j++)
{

if(pr[j]<pr[po
s])pos=j;
}



temp=pr[i];
pr[i]=pr[po
s];
pr[pos]=te
mp;


temp=bt[i];
bt[i]=bt[po
s];
bt[pos]=te
mp;


temp=p[i];
p[i]=p[pos
];
p[pos]=te
mp;
}
wt[0]=
0;
for(i=1;i<n;i++)
```

```
{

wt[i]=0;
for(j=0;j<i;j
++)
wt[i]+=bt[j];


total+=wt[i];

}

avg_wt=total
 /n;total=0;
printf("\nProcess\t             Burst Time     \tWaiting
Time\tTurnaround Time");for(i=0;i<n;i++)
{

tat[i]=bt[i]+wt
[i];
total+=tat[i];
printf("\nP[%d]\t\t  %d\t\t    %d\t\t\t%d",p[i],bt[i],wt[i],tat[i]);

}

avg_tat=total/n;

printf("\n\nAverage Waiting
Time=%d",avg_wt); printf("\nAverage
Turnaround Time=%d\n",avg_tat);return 0;
```

OUTPUT:

```
Enter Total Number of
Process: 4Enter Burst Time
and Priority:


P[1]

Burst
```

```
Time: 6
Priority:3
P[2]

Burst
Time: 2
Priority:2


P[3]

Burst Time:
14Priority:1


P[4]

Burst
Time: 6
Priority:4


Process       Burst Time      Waiting Time      Turnaround
                                                Time
P[3]             14                0               14
P[2]              2               14               16
P[1]              6               16               22
P[4]              6               22               28

Average Waiting
Time=13 Average
Turnaround Time=20
```

EXERCISE-6

AIM: Multiprogramming Memory management Implementation of fork(),wait(),exec() and exit(),system calls

DESCRIPTION:

FORK():

Fork system call use for creates a new process, which is called ***child process***, which runs concurrently with process(which process called system call fork) and this process is called ***parent process***. After a new child process created,both processes will execute the next instruction following the fork() system call. A child process uses the same pc(program counter), same CPU registers, same open files which use in the parent process.

EXEC():

The exec family of functions replaces the current running process with a new process. It can be used to run a C programby using another C program. It comes under the header file **unistd.h.** There are many members in the exec family which are shown below with examples.

- **execvp** : Using this command, the created child process does not have to run the same program as the parentprocess does. The **exec** type system calls allow a process to run any program files, which include a binary executable or a shell script .

Syntax:

```
int execvp (const char *file, char *const argv[]);
```

file: points to the file name associated with the file being executed.

argv: is a null terminated array of character pointers.

- **execv** : This is very similar to execvp() function in terms of syntax as well. The syntax of **execv()** is as shownbelow:

Syntax:

```
int execv(const char *path, char *const argv[]);
```

path: should point to the path of the file being executed.

argv[]: is a null terminated array of character pointers.

- **execlp and execl** : These two also serve the same purpose but the syntax of of them are a bit different which isas shown below:**Syntax:**
- **execvpe and execle** : These two also serve the same purpose but the syntax of them are a

```
int execlp(const char *file, const char *arg,.../* (char  *) NULL */);

int execl(const char *path, const char *arg,.../* (char  *) NULL */);
```

bit different from allthe above members of exec family. The synatxes of both of them are shown below :

Syntax:

```
int execvpe(const char *file, char *const argv[],char *const envp[]);
```

Syntax:

```
int execle(const char *path, const char *arg, .../*, (char *) NULL,

char * const envp[] */);
```

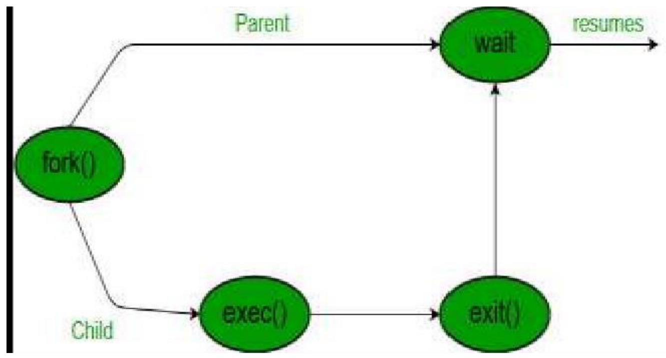
WAIT() :

A call to wait() blocks the calling process until one of its child processes exits or a signal is received. After childprocess terminates, parent ***continues*** its execution after wait system call instruction.
Child process may terminate due to any of these:

- It calls exit();
- It returns (an int) from main
- It receives a signal (from the OS or another process) whose default action is to terminate.

<u>EXIT()</u>:

It terminates the calling process without executing the rest code which is after the exit() function.



FORK():

PROGRAM:

/* C Program to implement Fork() system calls */

```
#include <stdio.h>
#include
<sys/types.h>
#include
<unistd.h> void
forkexample()
{

  if (fork() == 0)

    printf("Hello from Child!\n");



  else

    printf("Hello from Parent!\n");

}
```

```
int main()

{

   forkexample
   ();return 0;
}
```

OUTPUT:

```
Hello from Child!

