Experiment-1:

1. a) Study of Unix/Linux general purpose utility command list: man, who, cat, cd, cp, ps, ls,

mv, rm, mkdir, rmdir, echo, more, date, time, kill, history, chmod, chown, finger, pwd, cal, logout,

- **shutdown.**

1. **man** - view manual pages for UNIX commands

    Syntax: man [-s section] item

    ```
    $ man cat
    ```

2. **who** - displays the list of users logged presently

    Syntax: who [option] ... [file}[argl]

    ```
    $who
    ```

3. cat - concatenate files and show contents to the standard output

Syntax: cat [option] ...[file}

```
$cat> file]
  hello
```

4. cd - change directory

Syntax: cd [option] directory

```
$ cd dirl
```

5. cp - copy files

Syntax: cp [option] source destination

```
$ cp filel file2
```

6. ps - Reports process status

Syntax: Ps [options]

```
$ps
```

7. ls - List files & directories

Syntax: ls [option] [file]

```
$ ls
```

8. mv - Rename or move files & directories to another location

Syntax: mv [option] source destination

```
$ mv file] file2
```

9. rm - removes files & directories

Syntax: rm [option}...[file}

```
$ rmfilel
```

10. mkdir - makes new directory

Syntax: mkdir [option] directory

```
$ mkdir dirl
```

11. rmdir - removes directory

Syntax: rmdir [option] directory

```
$ rmdir dirl
```

12. echo - displays a line of text to the standard output

Syntax: $ echo "Hello, World!"

```
$ echo $x
```

13. more - basic pagination when viewing text files or parsing UNIX commands output

Syntax: more [-options] [-num] [+/pattern] [+linenum] [Jile_name]

```
$ more +/reset sample.txt
$ more +30 sample.txt
```

14. date - Show current date & time

Syntax: date [+format]

```
$ date +%d/%m/%y
```

15. time - shows current time

$ time

16. kill - terminates a specified process

Syntax: Kill PID

$ kill 4901

17. history - displays history commands in the current session

Syntax: history [options]

$ history

18. chmod - change file / directory access permissions

<u>Symbolic mode</u>

Syntax: chmod [ugoa][[+-=][mode]] file

- The first optional parameter indicates who - this can be (u)ser, (g)roup, (o)thers or (a)ll
- The second optional parameter indicates opcode - this can be for adding (+), removing (-

) or assigning (=) a permission.

- The third optional parameter indicates the mode - this can be (r)ead, (w)rite, or e(x)ecute. $ chmod o-w file]

<u>Numeric mode</u>

Syntax: chmod [mode] file

- The mode is a combination of three digits - the first digit indicates the permission for the

user, the second digit for the group, and the third digit for others.

- Each digit is computed by adding the associated permissions. Read permission is ' 4' , write permission is '2' and execute permission is ' 1'.

$ chmod 777 file]

19. chown - change file/ directory ownership

Syntax: chown [owner] [file]

$ chown user2 file]

20. finger - displays user info if login is known

Syntax: finger usemame

21. pwd - confirm current directory

Syntax: pwd [option]

```
$pwd
```

22. cal - displays calendar

Syntax: cal [[month] year]

```
$ cal 6 2019
```

23. logout - logs off UNIX

Syntax: Logout [options]

```
$ logout
```

24. shutdown - brings the system down in a secure way

Syntax: shutdown [options]

```
$ shutdown
```

b) study of vi editor.

- Visual editor for editing programs.
- The vi editor is the most popular and classic text editor in the Linux family.
- This editor enables you to edit lines in context with other lines in the file.
- Below, are some reasons which make it a widely used editor?
    - ♦ It is available in almost all Linux Distributions
    - ♦ It works the same across different platforms and Distributions
    - ♦ It requires very few resources.
    - ♦ it is more user-friendly than other editors such as theed or the ex.
- we can use the vi editor to edit an existing file or to create a new file from scratch.
- We can also use this editor to just read a text file.
- An improved version of the vi editor which is called the VIM has also been made

available now. Here, VIM stands for Vi IMproved.

Starting the vi Editor

To launch the vi Editor -Open the Terminal (CLI) and type

vi <**filename_NEW**> or <**filename_EXISTING**>

& if you specify an existing file, then the editor would open it for you to edit. Else, you can

create a new file.

