EXPERIMENT NO . 1

BASIC LINUX COMMANDS

AIM

To familiarise basic Linux commands.

BASIC LINUX COMMANDS

1. pwd :- To know the working directory pwd command can be used. It gives us the
absolute path.
2. ls :- This command can be used to know what files are in the directory you are in.
● Open last edited file by using ls-t : This command sorts the file by modification time, showing the last edited file first;head-1 picks up this first file. To open the last edited file in the current directory use the combination of ls and head.
● ls-1 command can be used to display one file per time.
● Display all information about files using ls-l.
● ls-lh command can be used to display file size in human readable format.
● ls-a command can be used to display the hidden files in a directory.
● Display directory information using ls-lol.
3. cd :- cd command in linux known as change directory command. It is used to change
current working directory.
Different functionalities of cd command are:
● cd 1 : To change directory to the root directory.
● cd dir-1/dir-2/dir-3 : used to move inside a directory from a directory.
● cd ~ : To change directory to home directory
● cd.. : Used to move the parent directory to current directory.
4. mk dir :- This command allows the user to create directories. This can create
multiple directories as well as the permissions for the directories.
Syntax: mk dir[options..][directories..]
● --version : It displays the version number. Some information regarding the licence.
● --help : It displays the help related information and exits.
● -v : It displays a message for every directory created.

Syntax: mk dir –v [directories]
● -p : A flag which enables the command to create parent directories as necessary.
Syntax : mk dir –p[directories]
5. trmdir: This command is used to remove empty directories from the file system. It
removes each and every specified in the command line only if these directories are
empty options.
● -help : Print the general syntax of the command along with various options that can be used.
● rmdir-p : Each of the directory argumented is treated as path name of which all components will be removed.
● rmdir-v1-verbase : Displays the verbose information for every directory being processed.
● rmdir-vesion : Displays the version information and exit.
5. rm:- rm stands for remove. rm commands is used to remove objects such as files,
directories, symbolic links
Syntax: rm[option]..FILE
6. man:- Used to display the user manual of any command that we can run on the
terminal. It provides a detailed view of the command.
Syntax: $man[OPTION..]..[COMMAND NAME]
Options and examples
● No option: Displays the whole manual of the commands.
● Selection_num: Displays only the section number of the manual.
● -f option: Gives the section in which the given command is present.
● -a option: Displays all unavailable intro manual pages.
● -k option: Searches the given command as a regular expression in all manuals
and returns the manual pages with section number.
● -w option: Returns the location in which manual page of a given command is present.
7. cp- cp stands for copy. The command is used to copy files. It creates an exact image
of a file on a disk with different file name.
Syntax: cp[option]source destination
cp[option]source directory
cp[option]source-1 source-2 source-n directory

Arguments passed can be
● Two file names: Copies the content of first file to another file. If the file ‘n’ exists it
creates one and content is copied to it otherwise it is overwritten.
● One or more arguments: If the command has one or more arguments specifying file
names and following those arguments, an argument specifying directory name this
command copies each source file to destination directory with some name.
● Two directory names: This command copies al files of source directory to destination creating all files/directories needed.
8. mv- a mv command is used to move files and also to rename files.
Syntax to move files: mv<filename><directory path>

Syntax to rename files: mv<oldfilename><newfilename>
9. cat- cat command is used to create a file, display content of the file and to copy the
content of one file to another file.
Syntax: cat[option]…[file]
To create a file: cat<filename>
To display: cat<filename>
10. echo- echo command is used to display line of text/string that are passes as an
argument
Syntax: echo[option][string]
11. chmod- chmod command is used to change the read, write and execute permission
of files and directories.
Syntax: chmod[reference][operation][modes]files
12. ping- Used to check connectivity between 2 nodes i.e.whether the server is
connected. Full form of ping is packet internet groper
Syntax: ping<destination>
13. grep- The grep filter searches a file for a particular pattern of characters and
displays all lines that contain that pattern. The pattern that is searched in the file is
referred to as the regular expression (grep stands for global search for regular expression and print out).
Syntax: grep[options]pattern[files]

Options Description
● -c: Prints only a count of the line that match a pattern.
● -h: Display the matched lines, but do not display the file names.
● -i: Ignores case for matching.
● -l: Display list of filenames only.
● -n: Display the match lines and their line numbers.
● -v: This prints out all the lines that do not match the pattern.
● -e exp: specifies expression with this option can use multiple times.
● -f file: Take pattern from file one per line.
● -E: Treats the pattern as an extended regular expression.
● -W:Match whole word.
● -O: prints only the matched parts of a matching line with each part on a separate output line.
● -An: prints searched line and n lines after the result.
● -Bn: prints searched line and n lines before the result.
● -Cn: prints searched lines and n lines after & before the result.
14. cut – cut command is used for cutting out the sections from each line of files and
writing the result to standard output. It can be used to cut parts of a line by byte
position, characters and field. It is necessary to specify option with command otherwise it gives errors.