Hello from Parent!
```

EXEC():

/* C Program to implement Exec() system calls */

PROGRAM:

```
#include <stdio.h>
#include
<sys/types.h>
#include
<unistd.h>
#include
<stdlib.h>
#include <errno.h>
#include
<sys/wait.h> int
main()
{

  pid_t   pid;
  int ret = 1;
  int  status;
  pid =
  fork(); if
  (pid == -1)
{



        printf("can't fork, error
        occured\n");
        exit(EXIT_FAILURE);
  }

  else if (pid == 0)
{

         printf("child process, pid =
         %u\n",getpid());
         execv("ls",argv_list);
         exit(0);


  }

  else{

         printf("parent process, pid =
           %u\n",getppid());if (waitpid(pid,
           &status, 0) > 0)
           {

                 if (WIFEXITED(status) &&
                  !WEXITSTATUS(status))printf("program
                  execution successfull\n");
                    else if (WIFEXITED(status) && WEXITSTATUS(status))

                  {

                        if (WEXITSTATUS(status) == 127)

                        {

                              printf("execv failed\n");

                         }

                        else
                    printf("program terminated
                   normally,"" but returned a non-
                   zero status\n");
                }

                else

                printf("program didn't terminate normally\n");

            }

            else

           {

                printf("waitpid() failed\n");

            }

            exit(0);

        }
    return 0;

  }
```

OUTPUT:

```
parent process, pid = 11523

child process, pid = 14188

Program execution successfull
```

WAIT() :

PROGRAM:

/* C Program to implement Wait() system calls */

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


int main()

{

   pid_t cpid;

   if (fork()== 0)

      exit(0
   );else
      cpid = wait(NULL);

   printf("Parent pid = %d\n",
   getpid());printf("Child pid =
   %d\n", cpid);

   return 0;

}
```

EXIT():

PROGRAM:

/* C Program to implement Exit() system calls */

```
#include
<stdio.h>
#include
<stdlib.h>int
main(void)
{

   printf("STAR
     T");exit(0);
   printf("End of program");

}
```

OUTPUT:

```
START
```

EXERCISE-7

AIM: Simulate the following

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

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

DESCRIPTION:

MFT : Multiprogramming with a Fixed number of Tasks is one of the old memory management techniques in whichthe memory is partitioned into fixed size partitions and each job is assigned to a partition. The memory assigned to a partition does not change.
MVT : Multiprogramming with a Variable number of Tasks is the memory management technique in which each job gets just the amount of memory it needs. That is, the partitioning of memory is dynamic and changes as jobs enter and leave the system. MVT is a more ``efficient'' user of resources. MFT suffers with the problem of internal fragmentationand MVT suffers with external fragmentation.

PROGRAM

/* C Program to implement Multiprogramming with a fixed number of tasks (MFT)*/

```
#include<stdio.
h>
#include<conio
.h>main()
{

int ms, bs, nob, ef,n,
mp[10],tif=0;int i,p=0;
clrscr();

printf("Enter the total memory available (in
Bytes) -- ");scanf("%d",&ms);
printf("Enter the block size (in
Bytes) -- ");scanf("%d", &bs);
nob=ms/bs;
ef=ms -
nob*bs;
printf("\nEnter the number of
processes -- ");scanf("%d",&n);
for(i=0;i<n;i++)
```

```
{

printf("Enter memory required for process %d (in
Bytes)-- ",i+1);scanf("%d",&mp[i]);
}

printf("\nNo. of Blocks available in memory -- %d",nob);

printf("\n\nPROCESS\tMEMORY REQUIRED\t ALLOCATED\tINTERNAL
FRAGMENTATION");

for(i=0;i<n && p<nob;i++)

{

printf("\n
%d\t\t%d",i+1,mp[i]);
if(mp[i] > bs)
printf("\t\tNO\t\t---
");else
{

printf("\t\tYES\t%d",bs-
mp[i]); tif = tif + bs-
mp[i];
p++;

}

}

if(i<n)

printf("\nMemory is Full, Remaining Processes cannot be accomodated");
printf("\n\nTotal Internal Fragmentation is %d",tif);
printf("\nTotal External Fragmentation is
%d",ef);getch();
```