The following table lists out the basic commands to use the vi editor –

S.No	command	description
1	Vi filename	Creates a new file if it already does not exist, otherwise opens an
		Existing file
2	vi-R filename	Opens an existing file in the read-only mode

Operation Modes

To work on vi editor, we need to understand its operation modes.

- **Command mode** - This mode enables you to perform administrative tasks such as saving the files, executing the commands, moving the cursor, cutting (yanking) and pasting the lines or words, as well as finding and replacing. In this mode, whatever you type is interpreted as a command.
- **Insert mode** - This mode enables you to insert text into the file. Everything that's typed in this mode is interpreted as input and placed in the file

vi always starts in the command mode. To enter text, you must be in the insert mode for which simply type i. To come out of the insert mode, press the Esc key, which will take you back to the command mode.

vi Editing commands

Note: You should be in the **"command mode"** to execute these commands.

vi editor is **case-sensitive** so make sure to type the commands in the right letter-case, otherwise

you will end up making undesirable changes to the file.

S.NO	COMMAND	DESCRIPTION
1	i	Insert at cursor (goes into insert mode)
2	a	Write after cursor (goes into insert mode)
3	A	Write at the end of line (goes into insert mode)
4	ESC	Terminate insert mode
5	u	Undo last change
6	U	Undo all changes to the entire line
7	o	Open a new line(goes into insert mode)
8	dd	Delete line
9	3dd	Delete 3 lines
10	D	Delete content of line after the cursor
11	C	Delete contents of a line after the cursor and in sert new text. Press ESC key to end insertion
12	dw	Delete word
13	4dw	Delete 4 words
14	cw	Change word
15	x	Delete character at the cursor
16	r	Replace character
17	R	Overwrite character from cursor onward
18	s	Substitute one character under cursor continue to insert
19	S	Substitute entire line and begin to insert at the beginning of the line
20	-	Change case of individual character

Moving within a file

Note: You need to be in the "command mode" to move within a file. The default keys for navigation are mentioned below else; you can also use the arrow keys on the keyboard.

s.no	command	description
1	K	Move cursor up
2	J	Move cursor down
3	H	Move cursor left
4	1	Move cursor right

Saving and closing the file

Note: You should be in the "command mode" to exit the editor and save changes to the file.

s.no	command	Description
1	**Shift +zz**	Save the file and quit
2	**:w**	Save the file but keep it open
3	**:q**	Quit without saving
4	**:wq**	Save the file and quit

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

What Is a Shell?

A shell is a program that provides an interface between a user and an operating system (OS)kernel.

An OS starts a shell for each user when the user logs in or opens a terminal or console window.

A kernel is a program that:

- Controls all computer operations.
- Coordinates all executing utilities
- Ensures that executing utilities do not interfere with each other or consume all system resources.
- Schedules and manages all system processes.

By interfacing with a kernel, a shell provides a way for a user to execute utilities and programs.

Experiment 2:

UNIX/LINUX based exercises to practice/simulate File system related system calls and some of the process management concepts in LINUX Environment.

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

AIM:

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

Program:

```
#include <stdio.h>

#include <unistd.h>

#include <fcntl.h>

void typefile (char *filename)

{

int fd, nread;

char buf[1024];

fd = open (filename, O_RDONLY);

if (fd == -1) {

perror (filename);

return;

}

while ((nread = read (fd, buf, sizeof (buf))) > 0)

write (1, buf, nread);
close (fd);
}
int
main (int argc, char **argv)
{
int argno;
for (argno = 1; argno < argc; argon++ )
typefile (argv[argno]);
exit (0);
}
```

Output:

```
student@ubuntu:~$gcc –o prg10.out prg10.c
student@ubuntu:~$cat > ff
hello
hai
student@ubuntu:~$./prg10.out ff
hello
hai
```

2b) Write a C program to emulate the UNIX ls –l command.

AIM:

Write a C program to emulate the Unix ls-l command.

Program:

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

Output:

```
guest-glcbIs@ubuntu:~$gcc –o lsc.out lsc.c
guest-glcbIs@ubuntu:~$./lsc.out
total 100
-rwxrwx—x 1 guest-glcbls guest-glcbls 140 2012-07-06 14:55 f1
drwxrwxr-x 4 guest-glcbls guest-glcbls 140 2012-07-06 14:40 dir1
child complete
```

——

2c) Write a C program that illustrates how to execute two commands concurrently with a command pipe Ex: - ls –l | sort.