● –b(byte):- To extract the specific bytes ,you need to follow =b option with the list of byte numbers separated by comma .Range of bytes can also be specified
using hyphen.
List without ranges: cut –b 1,2,3 state.txt
List with ranges: cut –b 1,-3,5-7 state .txt
● -c(column):- To cut by character ,use the –c option. This selects the character
given to the –c option. This can be a list of numbers separated by comma or a range of numbers separated by hyphen.
Syntax: cut-c[(k)-(n)/(k);(n)/(n)].filename
K denotes starting position of character n denotes ending position of character in each line if there are separated by ‘_’.
● -f (field):- -c option is used for fixed length lines .To extract the useful information you need to cut by fields rather than columns. List of field numbers specified must be separated by comma. Ranges are not described here.

Syntax: -d”delimiter” –f(field number)file.txt

● - complement:- As the name suggests it complement the output .This option can be used in the combination with other option either –f or with –c.

● -output-delimiter :- By default the output delimiter is same as input delimiter that we specify as input delimiter that we specify in the cut with –d option. To change the output delimiter we use option output –delimiter=”delimiter”.

● --version:- This option is used to display the version of cut which is currently running on your system.

RESULT

Basic Linux Command is familiarised successfully

EXPERIMENT NO. 2

BASIC SYSTEM CALLS IN LINUX

AIM

To familiarise basic Linux system calls

BASIC LINUX SYSTEM CALLS

PID AND PPID

an executable code stored on a disk is called a program and a program loaded into memory and running is called a process. when a process starts it is given a unique number called process ID or PID that identifies that process to the system
In addition to a unique process ID each process is
assigned a process process ID or PPID. It to tells which process started the PPID is the PID of the processes parent

FORK ( )

A new process is created by the fork() system call. A new process may be created with fork() without a new program being run – the new sub – process simply continue to execute exactly the same program that the first (parent) process was running. It is one of the most widely used system calls under process management

EXEC ( )

A new program will start executing after a call to exec(). Running a new program does not require that a new process be created first. Any process may be created at anytime. The currently running program is immediately terminated.

GETPID( )

getPid stands for get the process ID. The function shall return the process ID of the calling process. The getpid() function shall always be successful and no return values is reserved to indicate error.

EXIT ( )

The exit() system call is used by a program to terminate it’s execution. The operating system reclaims resources that were used by the process after the exit() system call

WAIT ( )

A call wait() blocks the calling process until one of its child processes exits or a signal is recieved. After child process terminates parents continues its execution after wait system calls instruction

OPEN ( )
Its the system call to open a file. This system call just opens the file and inorder to execute read and write operation we need to call other system calls.
CLOSE ( )
This system calls closes an opened file

STAT ( )
To use the stat system call we need to include. Sys/stat.h header file. Stat() is used to get the status of a file. Its returns file attributes about an inode.

OPENDIR ( )
To use this system call one must include <dirent.h> header file. This function shall open a directory stream corresponding to the directory named by the dirname argument. Upon completion it returns a pointer to an object of type DIR. Its used with readdir() and closedir() to get the list of files names contained in the directory specified by the dirname

READDIR ( )
ReadDir() returns a pointer to a dirent structure describing the next entry in the stream associated with dir. A call to readdir() updates the st_atime (access time) field for the directory. If the successful, readdir() returns a pointer to a dirent structure describing the next directory entry in the directory stream. When it reaches the end of the datastream it returns NULL
.
READ( )
This system call opens the file in writing mode. We cannot edit the files with this system calls. Multiple process can execute the read() system call on the same file simultaneously

WRITE ( )
This system call opens the file in writing mode. We cannot edit the files with this system calls. Multiple processes can not execute the write() system call on the same file simultaneously

RESULT

Basic system calls in Linux successfully familiarised.

PROGRAM-A

```
#include<stdio.h>
#include<unistd.h>
void main()
{
  int n;
  printf("Enter the number : ");
  scanf("%d\n",&n);
  int a=fork();
  if(a>0)
  {
    printf("Odd Numbers : ");
    for(int i=0;i<=n;i++)
      if(i%2!=0)
        printf("%d",i);
  }
  else{
    printf("\nEven Numbers : ");
    for(int j=0;j<=n;j++)
      if(j%2==0)
        printf("%d",j);
    printf("\nChild process id : %d\nParent Process id : %d\n",getpid(),getppid());
  }
}
```

OUTPUT-A

```
Enter number : 8
Odd Numbers : 1 3 5 7
Even Numbers : 0 2 4 6 8
Child process id : 6632
Parent process id : 6631
```

EXPERIMENT NO .2(A)