OUTPUT:

```
Enter the total memory available (in
Bytes) -- 1000Enter the block size (in
Bytes)-- 300
Enter the number of processes – 5
Enter memory required for process 1 (in
Bytes) -- 275 Enter memory required for
process 2 (in Bytes) -- 400 Enter memory
required for process 3 (in Bytes) -- 290 Enter
memory required for process 4 (in Bytes) --
293 Enter memory required for process 5 (in
Bytes) -- 100 No. of Blocks available in
memory -- 3
PROCESS MEMORY-REQUIRED ALLOCATED INTERNAL-FRAGMENTATION

1          27          YE          25
           5           S
2          40          NO          -----
           0
3          29          YE          10
           0           S
4          29          YE           7
           3           S
Memory is Full, Remaining Processes cannot be
accommodated Total Internal Fragmentation is 42
Total External Fragmentation is 100
```

MVT :

PROGRAM:

/* C Program to implement Multiprogramming with a variable number of tasks (MVT)*/

```
#include<stdio.h>

#include<conio
.h>main()
{

int ms,mp[10],i,
temp,n=0;char ch =
'y';
clrscr();

printf("\nEnter the total memory available (in
Bytes)-- ");scanf("%d",&ms);
temp=ms;
for(i=0;ch=='y';i++,n
++)
{

printf("\nEnter memory required for process %d (in
Bytes) -- ",i+1);scanf("%d",&mp[i]);
if(mp[i]<=temp)

{

printf("\nMemory is allocated for Process
%d ",i+1);temp = temp - mp[i];
}

else

{

printf("\nMemory is
Full");break;
}

printf("\nDo you want to
continue(y/n) -- ");scanf(" %c",
&ch);
}

printf("\n\nTotal Memory Available -- %d",
ms); printf("\n\n\tPROCESS\t\t MEMORY
ALLOCATED ");for(i=0;i<n;i++)
printf("\n \t%d\t\t%d",i+1,mp[i]);
printf("\n\nTotal Memory Allocated is
%d",ms-temp);printf("\nTotal External
Fragmentation is %d",temp); getch();
}
```

OUTPUT

```
Enter the total memory available (in Bytes) -
- 1000 Enter memory required for process 1
(in Bytes) -- 400Memory is allocated for
```

```
Process 1
Do you want to continue(y/n) -- y

Enter memory required for process 2 (in
Bytes) -- 275Memory is allocated for
Process 2
Do you want to continue(y/n) -- y

Enter memory required for process 3 (in
Bytes) -- 550Memory is Full
Total Memory Available --
1000 PROCESS MEMORY-
ALLOCATED1400
2                  275

Total Memory Allocated is 675



Total External Fragmentation is 325
```

EXERCISE-8

AIM: Simulate the Banker's algorithm for Dead Lock Avoidance

DESCRIPTION:

The banker's algorithm is a resource allocation and deadlock avoidance algorithm that tests for safety by simulating theallocation for predetermined maximum possible amounts of all resources, then makes an "s-state" check to test for possible activities, before deciding whether allocation should be allowed to continue.
Following **Data structures** are used to implement the Banker's Algorithm:

Let **'n'** be the number of processes in the system and **'m'** be the number of resources types.

Available :

- It is a 1-d array of size **'m'** indicating the number of available resources of each type.

- Available[j] = k means there are **'k'** instances of
resource type **R_jMax :**
- It is a 2-d array of size '**n*m**' that defines the maximum demand of each process in a system.

- Max[i, j] = k means process **P_i** may request at most **'k'** instances of resource type **R_j.**

Allocation :

- It is a 2-d array of size **'n*m'** that defines the number of resources of each type currently allocated to eachprocess.
- Allocation[i, j] = k means process **P_i** is currently allocated **'k'** instances of
resource type **R_jNeed :**
- It is a 2-d array of size **'n*m'** that indicates the remaining resource need of each process.

- Need [i, j] = k means process **P_i** currently allocated **'k'** instances of resource type **R_j**
- Need [i, j] = Max [i, j] – Allocation [i, j]