AIM:

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

Program:

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

```
#include <unistd.h>
#include <sys/types.h>
#include <stdlib.h>
int main()
{
int pfds[2];
char buf[30];
if(pipe(pfds)==-1)
{
perror("pipe failed");
exit(1);
}
if(!fork())
{
close(1);
dup(pfds[1];
system (“ls –l”);
}
else
{
printf("parent reading from pipe \n");
while(read(pfds[0],buf,80))
printf("%s \n" ,buf);
}
}
```

Output:

```
[student@gcet ~]$ vi pipes2.c

[student@gcet ~]$ cc pipes2.c

[student@gcet ~]$ ./a.out

Parent reading from pipe

Total 24

-rwxrwxr-x l student student 5563Aug 3 10:39 a.out

-rw-rw-r—l

Student student 340 jul 27 10:45 pipe2.c

-rw-rw-r—l student student

Pipes2.c

-rw-rw-r—l student student 401 34127 10:27 pipe2.c

Student
```

2d) Write a C program that illustrates two processes communicating using shared memory.

Aim:

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

Program:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <unistd.h>

#define BUF_SIZE 1024
```

```
int main() {
  int shmid;
  char *shm;

  // Create a shared memory segment
  shmid = shmget(IPC_PRIVATE, BUF_SIZE, 0666|IPC_CREAT);
  if (shmid < 0) {
    perror("shmget failed");
    exit(1);
  }

  // Attach the shared memory segment to the current process
  shm = (char*)shmat(shmid, NULL, 0);
  if (shm == (char*)-1) {
    perror("shmat failed");
    exit(1);
  }

  int pid = fork();
  if (pid == 0) {
    // Child process
    strcpy(shm, "Hello, World!");
    printf("Child: wrote message to shared memory: %s\n", shm);
    shmdt(shm);
  } else {
    // Parent process
    sleep(1);
    printf("Parent: read message from shared memory: %s\n", shm);
    shmdt(shm);

    // Destroy the shared memory segment
    shmctl(shmid, IPC_RMID, NULL);
  }
  return 0;
}
```

Output :

```
Student@ubuntu:~$gcc shm.c
 Student@ubuntu:~$ ./a.out
Child:wrote message to shared memory:hello world!
Parent:read message from shared memory: hello world!
```

Experiment 3:

3-a) write a c program for fcfs scheduling.

Aim:

To write a source code of the c program for the fcfs scheduling.

Program:

```c
#include <stdio.h>
int main()
{
    int pid[15];
    int bt[15];
    int n;
    printf("Enter the number of processes: ");
    scanf("%d",&n);

        printf("Enter process id of all the processes: ");
        for(int i=0;i<n;i++)
        {
            scanf("%d",&pid[i]);
        }

        printf("Enter burst time of all the processes: ");
        for(int i=0;i<n;i++)
        {
            scanf("%d",&bt[i]);
        }

        int i, wt[n];
        wt[0]=0;

        //for calculating waiting time of each process
        for(i=1; i<n; i++)
        {
            wt[i]= bt[i-1]+ wt[i-1];
        }

        printf("Process ID     Burst Time     Waiting Time     TurnAround Time\n");
        float twt=0.0;
        float tat= 0.0;
        for(i=0; i<n; i++)
        {
            printf("%d\t\t", pid[i]);
            printf("%d\t\t", bt[i]);
            printf("%d\t\t", wt[i]);

            //calculating and printing turnaround time of each process
            printf("%d\t\t", bt[i]+wt[i]);
            printf("\n");

            //for calculating total waiting time
            twt += wt[i];

            //for calculating total turnaround time
            tat += (wt[i]+bt[i]);
        }
        float att,awt;

        //for calculating average waiting time
        awt = twt/n;

        //for calculating average turnaround time
        att = tat/n;
        printf("Avg. waiting time= %f\n",awt);
        printf("Avg. turnaround time=%f",att);
}
```

Output:

```
Enter the number of processes: 3
Enter process id of all the processes: 1 2 3
Enter burst time of all the processes: 5 11 11
Process ID      Burst Time      Waiting Time TurnAround Time
1                5               0               5
2                11              5               16
3                11              16              27
Avg. waiting time= 7.000000
Avg. turnaround time= 16.000000
```

3-b)write a c program for SJF scheduling algorithm.

Aim:

To write a source code of the c program for the SJF scheduling.

Program:

```
#include<stdio.h>
int main()
{
    int bt[20],p[20],wt[20],tat[20],i,j,n,total=0,totalT=0,pos,temp;
    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;
        }