IMPLEMENTATION OF FORK SYSTEM CALL

AIM

Write a c program to create 2 processes.The parent process must print odd numbers upto n and child process must print even numbers.Also print process id

ALGORITHM

1. Start
2. Read the input
3. Use fork() to create a process and assign the returned value to an integer a
4. If a>0 then
    4.1 for i from 0 to n,if i%2!=0
        Print i and i++
5.Otherwise
    5.1 for j from 0 to n if j%2==0
        Print j and j++
6. When a=0 ,print process id of child process using getpid() and print process id of parent process using getppid()
7. Stop

RESULT

The program was written and successfully executed

PROGRAM-B

```
#include <stdio.h>
#include <dirent.h>
void main()
{
  char c[100];
  DIR *d;
  struct dirent *dir;
  printf ("Enter path: ");
  scanf ("%s", c);
  d = opendir (c);
if (c) {

  while ((dir = readdir(d)) != NULL)
  {
    printf("%s\n",dir->d_name);
  }

}
closedir(d);
}
```

OUTPUT-B

```
Enter path : s4c9
ark.c
a.out
```

EXPERIMENT NO .2(B)

IMPLEMENTATION OF OPENDIR,READDIR,CLOSEDIR SYSTEM CALL

AIM

Write a C program to display the files in the given directory.

ALGORITHM

1. Start
2. Read the directory path from a user and store it in a character array.
3. Open directory stream using the path with opendir system call.
4. If the path of the directory is not null, go to step 5 else go to step 8.
5. Using the readdir system call, read the opened directory.
6. Display name of the directory content.
7. Repeat step 5 and 6 until readdir() system call returns null.
8. Close the directory stream using closedir system call.
9. Stop

RESULT

The program was written and successfully executed

PROGRAM-C

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

void main()
{
char c[100];
struct stat s;
printf("Enter path :");
scanf("%s" ,c);
if (stat(c,&s)==0)
  printf ("File size: %d", s.st_size);
else
  printf("Sorry,try again");
}
```

f1.txt

Said

OUTPUT-C

```
Enter path : f1.txt
File size : 5
```

EXPERIMENT NO .2(C)

IMPLEMENTATION OF STAT SYSTEM CALL

AIM

Write a C program to display the statistics of a given file

ALGORITHM

1. Start
2. Read path of file from user and store it in a character array c[100]
3. Declare a structure of stat type.
4. Call stat function with parameters file path and structure declared in step 3.
5. If the function returns 0, go to step 6.
   else goto step 7
6. Display the size of file and other properties about file and go to step 8.
7. Display message to show error.
8. Stop the program

RESULT

The program was written and successfully executed

PROGRAM-D

ex1.c

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

int main (int argc, char *argv[])
{
  printf ("PID of ex1.c = %d\n", getpid());
  char *args[] = { "Hello", "Welcome", "Home", NULL);
  execv ("./ex2" ,args);
  printf ("Back to ex1.c");
  return 0;
}
```

ex2.c

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

int main(int argc, char * argv[])
{
  printf ("We are in ex2.c\n");
  printf ("PID of ex2.c = %d\n", getpid());
  return 0;
}
```

OUTPUT-D

```
PID of ex1.c = 5962
We are in ex2.c
PID of ex2.c = 5962
```

EXPERIMENT NO .2(D)

IMPLEMENTATION OF EXEC SYSTEM CALL

AIM

Write a C program to familiarise exec() system call

ALGORITHM

1. Start
2. Print PID of the current process
3. Declare an argument array of string with.NULL as the last element
4. Pass the path name of the file and array of arguments to the exec() system call
5. The function replaces the current process with a new process given as argument and executes it.
6. After execution of the new process the program ends.
7.Stop

RESULT

The program was written and successfully executed

EXPERIMENT.NO.03

INTRODUCTION TO SHELL PROGRAMMING

AIM

To familiarise shell programming and able to implement programs

VI EDITOR

vi editor is create and edit text, files documents and programs.
There are two modes: i) command mode ii) insert mode

COMMANDS IN Vi

$vi<filename> : This command is used to start the vi editor
ESC i command : Used to insert the text before the current cursor position
ESC I command : Used to insert at the beginning of the current line
ESC o command : Used to insert a blank line below the current line
ESC O command: Used to insert a Blank Line above and allow insertion of contents
ESC h command : Used to move to the left of the text
ESC l command: Used to move to the right of the cursor
ESC j command : Used to move down a single line.
ESC k command : Used to move up a single line
ESC 0 command : Bring the cursor to the beginning of the same current line
ESC $: Bring the cursor to the end of the current line
ESC x command : Delete a character to the right of current cursor
ESC X command: Delete a character to the left of the current cursor
ESC dd command: To delete the current line

ESC w command : To save given text present in the file
ESC q! Command : To the quit the given text without saving
ESC wq command: Quits the vi editor after saving in the mentioned file

RESULT

Successfully familiarised shell programming and basic commands used .

SHELL PROGRAMS

AIM

To write programs in shell programming language and gain the necessary output

DESCRIPTION:

The activities of a shell are not restricted to command interpretation alone. The shell also has
Rudimentary programming features. When a group of commands has to be executed regularly,
they are stored in a file (with extension .sh). All such files are called shell scripts or shell programs.
Shell programs run in interpretive mode. The original UNIX came with the Bourne shell (sh) and it
is universal even today. Then came a plethora of shells offering new features. Two of them, C shell
(csh) and Korn shell (ksh) has been well accepted by the UNIX fraternity. Linux offers Bash shell
(bash) as a superior alternative to Bourne shell.

Preliminaries

1. Comments in shell script start with #. It can be placed anywhere in a line; the shell ignores
contents to its right. Comments are recommended but not mandatory.
2. Shell variables are loosely typed i.e. not declared. Their type depends on the value assigned.
Variables when used in an expression or output must be prefixed by $.
3. The read statement is shell's internal tool for making scripts interactive.
4. Output is displayed using echo statement. Any text should be within quotes. Escape sequence
should be used with –e option.
5. Commands are always enclosed with ` ` (back quotes).
6. Expressions are computed using the expr command. Arithmetic operators are + - * / %. Meta
characters * ( ) should be escaped with a \.
7. Multiple statements can be written in a single line separated by ;
8. The shell scripts are executed using the sh command (sh filename).

PROGRAM -A