PROGRAM:

```
/* C Program to implement Dead Lock Avoidance  using  Banker's algorithm */
#include <stdio.h>

  #include <stdlib.h>

  int main()

  {
```

```
int Max[10][10], need[10][10], alloc[10][10], avail[10], completed[10], safeSequence[10];

int p, r, i, j, process, count;

count = 0;


printf("Enter the no of processes : ");

scanf("%d", &p);

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

   completed[i] = 0;


printf("\n\nEnter the no of resources : ");

scanf("%d", &r);


printf("\n\nEnter the Max Matrix for each process : ");

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

{

   printf("\nFor process %d : ", i + 1);

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

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

}


printf("\n\nEnter the allocation for each process : ");

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

{

   printf("\nFor process %d : ",i + 1);
   for(j = 0; j < r; j++)
```

```
        scanf("%d", &alloc[i][j]);
    }


    printf("\n\nEnter the Available Resources : ");
    for(i = 0; i < r; i++)
      scanf("%d", &avail[i]);


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


      for(j = 0; j < r; j++)
        need[i][j] = Max[i][j] - alloc[i][j];


      do
      {
        printf("\n Max matrix:\tAllocation matrix:\n");


        for(i = 0; i < p; i++)
        {
          for( j = 0; j < r; j++)
            printf("%d ", Max[i][j]);
          printf("\t\t");
          for( j = 0; j < r; j++)
            printf("%d ", alloc[i][j]);
          printf("\n");
        }
```

```
process = -1;


for(i = 0; i < p; i++)
{

   if(completed[i] == 0)//if not completed

   {

      process = i ;

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

      {

         if(avail[j] < need[i][j])

         {

            process = -1;

            break;

         }

      }

   }

   if(process != -1)

      break;

}


if(process != -1)

{

   printf("\nProcess %d runs to completion!", process + 1);

   safeSequence[count] = process + 1;

   count++;

   for(j = 0; j < r; j++)
```

```
            {

               avail[j] += alloc[process][j];

               alloc[process][j] = 0;

               Max[process][j] = 0;

               completed[process] = 1;

            }

         }
      }

      while(count != p && process != -1);


      if(count == p)

      {

         printf("\nThe system is in a safe state!!\n");

         printf("Safe Sequence : < ");

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

            printf("%d ", safeSequence[i]);

         printf(">\n");

      }

      else

         printf("\nThe system is in an unsafe state!!");


}
```

OUTPUT:

```
Enter the no of processes : 5
Enter the no of resources : 3


Enter the Max Matrix for each process :

For process 1 : 7

5

3

For process 2 : 3

2

2

For process 3 : 7
0

2

For process 4 : 2

2

2

For process 5 : 4

3

3

Enter the allocation for each process :

For process 1 : 0

1

0

For process 2 : 2

0

0
```

```
For process 3 : 3

0

2

For process 4 : 2

1

1

For process 5 : 0

0

2

Enter the Available Resources : 3

3
Process 3 runs to completion!
2



 Max matrix:Allocation matrix:

7 5 3       0 1 0

3 2 2       2 0 0

7 0 2       3 0 2

2 2 2       2 1 1

4 3 3       0 0 2

Process 2 runs to
 completion! Max matrix:
                 Allocation
 matrix:
7 5 3       0 1 0

0 0 0       0 0 0

7 0 2       3 0 2

2 2 2       2 1 1

4 3 3       0 0 2
```

```
Max matrix:     Allocation matrix:

7 5 3       0 1 0
0 0 0       0 0 0
0 0 0       0 0 0
2 2 2       2 1 1
                  ystem is in
4 3 3       0 0 2
a safe state!!
```

```
7 5 3       0 1 0
0 0 0       0 0 0
0 0 0       0 0 0
0 0 0       0 0 0
4 3 3       0 0 2

Process 1 runs
to
```

```
0 0 0       0 0 0
0 0 0       0 0 0
0 0 0       0 0 0
0 0 0       0 0 0
4 3 3       0 0 2
```

EXERCISE-9

AIM: Simulate the Banker’s algorithm for Dead Lock Prevention

DESCRIPTION: We can prevent Deadlock by eliminating any of the above four condition.

- **Eliminate Mutual Exclusion :** It is not possible to dis-satisfy the mutual exclusion because some resources,such as the tap drive and printer, are inherently non-shareable.
- **Eliminate Hold and wait**: Allocate all required resources to the process before start of its execution, this wayhold and wait condition is eliminated but it will lead to low device utilization.
- **Eliminate No Preemption**: Preempt resources from process when resources required by other high priorityprocess.
- **Eliminate Circular Wait:** Each resource will be assigned with a numerical number. A process can request forthe resources only in increasing order of numbering.