        //sorting of burst times
        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[pos];
            bt[pos]=temp;

            temp=p[i];
            p[i]=p[pos];
            p[pos]=temp;
        }

        wt[0]=0;

        //finding the waiting time of all the processes
        for(i=1;i<n;i++)
        {
            wt[i]=0;
            for(j=0;j<i;j++)
                 //individual WT by adding BT of all previous completed processes
                wt[i]+=bt[j];

            //total waiting time
            total+=wt[i];
        }

        //average waiting time
        avg_wt=(float)total/n;

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

```c
            //turnaround time of individual processes
            tat[i]=bt[i]+wt[i];

            //total turnaround time
            totalT+=tat[i];
            printf("\np%d\t\t %d\t\t %d\t\t\t%d",p[i],bt[i],wt[i],tat[i]);
        }

        //average turnaround time
        avg_tat=(float)totalT/n;
        printf("\n\nAverage Waiting Time=%f",avg_wt);
        printf("\nAverage Turnaround Time=%f",avg_tat);
    }
```

Output:

```
Enter number of process:4

Enter Burst Time:
p1:5
p2:4
p3:12
p4:7

Process       Burst Time      Waiting Time    Turnaround Time
p2               4               0                   4
p1               5               4                   9
p4               7               9                  16
p3               12              16                 28

Average Waiting Time=7.250000
Average Turnaround Time=14.250000
```

Experiment-4

4.write a c program for priority scheduling.

Aim:

To write a source code of the c program for priority scheduling.

Program:

```c
#include <stdio.h>

//Function to swap two variables
void swap(int *a,int *b)
{
    int temp=*a;
    *a=*b;
    *b=temp;
}
    int main()
    {
        int n;
        printf("Enter Number of Processes: ");
        scanf("%d",&n);

        // b is array for burst time, p for priority and index for process id
        int b[n],p[n],index[n];
        for(int i=0;i<n;i++)
        {
            printf("Enter Burst Time and Priority Value for Process %d: ",i+1);
            scanf("%d %d",&b[i],&p[i]);
            index[i]=i+1;
        }
        for(int i=0;i<n;i++)
        {
            int a=p[i],m=i;

            //Finding out highest priority element and placing it at its desired position
            for(int j=i;j<n;j++)
            {
                if(p[j] > a)
                {
                    a=p[j];
                    m=j;
                }
            }

            //Swapping processes
            swap(&p[i], &p[m]);
            swap(&b[i], &b[m]);
            swap(&index[i],&index[m]);
        }

        // T stores the starting time of process
        int t=0;

        //Printing scheduled process
        printf("Order of process Execution is\n");
        for(int i=0;i<n;i++)
        {
            printf("P%d is executed from %d to %d\n",index[i],t,t+b[i]);
            t+=b[i];
        }
        printf("\n");
        printf("Process Id      Burst Time      Wait Time     TurnAround Time\n");
        int wait_time=0;
        for(int i=0;i<n;i++)
        {
            printf("P%d          %d          %d          %d\n",index[i],b[i],wait_time,wait_time + b[i]);
            wait_time += b[i];
        }
        return 0;
    }
```

Output:

```
Enter Number of Processes: 3
Enter Burst Time and Priority Value for Process 1: 10
2
```

```
Enter Burst Time and Priority Value for Process 2: 5
0
Enter Burst Time and Priority Value for Process 3: 8
1
Order of process Execution is
P1 is executed from 0 to 10
P3 is executed from 10 to 18
P2 is executed from 18 to 23

Process Id      Burst Time    Wait Time     TurnAround
Time
    P1             10            0             10
    P3             8            10             18
    P2             5            18             23
```

Experiment-5

5.write a c program for round robin scheduling.

Aim:

To write a source code of the c program for round robin scheduling.