```
# Program to find factorial of a number
echo -n "Enter number: "
read num
fact=1
while [ $num -ne 0 ]
do
fact=$(( $fact * $num ))
num=$(( $num - 1 ))
done
echo "Factorial is $fact"
```

OUTPUT

```
Enter the number: 5
Factorial is 120
```

EXPERIMENT NO.3(A)

FACTORIAL OF A NUMBER IN SHELL PROGRAM

AIM

Write a shell program to find factorial of a number

ALGORITHM

1. Start
2. Read the value of n.
3. .Declare fact and fact=0
4. If num!=0 then set fact=fact*num and num=num-1 and repeat the step 4
5. Display fact
6. Stop

RESULT

Successfully executed the program

PROGRAM -B

```
# Program to find the largest of N numbers
echo -n "Enter N numbers: "
read nums
largest=0
for num in $nums
do
if [ $num -gt $largest ]
then
largest=$num
fi
done
echo "Largest number is $largest"
```

OUTPUT

```
Enter N numbers: 3 5  6
Largest number is 6
```

EXPERIMENT NO.3(B)

LARGEST OF N NUMBERS IN SHELL PROGRAM

AIM

Write a shell program to find largest of n numbers

ALGORITHM

1. Start
2. Read n numbers in nums
3. Declare largest and set largest=0
4. For all num in nums
    4.1 If num>largest then set largest=num
5. Display largest
6. Stop

RESULT

Successfully executed the program

PROGRAM -C

```
# Program to check whether a number is odd or even
echo -n "Enter a number: "
read num
remainder=`expr $num % 2`
if [ $remainder -eq 0 ]
then
echo "$num is even"
else
echo "$num is odd"
fi
```

OUTPUT

```
Enter a number: 6
6 is even
```

EXPERIMENT NO.3(C)

ODD OR EVEN IN SHELL PROGRAM

AIM

Write a shell program to check if a number is odd or even

ALGORITHM

1. Start
2. Read the value of num
3. Set reminder=num%2
4. If remainder==0 then print num is even
   Otherwise print num is odd
5. Stop

RESULT

Successfully executed the program

PROGRAM -D

```
# Program to find sum of three numbers
echo -n "First Number is: "
read num1
echo -n "Second Number is: "
read num2
echo -n "Third Number is: "
read num3
sum=`expr $num1 + $num2 + $num3`
echo "Sum is $sum"
```

OUTPUT

```
First Number is: 1
Second Number is: 2
Third Number is: 4
Sum is 7
```

EXPERIMENT NO.3(D)

SUM OF THREE NUMBERS IN SHELL PROGRAM

AIM

Write a shell program to find sum of 3 numbers

ALGORITHM

1. Start
2. Read 3 numbers in num1,num2,num3
3. Declare sum
4. Set sum=num1+num2+num3
5. Display sum
6. Stop

RESULT

Successfully executed the program

PROGRAM -E

```
# Program to swap two variables
echo -n "Enter first number: "
read n1
echo -n "Enter second number: "
read n2
tmp=$n1
n1=$n2
n2=$tmp
echo -n “First No =$n1 and Second No=$n2”
```

OUTPUT