PROGRAM:

/* C Program to implement Dead Lock Prevention using Banker’s algorithm */

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

{

  int
  max[10][10],a1[10][10],av[10],i,j,k,m,n,ne[10][10],flag=
  0;printf("\nEnter the matrix dimensions:");
  scanf("%d%d",&m,&n);

  printf("\n Enter the maximum
  matrix:\n");for(i=0;i<m;i++)
```

```
{
  for(j=0;j<n;j++)
  {
    scanf("%d",&max[i][j]);
  }
}
printf("\n Enter allocated
matrix:\n");for(i=0;i<m;i++)
{
  for(j=0;j<n;j++)
  {
    scanf("%d",&a1[i][j]);
  }
}
printf("\n The need
matrix:\n");for(i=0;i<m;i++)
{
  for(j=0;j<n;j++)
  {
      ne[i][j]=max[i][j]-a1[i][j];

      printf("\t%d",ne[i][j]);
    }

   printf("\n");

  }

 printf("\n Enter available
 matrix:\n");for(i=0;i<n;i++)
   scanf("%d",&av[i]);

 printf("\n Maximum
 matrix\n");for(i=0;i<m;i++)
  {

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

    {

      printf("\t%d",max[i][j]);

    }

   printf("\n");

  }

 printf("\n Allocated
 matrix:\n");for(i=0;i<m;i++)
  {
    for(j=0;j<n;j++)

    {

      printf("\t%d",a1[i][j]);
    }

   printf("\n");

  }

 printf("\n Available
 matrix:\n");for(i=0;i<n;i++)
 {

   printf("%d\t",av[i]);

 }

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

  {

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

   {

      if(av[i]>=ne[i][j])
         flag=1;
       else

         flag=0;

    }
  }

if(flag==0)

   printf("\n Unsafe
  state");else
   printf("\n safe state");

}
```

OUTPUT:

```
Enter the matrix
dimensions:3 3Enter the
maximum matrix:
3 6 8

4 3 3

3 4 4

Enter allocated
matrix:2 2 3
2 0 3

1 2 4

The need
matrix:1 4 5
2 3 0

2 2 0
```

```
Enter available
matix:2 3 0

Maximum
matrix:


3 6 8


4 3 3

3 4 4


Allocated
matrix:2 2 3

2 0 3


1 2 4


Available
matrix:2 3 0

safe state
```

EXERCISE-10

AIM: Simulate the following Page Replacement algorithms

a) FIFO b)LRU c)LFU

DESCRIPTION:

a) FIFO :

This is the simplest page replacement algorithm. In this algorithm, operating system keeps track of all pagesin the memory in a queue, oldest page is in the front of the queue. When a page needs to be replaced page in the frontof the queue is selected for removal.

PROGRAM:

/* C Program to implement FIFO Page Replacement algorithms */

```
#include<stdio.h>int main()
{

    int
  i,j,n,a[50],frame[10],no,k,avail,count=0;
  printf("\n ENTER THE NUMBER OF
  PAGES:\n");
    scanf("%d",&n);

  printf("\n ENTER THE PAGE NUMBER :\n");

  for(i=1;i<=n;i+
  +)
  scanf("%d",&a[
  i]);
  printf("\n ENTER THE NUMBER OF FRAMES :");

  scanf("%d",&n
    o);
    for(i=0;i<no;i
    ++)
```

```
        frame[i]=
         -1;j=0;
        printf("\tref string\t page
        frames\n");for(i=1;i<=n;i++)
        {

                printf("%d\t\t",a[i]);
                avail=0;
                for(k=0;k<no;k++)
                if(frame[k]==a[i])
                  avail=1;

                   if (avail==0)

                  {

                    frame[j]=a[i];
                    j=(j+1)%no;
                    count++;
                    for(k=0;k<no;k
                    ++)
                    printf("%d\t",frame[k]);

                   }

                 printf("\n");

         }

            printf("Page Fault Is
            %d",count);return 0;
    }
```

OUTPUT:

```
ENTER THE NUMBER OF PAGES: 20


ENTER THE PAGE NUMBER :    7 0 1 2 0 3 0 4 2 3 0 3
2 1 2 0 1 7 0 1ENTER THE NUMBER OF FRAMES : 3
```

```
Ref string  Page frames

7             7    -1    -1
0             7     0    -1
1            7     0     1
2            2     0     1
0
3            2     3     1
0            2     3     0
4            4     3     0
2            4     2     0
3            4     2     3
0            0     2     3
3
2
1            0     1     3
2            0     1     2
0
1
7            7     1     2
0            7     0     2
1            7     0     1