Program:

```c
#include<stdio.h>

int main()
{
    //Input no of processed
    int  n;
    printf("Enter Total Number of Processes:");
    scanf("%d", &n);
    int wait_time = 0, ta_time = 0, arr_time[n],
burst_time[n], temp_burst_time[n];
        int x = n;

        //Input details of processes
        for(int i = 0; i < n; i++)
        {
            printf("Enter Details of Process %d \n", i + 1);
            printf("Arrival Time:  ");
            scanf("%d", &arr_time[i]);
            printf("Burst Time:   ");
            scanf("%d", &burst_time[i]);
            temp_burst_time[i] = burst_time[i];
        }

        //Input time slot
        int time_slot;
        printf("Enter Time Slot:");
        scanf("%d", &time_slot);

        //Total indicates total time
        //counter indicates which process is executed
        int total = 0,  counter = 0,i;
        printf("Process ID       Burst Time       Turnaround Time      Waiting Time\n");
        for(total=0, i = 0; x!=0; )
        {
            // define the conditions
            if(temp_burst_time[i] <= time_slot && temp_burst_time[i] > 0)
            {
                total = total + temp_burst_time[i];
                temp_burst_time[i] = 0;
                counter=1;
            }
            else if(temp_burst_time[i] > 0)
            {
                temp_burst_time[i] = temp_burst_time[i] - time_slot;
                total  += time_slot;
            }
            if(temp_burst_time[i]==0 && counter==1)
            {
                x--; //decrement the process no.
```

```c
                printf("\nProcess No %d  \t\t %d\t\t\t\t %d\t\t\t %d", i+1, burst_time[i],
                       total-arr_time[i], total-arr_time[i]-burst_time[i]);
                wait_time = wait_time+total-arr_time[i]-burst_time[i];
                ta_time += total -arr_time[i];
                counter =0;
            }
            if(i==n-1)
            {
                i=0;
            }
            else if(arr_time[i+1]<=total)
            {
                i++;
            }
            else
            {
                i=0;
            }
        }
        float average_wait_time = wait_time * 1.0 / n;
        float average_turnaround_time = ta_time * 1.0 / n;
        printf("\nAverage Waiting Time:%f", average_wait_time);
        printf("\nAvg Turnaround Time:%f", average_turnaround_time);
        return 0;
    }
```

Output:

```
Enter Total Number of Processes:3
Enter Details of Process 1
Arrival Time:  0
Burst Time:   10
Enter Details of Process 2
Arrival Time:  1
Burst Time:   8
Enter Details of Process 3
Arrival Time:  2
```

```
Burst Time:    7
Enter Time Slot:5

Process ID        Burst Time        Turnaround Time
Waiting Time

Process No 1          10                  20
10
Process No 2          8                   22
14
Process No 3          7                   23
16

Average Waiting Time: 13.333333
Avg Turnaround Time: 21.666666
```

Experiment-6

6.write a c program for banker's algorithm.

Aim:

To write a source code of the c program for banker's alogrithm.

Program:

```
#include <stdio.h>
int main()
{
   // P0, P1, P2, P3, P4 are the Process names here

   int n, m, i, j, k;
   n = 5;                     // Number of processes
   m = 3;                      // Number of resources
   int alloc[5][3] = {{0, 1, 0},  // P0 // Allocation Matrix
```

```
                 {2, 0, 0},  // P1
                 {3, 0, 2},  // P2
                 {2, 1, 1},  // P3
                 {0, 0, 2}}; // P4

int max[5][3] = {{7, 5, 3},  // P0 // MAX Matrix
                {3, 2, 2},  // P1
                {9, 0, 2},  // P2
                {2, 2, 2},  // P3
                {4, 3, 3}}; // P4

int avail[3] = {3, 3, 2}; // Available Resources

int f[n], ans[n], ind = 0;
for (k = 0; k < n; k++)
{
    f[k] = 0;
}
int need[n][m];
for (i = 0; i < n; i++)
{
    for (j = 0; j < m; j++)
        need[i][j] = max[i][j] - alloc[i][j];
}
```

```
int y = 0;
for (k = 0; k < 5; k++)
{
    for (i = 0; i < n; i++)
    {
        if (f[i] == 0)
        {
            int flag = 0;
            for (j = 0; j < m; j++)
            {
                if (need[i][j] > avail[j])
                {
                    flag = 1;
                    break;
                }
            }
            if (flag == 0)
            {
                ans[ind++] = i;
                for (y = 0; y < m; y++)
                    avail[y] += alloc[i][y];
                f[i] = 1;
            }
       }
```

```
        }
    }
    int flag = 1;
    for (int i = 0; i < n; i++)
    {
        if (f[i] == 0)
        {
            flag = 0;
            printf("The following system is not safe");
            break;
        }
    }
    if (flag == 1)
    {
        printf("Following is the SAFE Sequence\n");
        for (i = 0; i < n - 1; i++)
            printf(" P%d ->", ans[i]);
        printf(" P%d", ans[n - 1]);
    }
    return (0);
}
```

Output:

```
Following is the SAFE Sequence
```

P1 -> P3 -> P4 -> P0 -> P2

....