```
Enter first number: 3
Enter second number : 2
First No =2 and Second No=3
```

EXPERIMENT NO.3(E)

SWAP 2 VARIABLES IN SHELL PROGRAM

AIM

Write a shell program to swap 2 numbers

ALGORITHM

1. Start
2. Read 2 numbers in n1,n2
3.Declare tmp
4.Set tmp=n1
5.Set n1=n2
6.Set n2=tmp
7.Print n1,n2
8.Stop

RESULT

Successfully executed the program

PROGRAM -3F

```
echo-n ”Enter temperature in fahrenheit : ”
read f
c=$(( $(($f-32))*5/9 ))
echo ”Temperature in celsius= $c”
```

OUTPUT

```
Enter temperature in fahrenheit : 212
Temperature in celsius=100
```

EXPERIMENT NO.3(F)

CONVERT FAHRENHEIT TO CELSIUS IN SHELL PROGRAM

AIM

Write a shell program convert temperature from fahrenheit to celsius

ALGORITHM

1. Start
2. Read f
3.Set c= (f-32)*5/9
4. Display c
5. Stop

RESULT

Successfully executed the program

PROGRAM-A

```
#include<stdio.h>
void main()
{
  int n;
  int twt=0,tat=0;
  printf("Enter the number of process : ");
  scanf("%d",&n);
  int a[n][6];
  //0-process id ,1-at,2-bt,3-completion time,4-tat,5-waiting time
  for(int i=0;i<n;i++)
    {
      printf("\nProcess ID :"); scanf("%d",&a[i][0]);
      printf("A.T of Process.%d :",a[i][0]); scanf("%d",&a[i][1]);
      printf("B.T of Process.%d :",a[i][0]); scanf("%d",&a[i][2]);
    }
  a[0][3]=a[0][1]+a[0][2]; //completion time of 1st process
  a[0][4]=a[0][3]-a[0][1]; //tat of 1st process
  a[0][5]=a[0][4]-a[0][2]; //waiting time of 1st process

  for(int i=1;i<n;i++)
    {
      a[i][3]=a[i-1][3]+a[i][2]; //completion time of ith process
      a[i][4]=a[i][3]-a[i][1]; //tat of ith process
      a[i][5]=a[i][4]-a[i][2]; //waiting time of ith process
    }
   printf("\nP.No.\tAT\t  BT\t TAT\t WT");
    for (int i=0; i<n; i++)
    {
        printf("\n%d\t\t%d\t%d\t%d\t%d", a[i][0], a[i][1], a[i][2], a[i][4], a[i][5]);
        tat+=a[i][4];
        twt+=a[i][5];
    }
   printf ("\nAverage Turnaround time = %f", tat/(float)n);
   printf ("\nAverage Waiting time = %f", twt/(float)n);
}
```

EXPERIMENT NO .04

CPU SCHEDULING ALGORITHM

a. FCFS

AIM

Write a C Program to implement FCFS CPU scheduling algorithm

DESCRIPTION

FCFS schedules the process simply according to their arrival time.The job which comes first in the ready queue gets the CPU first.

ALGORITHM

1. Start
2. Get the number of processes to be inserted.
3. Declare the array size.
4. Get the arrival time and burst time for each process and store it in an array.
5. Set the burst time + arrival time of the first process as its completion time.
6. Calculate the completion time of the remaining process by adding its burst time and completion of the process before it and store it in the array.
7.Calculate the turnaround time for each process by subtracting arrival time from completion time.
8. calculate the waiting time for each process by subtracting burst time from turnaround time.
9. Calculate the average turnaround time and average waiting time.
10. Display the details such as Process number, arrival time, burst time, turn around time, waiting time, average waiting time, average turnaround time.
11. Stop

OUTPUT

Enter the number of process : 3

Process ID :1
A.T of Process.1 :0
B.T of Process.1 :5

Process ID :2
A.T of Process.2 :3
B.T of Process.2 :9

Process ID :3
A.T of Process.3 :6
B.T of Process.3 :6

P.No.	AT	BT	TAT	WT
1	0	5	5	0
2	3	9	11	2
3	6	6	14	8

Average Turnaround time = 10.000000
Average Waiting time = 3.333333

RESULT

Successfully implemented FCFS CPU scheduling.

PROGRAM-B

```
#include<stdio.h>

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

int main()
{
    int n;
    int i, j, temp, cmpt, min;
    float twt=0;
    float ttat=0;
    printf("\nEnter the number of processes: ");
    scanf("%d", &n);
    int a[n][5];
    printf("\nEnter the Arrival Time & Burst Time of each process: ");
    for (i=0; i<n; i++)
    {
        printf ("\nProcess ID: ");
        scanf ("%d", &a[i][0]);
        printf("A.T of Process.%d: ", a[i][0]);
        scanf("%d", &a[i][1]);
        printf("B.T of Process.%d: ", a[i][0]);
        scanf("%d", &a[i][2]);
    }

    for (i=0; i<n; i++)
    {
        for (j=0; j<n-i-1; j++)
        {
            if (a[j][1]>a[j+1][1])
            {