Page Fault Is 15
```

b)LRU:

On a page fault, the frame that was least recently used in replaced.

PROGRAM:

/* C Program to implement LRU Page Replacement algorithms */

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

      int
      q[20],p[50],c=0,c1,d,f,i,j,k=0,n,r,t,b[20],c2[20]
      ;printf("Enter no of pages:");
      scanf("%d",&n);

      printf("Enter the reference
      string:");for(i=0;i<n;i++)
     scanf("%d",&p[i]);
      printf("Enter no of
      frames:");
      scanf("%d",&f);
      q[k]=p[k];
      printf("\n\t%d\n",q[k]
      );c++;
      k++;

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

     {

          c1=0;

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

          {

               if(p[i]!=q[
               j])c1++;
          }if(c1==f)

          {

               c++;

               if(k<f)

               {

                     q[k]=p[i
                     ];k++;
                     for(j=0;j<k;j++
                     )
                     printf("\t%d",q[
                     j]);printf("\n");
               }

               else

               {

                     for(r=0;r<f;r++)

                     {

                           c2[r]=0;

                           for(j=i-1;j<n;j--)

                           {

                           if(q[r]!=p[
                           j])
                           c2[r]++;
                           else
                           brea
                           k;
                     }

               }

               for(r=0;r<f;r
                    ++)
                    b[r]=c2[r];
                    for(r=0;r<f;r
                    ++)
                    {

                        for(j=r;j<f;j++)

                        {
                            if(b[r]<b[j])

                            {

                                t=b[r];
                                b[r]=b[j
                                ];
                                b[j]=t;
                            }


                        }


                    }

                    for(r=0;r<f;r++)

                    {

                        if(c2[r]==b[0])

                        q[r]=p[i];
                        printf("\t%d",q[
                        r]);
                    }

                    printf("\n");


                }


            }


        }

        printf("\nThe no of page faults is %d",c);
```

```
}
```

OUTPUT:

```
Enter no of pages:10
Enter the reference string:7 5 9 4 3 7
9 6 2 1Enter no of frames:3

    7
    7   5
    7   5   9
    4   5   9

    4   3   9
    4   3   7
    9   3   7
    9   6   7
    9   6   2
    1   6   2


The no of page faults is 10
```

c) LFU:

Pages with a current copy on disk are first choice for pages to be removed when more memory isneeded. To facilitate Page Replacement Algorithms, a table of valid or invalid bits (also called *dirty bits*) is maintained.

PROGRAM:

/* C Program to implement LFU Page Replacement algorithms */

```
#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
```

EXERCISE-11

AIM: Simulate the following File allocation strategies

a) **SEQUENCEDDESCRIPTION:**

Each file occupies a contiguous set of blocks on the disk. For example, if a file requires n blocks and is given a block b as the starting location, then the blocks assigned to the file will be: *b, b+1, b+2,……b+n-1.* This meansthat given the starting block address and the length of the file (in terms of blocks required), we can determine the blocks occupied by the file.
The directory entry for a file with contiguous allocation contains

- Address of starting block
- Length of the allocated portion.

PROGRAM:

/* C Program to implement Sequenced File allocation strategies */

```
#include        <
stdio.h>
#include<conio
.h>void main()
{

int f[50], i, st, len, j, c, k,
count = 0;clrscr();
for(i=0;i<50;i++)
f[i]=0;
printf("Files Allocated are
: \n");x: count=0;
printf("Enter starting block and length of
files: ");scanf("%d%d", &st,&len);
for(k=st;k<(st+len);k++)
 if(f[k]==0)
count++;
if(len==cou
nt)
{

 for(j=st;j<(st+len);j+
 +) if(f[j]==0)
 {
 f[j]=
 1;
 printf("%d\t%d\n",j,f[j]);

 }

 if(j!=(st+len-1))

 printf(" The file is allocated to disk\n");

}

else

printf(" The file is not allocated \n");

printf("Do you want to enter more file(Yes -
1/No - 0)");scanf("%d", &c);
if(c==
1)goto
x; else
exit();
getch();

}
```

OUTPUT:

```
Files Allocated are :

Enter starting block and length of
files:17  417 1
18   1

19   1

20   1

The file is allocated to disk
```

```
Do you want to enter more file(Yes - 1/No
– 0) 1Enter starting block and length of
files:21  3
21   1

22   1

23   1

The file is allocated to disk

Do you want to enter more file(Yes - 1/No – 0)  0
```

b)INDEXED :

A special block known as the **Index block** contains the pointers to all the blocks occupied by a file. Each file hasits own index block. The ith entry in the index block contains the disk address of the ith file block. The directoryentry contains the address of the index block as shown in the image

PROGRAM:

/* C Program to implement Indexed File allocation strategies */

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>void main()
{

int f[50], index[50],i, n, st, len, j, c, k,
ind,count=0;clrscr();
for(i=0;i<50;i+
+) f[i]=0;
x:printf("Enter the index
block: ");scanf("%d",&ind);
if(f[ind]!=1)

{

 printf("Enter no of blocks needed and no of files for the index %d on
the disk : \n",ind);
 scanf("%d",&n);

}
```

```
else

{

 printf("%d index is already allocated
 \n",ind);goto x;
}

y: count=0;
for(i=0;i<n;i
++)
{

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

 if(f[index[i]]=
 =0)count++;
}

if(count==n)