Experiment-7

7.write a c program to simulate sequencial file allocation strategy.

Aim:

To write a source code of the c program for simulate sequencial file allocation strategy.

Program:

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_BLOCKS 100
int main() {
 int file_size, block_size, num_blocks, blocks[MAX_BLOCKS], i;
 // Get input from user
 printf("Enter file size: ");
 scanf("%d", &file_size);
 printf("Enter block size: ");
 scanf("%d", &block_size);
 // Calculate number of blocks needed
 num_blocks = file_size / block_size;
 if (file_size % block_size != 0) {
 num_blocks++;
 }
 // Allocate blocks sequentially
 for (i = 0; i < num_blocks; i++) {
 blocks[i] = i;
 }
 // Print allocated blocks
 printf("Allocated blocks: ");
 for (i = 0; i < num_blocks; i++) {
 printf("%d ", blocks[i]);
 }
 printf("\n");
 return 0;
}
```

Output:

```
Enter file size:100
Enter block size:25
Allocated blocks:0 1 2 3
```

Experiment-8

8.write a c program to simulate linked file allocation strategy.

Aim:

To write a source code of the c program for simulate linked file allocation strategy.

Program:

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

```
#define MAX_FILE 10
#define MAX_BLOCK 100
struct file {
 char name[10];
 int start_block;
 int length;
} files[MAX_FILE];
int blocks[MAX_BLOCK];
void init_blocks(int n) {
 int i;
 for (i = 0; i < n; i++) {
 blocks[i] = 0;
 }
}
int allocate_blocks(int n) {
 int i;
 for (i = 0; i < MAX_BLOCK; i++) {
 if (blocks[i] == 0) {
 int j;
 for (j = i; j < i + n; j++) {
 if (blocks[j] != 0) {
 break;
 }
 }
 if (j == i + n) {
 return i;
 }
 i = j;
 }
 }
 return -1;
}
void main() {
 int n, i;
 printf("Enter the number of files: ");
 scanf("%d", &n);
 printf("Enter information for %d files:\n", n);
 for (i = 0; i < n; i++) {
 printf("Enter file name: ");
 scanf("%s", files[i].name);
 printf("Enter file length: ");
 scanf("%d", &files[i].length);
 int start_block = allocate_blocks(files[i].length);
 if (start_block == -1) {
 printf("No contiguous block of size %d found to allocate file %s\n",
files[i].length, files[i].name);
 continue;
 }
 files[i].start_block = start_block;
 int j;
 for (j = start_block; j < start_block + files[i].length; j++) {
 blocks[j] = 1;
 }
 }
 printf("\nFile\tStart_Block\tLength\n");
 for (i = 0; i < n; i++) {
 printf("%s\t%d\t\t%d\n", files[i].name, files[i].start_block,
files[i].length);
 }
}
```

Output:

```
Enter the number of files:5
Enter information for 5 files:
Enter file name:a1
Enter file length: 8
Enter file name:b1
Enter file length: 25
Enter file name:c1
Enter file length: 15
Enter file name:d1
Enter file length: 14
Enter file name:e1
Enter file length: 9

File        Start_Block        length

a1              0                  8

b1             8                  25

c1             33                 15

d1             48                 14

e1              62                 9
```

Experiment-9

9.write a c program to simulate Indexed file allocation strategy.

Aim:

To write a source code of the c program for simulate Indexed file allocation strategy.

Program:

```
#include <stdio.h>

#include <stdlib.h>

#define MAX_BLOCKS 50

struct index_block {

 int block_num;

};

int main() {

 int file_size, block_size, num_blocks, i, j, num_index_blocks,
index_block_size;

 struct index_block index[MAX_BLOCKS];
```

```
int data[MAX_BLOCKS][MAX_BLOCKS];
// Get input from user
printf("Enter file size: ");
scanf("%d", &file_size);
printf("Enter block size: ");
scanf("%d", &block_size);
// Calculate number of blocks needed
num_blocks = file_size / block_size;
if (file_size % block_size != 0) {
num_blocks++;
}
// Calculate number of index blocks needed
num_index_blocks = num_blocks / (block_size / sizeof(struct
index_block));
if (num_blocks % (block_size / sizeof(struct index_block)) != 0) {
num_index_blocks++;
}
// Calculate size of index block
index_block_size = block_size / sizeof(struct index_block);
// Allocate data blocks
for (i = 0; i < num_blocks; i++) {
for (j = 0; j < block_size; j++) {
data[i][j] = i;
}
```