                swap(&a[j][0], &a[j+1][0]);
                swap(&a[j][1], &a[j+1][1]);
                swap(&a[j][2], &a[j+1][2]);
            }
```

b. SJF

AIM

Write a C Program to implement SJF CPU scheduling algorithm

DESCRIPTION

In SJFwith arrival time, the processes with least burst time which is present in the ready queue at the time when CPU is idle is taken for execution

ALGORITHM

1. Start the program.
2. Get the number of processes to be inserted,say n.
3. Declare an array with ‘n’ rows and five columns a[n][5].
4. Get the arrival time and burst time of each process and store at a[i][1] and a[i][2].
5. Swap the processes in the increasing order of their arrival time.
   If arrival time of some processes are same then swap those process in the increasing order of their burst time
6. Set all the time of 1st arrived process as it will execute no matter what
   Set a[0][3]=a[0][1],
   a[0][4]=a[0][2]-a[0][1],
   cmpt=a[0][4],
   twt=twt+a[0][3],
   ttat=ttat+a[0][4].

7.Starting from the second process, if the burst time is less than the selected process and arrival time is less than completion time,swap the process.

8. Set a[i][3]=cmpt-a[i][1]
   twt=twt+a[i][3]
   cmpt=cmpt+a[i][3]
   a[i][4]=cmpt-a[i][1]
   ttat= ttat+a[i][4]
9. Display each process and corresponding arrival time,turnaround time and waiting time.
10. Display average turnaround time and average waiting time.
11. Stop the program.

```
            if (a[j][1]==a[j+1][1] && a[j][2]>a[j+1][2])
            {

                swap(&a[j][0], &a[j+1][0]);
                swap(&a[j][1], &a[j+1][1]);
                swap(&a[j][2], &a[j+1][2]);
            }
        }
    }
    a[0][3]=a[0][1];
    a[0][4]=a[0][2]-a[0][1];
    cmpt = a[0][4];
    twt = twt + a[0][3];
    ttat = ttat + a[0][4];
    for (i=1; i<n; i++)
    {
        min = a[i][2];
        for (j=i+1; j<n; j++)
        {
            if (min>a[j][2] && a[j][1]<=cmpt)
            {
                min=a[j][2];
                swap(&a[i][0], &a[j][0]);
                swap(&a[i][1], &a[j][1]);
                swap(&a[i][2], &a[j][2]);
            }
        }
        a[i][3] = cmpt - a[i][1];
        twt += a[i][3];
        cmpt += a[i][2];
        a[i][4] = cmpt - a[i][1];
        ttat += a[i][4];
    }
    printf("\nP.No.\tAT\tBT\tTAT\tWT");
    for (i=0; i<n; i++)
    {
        printf("\n%d\t%d\t%d\t%d\t%d", a[i][0], a[i][1], a[i][2], a[i][4], a[i][3]);
    }
    printf ("\nAverage Turnaround time = %f", ttat/n);
    printf ("\nAverage Waiting time = %f", twt/n);
    return 0;
}
```

OUTPUT

Enter the number of processes: 5

Enter the Arrival Time & Burst Time of each process:
Process ID: 1
A.T of Process.1: 2
B.T of Process.1: 6

Process ID: 2
A.T of Process.2: 5
B.T of Process.2: 2

Process ID: 3
A.T of Process.3: 1
B.T of Process.3: 8

Process ID: 4
A.T of Process.4: 0
B.T of Process.4: 3

Process ID: 5
A.T of Process.5: 4
B.T of Process.5: 4

P.No.	AT	BT	TAT	WT
4	0	3	3	0
1	2	6	7	1
2	5	2	6	4
5	4	4	11	7
3	1	8	22	14

Average Turnaround time = 9.800000
Average Waiting time = 5.200000

RESULT

Successfully implemented SJF CPU scheduling.

PROGRAM-C

```
#include<stdio.h>
void swap (int *a, int *b)
{
	int temp;
	temp = *a;
	*a = *b;
	*b = temp;
}
int main()
{
	int n, cmpt;
	float twt = 0, ttat = 0;
	printf ("\nEnter how many processes: ");
	scanf ("%d", &n);
	int a[n][6];
	printf ("\n-Enter arrival time, burst time and priority of each process:-");
	for (int i=0; i<n; i++)
	{