{

 for(j=0;j<n;j++)
 f[index[j]]=1;
 printf("Allocated\n"
 ); printf("File
 Indexed\n");
 for(k=0;k<n;k++)
 printf("%d---->%d : %d\n",ind,index[k],f[index[k]]);
}

else

{

printf("File in the index is already
allocated \n");printf("Enter another file
indexed");
goto y;

}
```

```
printf("Do you want to enter more file(Yes -
1/No - 0)");scanf("%d", &c);
if(c==
1)goto
x; else
exit(0)
;
getch();

}
```

OUTPUT:

```
Enter the index block:  5

Enter no of blocks needed and no of files for the index 5
on the disk :4
1  2  3  4

Allocated
File
Indexed
5..........> 1  :  1

5..........> 2  :  1

5..........> 3  :  1

5..........> 4  :  1
Do you want to enter more file(Yes - 1/No -
0) 1Enter the index block:  4
 4  index is already
allocatedEnter the index
block:  6
Enter no of blocks needed and no of files for the index 6 on the disk :

1

2

File in the index is already
allocatedEnter another file
indexed 6 Allocated
```

```
File Indexed

6..........> 6 :  1

Do you want to enter more file(Yes - 1/No - 0)  0
```

c)LINKED :

Each file is a linked list of disk blocks which **need not be** contiguous. The disk blocks can be scattered anywhereon the disk. The directory entry contains a pointer to the starting and the ending file block. Each block contains apointer to the next block occupied by the file.

PROGRAM:

/* C Program to implement Linked File allocation strategies */

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
void main()

{

int f[50], p,i, st, len, j, c,
k, a;clrscr();
for(i=0;i<50;i+
+) f[i]=0;
printf("Enter how many blocks already
allocated: ");scanf("%d",&p);
printf("Enter blocks already
allocated: ");for(i=0;i<p;i++)
{

 scanf("%d",&
 a);f[a]=1;
}

x: printf("Enter index starting block and
length: ");scanf("%d%d", &st,&len);
k=len;
if(f[st]==
0)
{

 for(j=st;j<(st+k);j++)

 {

 if(f[j]==0)

 {
 f[j]=
 1;
 printf("%d---->%d\n",j,f[j]);

 }

 else

 {

 printf("%d Block is already
 allocated \n",j);k++;
 }

 }

}

else

printf("%d starting block is already allocated
\n",st); printf("Do you want to enter more
file(Yes - 1/No - 0)");scanf("%d", &c);
if(c==
1)goto
x; else
exit(0)
;
getch();

}
```

OUTPUT:

```
Enter how many blocks already
allocated: 3Enter blocks already
allocated:1 3 5
Enter index starting block and
length: 24
2..........> 1

3  Block is already
allocated4....> 1
5  Block is already
allocated6....> 1
7..........> 1

Do you want to enter more file(Yes - 1/No - 0) 0
```

EXERCISE-12

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

Description :

Shared memory is the fastest form of IPC available. Once the memory is mapped into the address space of the processes that are sharing the memory region, no kernel involvement occurs in passing data between the processes. What is normally required, however, is some form of synchronization between the processes that are storing and fetching information to and from the shared memory region. Shared memory is the most useful of the 3 structures. All the other IPC structures have similar system calls. Shared memory is memory that is accessible to a number of processes. By several orders of magnitude, it is the quickest way of sharing information among a set of processes. Keep in mind that shared memory is available on all operating systems. Only the calls will be different.

Syntax :

```
#include <sys/types.h>
 #include <sys/ipc.h>
#include <sys/shm.h>
```

int shmget (key_t key, size_t size, int shmflg)

Algorithm :

```
Step 1: Start
Step 2 : Create a shared memory using mhmget().
Step 3 : Store integer value in shared memory. (shmat())
Step 4 : Create a child process using fork().
Step 5 : Get a semaphore on shared memory using semget().
Step 6 : Increase the value of shared variable
Step 7 : Release the semaphore
Step 8 : Repeat step 4,5,6 in child process also.
Step 9 : Remove shared memory
Step 10: Stop
```