```
}
// Allocate index blocks
for (i = 0; i < num_index_blocks; i++) {
for (j = 0; j < index_block_size; j++) {
int block_num = i * index_block_size + j;
if (block_num < num_blocks) {
index[i].block_num = block_num;
} else {
index[i].block_num = -1;
}
}
}
// Print allocated blocks
printf("Allocated blocks:\n");
for (i = 0; i < num_index_blocks; i++) {
printf("Index block %d: ", i);
for (j = 0; j < index_block_size; j++) {
int block_num = index[i].block_num;
if (block_num == -1) {
printf("- ");
} else {
printf("%d ", block_num);
}
}
```

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

Output:

```
Enter file size:50
Enter block size:5
Allocated blocks:
Index block 0:0
Index block 1:1
Index block 2:2
Index block 3:3
Index block 4:4
Index block 5:5
Index block 6:6
Index block 7:7
Index block 8:8
Index block 9:9
```

Experiment-10

10.write a c program to simulate FIFO page replacement algorithm.

Aim:

To write a source code of the c program for simulate FIFO page replacement algorithm.

Program:

```
#include<stdio.h>
#include<conio.h>
int fr[3];
void main()
{
	void display();
	int i,j,page[12]={7,4,1,2,8,3,7,4,1,2,8,3};
	int flag1=0,flag2=0,pf=0,frsize=3,top=0;
	clrscr();
	for(i=0;i<3;i++)
	{
	fr[i]=-1;
	}
	for(j=0;j<12;j++)
	{
	flag1=0;
	flag2=0;
	for(i=0;i<12;i++)
	{
	if(fr[i]==page[j])
	{
	flag1=1;
	flag2=1;
	break;
	}
	}
	if(flag1==0)
	{
	for(i=0;i<frsize;i++)
	{
	if(fr[i]==-1)
	{
	fr[i]=page[j];
flag2=1;
break;
}
}
}
if(flag2==0)
{
fr[top]=page[j];
top++;
pf++;
if(top>=frsize)
top=0;
}
display();
}
printf("Number of page faults  : %d ",pf);
getch();
}
void display()
{
int i;
printf("\n");
for(i=0;i<3;i++)
printf("%d\t",fr[i]);
}
```

Output:

```
7    -1    -1
7    4     -1
7    4     1
2    4     1
2    8     1
2    8     3
7    8     3
7    4     3
7    4     1
2    4     1
2    8     1
2    8     3     Number of page faults  : 9
```

Experiment-11

11.write a c program to simulate LRU(least recently used) page replacement algorithm.

Aim:

To write a source code of the c program for simulate LRU page replacement algorithm.

Program:

```
#include<stdio.h>
#include<conio.h>
int fr[3];
void main()
{
void display();
int p[12]={7,0,1,2,0,3,0,4,2,3,0,3},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:

```
        7       -1      -1
        7       0       -1
        7       0       1
        2       0       1
        2       0       1
        2       0       3
        2       0       3
        4       0       3
        4       0       2
        4       3       2
        0       3       2
        0       3       2
 no of page faults :6
```

Experiment-12

12.write a c program to simulate OPTIMAL page replacement algorithm.

Aim:

To write a source code of the c program for simulate OPTIMAL page replacement algorithm.

Program:

```
#include<stdio.h>
#include<conio.h>
  int fr[3];
    void main()
     {
      void display();
      int p[12]={2,3,2,1,5,2,4,5,3,2,5,2},i,j,fs[3];
      int max,found=0,lg[3],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++)
                 {    lg[i]=0;
         }
              for(i=0;i<frsize;i++)
       {
       for(k=j+1;k<12;k++)
          {
           if(fr[i]==p[k])
              {
                lg[i]=k-j;
                break;
              }
          }
       }
                  found=0;
          for(i=0;i<frsize;i++)
           {
             if(lg[i]==0)
                 {
           index=i;
           found=1;
          break;
                   }
            }
         if(found==0)
           {
             max=lg[0];
               index=0;
               for(i=1;i<frsize;i++)
               {
                 if(max<lg[i])
          {
           max=lg[i];
           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  3  5
2  3  5
4  3  5
4  3  5
4  3  5
2  3  5
2  3  5
2  3  5
no of page faults : 3
```