		printf ("\nProcess ID: ");
		scanf ("%d", &a[i][0]);
		printf ("A.T of process.%d: ", a[i][0]);
		scanf ("%d", &a[i][1]);
		printf ("B.T of process.%d: ", a[i][0]);
		scanf ("%d", &a[i][2]);
		printf ("Priority of process.%d: ", a[i][0]);
		scanf ("%d", &a[i][5]);
	}
	for (int i=0; i<n; i++)
	{
	   for (int j=0; j<n-i-1; j++)
	   {
		if (a[j][1]>a[j+1][1])
		{
			swap (&a[j][0], &a[j+1][0]);
			swap (&a[j][1], &a[j+1][1]);
			swap (&a[j][2], &a[j+1][2]);
			swap (&a[j][5], &a[j+1][5]);
		}
```

c. PRIORITY

AIM

Write a C Program to implement PRIORITY CPU scheduling algorithm

DESCRIPTION

In Priority scheduling with arrival time and non preemptive , the processes with most priority which is present in the ready queue at the time when CPU is idle is taken for execution

ALGORITHM

1. Start the program.
2. Get the number of processes to be inserted, let it be n.
3. Declare an array with n rows and 6 columns,a[n][6].
4. Get arrival time, burst time and the priority from each process and store at a[i][1],a[i][2] and a[i][3] respectively.
5. Swap the processes in the increasing order of their arrival time.
   If processes arrival time is same then swap them according to the decreasing priority
6. Set the time of first arrived process since it will execute 1st
   Set a[0][3]=a[0][1]
   a[0][4]=a[0][2]-a[0][1],
   cmpt=a[0][4],
   twt=twt+a[0][3],
   ttat=ttat+a[0][4].
7. Starting from the second process, if the burst time is less than the selected process and arrival time is less than completion time, swap the processes.
8. Set a[i][3]=cmpt-a[i][1]
   twt=twt+a[i][3]
   cmpt=cmpt+a[i][2]
   a[i][4]=cmpt-a[i][1]
   ttat=ttat+a[i][4]
9. Display each process and corresponding arrival time, burst time, turn around time and waiting time.
10. Display average turnaround time and average waiting time.
11. Stop

```
            if (a[j][1]==a[j+1][1] && a[j][5]>a[j+1][5])
            {
                swap (&a[j][0], &a[j+1][0]);
                swap (&a[j][1], &a[j+1][1]);
                swap (&a[j][2], &a[j+1][2]);
                swap (&a[j][5], &a[j+1][5]);
            }
        }
    }
    a[0][3] = a[0][1];
    a[0][4] = a[0][2] - a[0][1];
    cmpt = a[0][4];
    twt+=a[0][3];
    ttat+=a[0][4];
    for (int i=1; i<n; i++)
    {
        int min = a[i][5];
        for (int j=i+1; j<n; j++)
        {
            if (min>a[j][5] && a[j][1]<=cmpt)
            {
                min = a[j][5];
                swap (&a[i][0], &a[j][0]);
                swap (&a[i][1], &a[j][1]);
                swap (&a[i][2], &a[j][2]);
                swap (&a[i][5], &a[j][5]);
            }
        }
        a[i][3] = cmpt - a[i][1];
        twt+=a[i][3];
        cmpt+=a[i][2];
        a[i][4] = cmpt - a[i][1];
        ttat+=a[i][4];
    }
    printf("\nP.No.\tAT\tBT\tP.R\tTAT\tWT");
    for (int i=0; i<n; i++)
    {
        printf("\n%d\t%d\t%d\t%d\t%d\t%d", a[i][0], a[i][1], a[i][2], a[i][5], a[i][4], a[i][3]);
    }
    printf ("\nAverage Turnaround time = %f", ttat/n);
    printf ("\nAverage Waiting time = %f", twt/n);
}
```

OUTPUT

Enter how many processes: 5

-Enter arrival time, burst time and priority of each process:-
Process ID: 1
A.T of process.1: 0
B.T of process.1: 3
Priority of process.1: 3

Process ID: 2
A.T of process.2: 1
B.T of process.2: 6
Priority of process.2: 4

Process ID: 3
A.T of process.3: 3
B.T of process.3: 1
Priority of process.3: 9

Process ID: 4
A.T of process.4: 2
B.T of process.4: 2
Priority of process.4: 7

Process ID: 5
A.T of process.5: 4
B.T of process.5: 4
Priority of process.5: 8

P.No.	AT	BT	P.R	TAT	WT
1	0	3	3	3	0
2	1	6	4	8	2
4	2	2	7	9	7
5	4	4	8	11	7
3	3	1	9	13	12

Average Turnaround time = 8.800000
Average Waiting time = 5.600000

RESULT

Successfully implemented PRIORITY CPU scheduling.

PROGRAM-D