Program :

```
#include<stdio.h>
#include<sys/ipc.h>
#include<sys/stat.h>
#include<sys/msg.h>
#include<sys/sem.h>
#include<string.h>
int main()
```

```
{

            int shmid,sid=-1;
            int shmptr;
            pid_t pid;
            shmid=shmget(20,1024,0644|IPC_CREAT);
            shmptr=shmat(shmid,0,0);
            if(shmptr==1)
                      printf("error\n");
            else
                      printf("\nshared memory created\n");
            if(sid<0)
            {
                        if((sid=shmget(30,1,IPC_CREAT|0644))<0)
                                  printf("\nsemaphore is  created");
                        else
                                  printf("semaphore is not created\n");
            }
             printf("\nenter integer value");
             scanf("%d",&shmptr);
             printf("the entered value is:%d\n",shmptr);

           if((pid=fork())==0)
             {
                         wait(sid);
                         shmptr+=1;
                         printf("child value is:%d\n",shmptr);
                         signal(sid);
             }
             else if(pid>0)
             {
                         shmptr-=1;
                         printf("the parent value is :%d\n",shmptr);
             }
 }
```

Output :

```
[stu515@bvcits ~]$ cc w2a.c
[stu515@bvcits ~]$ a.out
Shared memory created
semaphore is created
enter integer value 41
```

```
the entered value is: 41
the parent value is: 40
the child value is: 42
[stu515@bvcits ~]$
```

Exercise-13

Write a C program to simulate producer and consumer problem usingsemaphores

SOURCE CODE:

```
#include<stdio.h>
int mutex=1,full=0,empty=3,x=0;
main()
{
   int n;
   void producer();
   void consumer();
   int wait(int);
   int signal(int);
   printf("\n 1.Producer  \n 2.Consumer \n 3.Exit");
   while(1)
   {
      printf("\n Enter your choice:");
      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("\n Producer 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:

```
[examuser35@localhost Jebastin]$ cc pc.c
 1.Producer
 2.Consumer
 3.Exit
 Enter your choice:1
 Producer produces the item 1
 Enter your choice:1
 Producer produces the item 2
 Enter your choice:1
 Producer produces the item 3
 Enter your choice:1
 Buffer is full
```

```
Enter your choice:2
Consumer consumes item 3
Enter your choice:2
Consumer consumes item 2
Enter your choice:2
Consumer consumes item 1
Enter your choice:2
Buffer is empty
Enter your choice:3
```

EXERCISE-14

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

What is a Thread?

A thread is a single sequence stream within in a process. Because threads have some of the properties of processes, they are sometimes called *lightweight processes*.

What are the differences between process and thread?

Threads are not independent of one other like processes as a result threads shares with other threads their code section, data section and OS resources like open files and signals. But, like process, a thread has its own program counter (PC), a register set, and a stack space.

Why Multithreading?

Threads are popular way to improve application through parallelism. For example, in a browser, multiple tabs can be different threads. MS word uses multiple threads, one thread to format the text, other thread to process inputs, etc.

Threads operate faster than processes due to following reasons:
1) Thread creation is much faster.
2) Context switching between threads is much faster.
3) Threads can be terminated easily
4) Communication between threads is faster.

A simple C program to demonstrate use of pthread basic functions

In main() we declare a variable called thread_id, which is of type pthread_t, which is an integer used to identify the thread in the system. After declaring thread_id, we call pthread_create() function to create a thread.
pthread_create() takes 4 arguments.
The first argument is a pointer to thread_id which is set by this function.
The second argument specifies attributes. If the value is NULL, then default attributes shall be used.
The third argument is name of function to be executed for the thread to be created.

The fourth argument is used to pass arguments to the function, myThreadFun.
The pthread_join() function for threads is the equivalent of wait() for processes. A call to pthread_join blocks the calling thread until the thread with identifier equal to the first argument terminates.

Program:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>  //Header file for sleep(). man 3 sleep for details.
#include <pthread.h>

// A normal C function that is executed as a thread
// when its name is specified in pthread_create()
void *myThreadFun(void *vargp)
{
   sleep(1);
   printf("Printing GeeksQuiz from Thread \n");
   return NULL;
}

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

Output:

```
gfg@ubuntu:~/$ cc multithread.c -lpthread
gfg@ubuntu:~/$ ./a.out
Before Thread
Printing GeeksQuiz from Thread
After Thread
gfg@ubuntu:~/$
```

EXERCISE-15

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

A concurrent programming environment lets us designate tasks that can run in parallel. It also lets us specify how we would like to handle the communication and synchronization issues that result when concurrent tasks attempt to talk to each other and share data.

Because most concurrent programming tools and languages have been the result of academic research or have been tailored to a particular vendor's products, they are often inflexible and hard to use. Pthreads, on the other hand, is designed to work across multiple vendors' platforms and is built on top of the familiar UNIX C programming interface. Pthreads gives you a simple and portable way of expressing multithreading in your programs.

Program:

```
#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);
}
```

OUT PUT ::

```
$ cc w8.c – l pthread
$./a.out
thread1
i=1
i=2;
i=3
thread2
j=1
j=2
j=3

j=4
kj=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
```