```
#include<stdio.h>
int q[100];
int f = -1;
int r = -1;
void insert (int n)
{
	if (f==-1)
		f=0;
	r = r+1;
	q[r] = n;
}
int delete ()
{
	int n;
	n = q[f];
	f = f+1;
	return n;
}
void main()
{
	int p, tq, n, i, t=0;
	float twt = 0, ttat = 0;
	printf ("\nEnter how many processes: ");
	scanf ("%d", &n);
	int a[n][5], bt[10], exist[10] = {0};
	printf ("\n Enter arrival time & burst time of each process:");
	for (i=0; i<n; i++)
	{

		printf ("\nProcess ID: ");
		scanf ("%d", &a[i][0]);
		printf ("\nA.T of process %d: ", a[i][0]);
		scanf ("%d", &a[i][1]);
		printf ("\nB.T of process %d: ", a[i][0]);
		scanf ("%d", &a[i][2]);
		bt[i] = a[i][2];
	}
	printf ("\nEnter time quantum: ");
	scanf ("%d", &tq);
	insert (0);
	exist[0] = 1;
```

d. Round Robin CPU Scheduling

AIM

Write a C Program to implement Round Robin CPU scheduling algorithm

DESCRIPTION

In RR, each process is assigned the CPU according to their arrival time from the ready queue for a certain amount of time known as time quantum , if that process execution is completed by that time then that process is terminated else will go back to the ready queue and wait for the next turn to complete .

ALGORITHM

1. Start
2. Create an array q with the front(f) and rear(r) pointers initialised to -1.
3. Get the number of processes, n and store it in an array a[n][5].Set t=0.
4. Get the arrival time and bus time for each process and store it at the first and the second column of a. Also store burst time in another array.
5. Get time quantum for processes, let it be tq.
6. Insert the first process to the queue and make exist[0] as one.
7.If f is less than or equal to r, remove a process from q[] and store at p. Go to step 8 to 12.
8. if(a[p][2]>=tq)
    a[p][2]=a[p][2]-tq
    t=t+tq
9. Else
    t=t+a[p][2]
    a[p][2]=0
10. For each process from i=0 to i<n
    If exist[i]==0 and a[i][1]<=t then
        insert(i) to the q[]
        Set exist[i] as 1.
11. If a[p][2] ==0 set
    a[p][4] =t-a[p][1]
    a[p][3]=a[p][4]-bt[p]
    ttat=ttat+ a[p][4]
    twt=twt+a[p][3]
12. Else
    Insert p to the q[]
13. Display each process and corresponding arrival time, burst time, turn around time and waiting time.

```
        while (f<=r)
        {
                p = delete ();
                if (a[p][2]>=tq)
                {
                        a[p][2]-=tq;
                        t+=tq;
                }
                else
                {
                        t+=a[p][2];
                        a[p][2] = 0;
                }
                for (i=0; i<n; i++)
                {
                        if (exist[i]==0 && a[i][1]<=t)
                        {
                                insert (i);
                                exist[i]=1;
                        }
                }
                if (a[p][2]==0)
                {
                        a[p][4]=t-a[p][1];
                        a[p][3]=a[p][4]-bt[p];
                        ttat+=a[p][4];
                        twt+=a[p][3];
                }
                else
                {
                        insert (p);
                }
        }
        printf("\nP.No.\tAT\tBT\tTAT\tWT");
        for (i=0; i<n; i++)
        {
            printf("\n%d\t%d\t%d\t%d\t%d", a[i][0], a[i][1], bt[i], a[i][4], a[i][3]);
        }
        printf ("\nAverage Turnaround time = %f", ttat/n);
        printf ("\nAverage Waiting time = %f", twt/n);
}
```

14. Display average time by displaying value obtained by twt/n.
15. Similarly display ttat/n As average turnaround time.
16. Stop

OUTPUT

Enter how many processes: 4

Enter arrival time & burst time of each process:

Process ID: 1
A.T of process 1: 0
B.T of process 1: 5

Process ID: 2
A.T of process 2: 1
B.T of process 2: 4

Process ID: 3
A.T of process 3: 2
B.T of process 3: 2

Process ID: 4
A.T of process 4: 3
B.T of process 4: 1

Enter time quantum: 2

P.No.	AT	BT	TAT	WT
1	0	5	12	7
2	1	4	10	6
3	2	2	4	2
4	3	1	6	5

Average Turnaround time = 8.000000
Average Waiting time = 5.000000

RESULT

Successfully implemented Round Robin CPU scheduling.

PROGRAM -1 (WRITER PROCESS)

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

void main()
{
  char str[100];
  int shmid=shmget((key_t)2366,1024,0666|IPC_CREAT);
  printf("Key of shared memory is %d\n",shmid);
  void* shm=shmat(shmid,NULL,0);
  printf("Process is attached at %p \n ",shm);
  printf("Enter data : ");
  scanf("%s",str); // read(0,str,100);
  strcpy(shm,str);
  printf("\nValue written by writer process : %s",(char*)shm);
}
```

OUTPUT-1

```
Key of shared memory is 0
Process is attached at 0x7fb8cef0f000
 Enter data : hai
Value written by writer process : hai
```

EXPERIMENT NO .05

IPC USING SHARED MEMORY

AIM

Write a C Program to implement IPC using shared memory (Readers-writers problem)

ALGORITHM

WRITER PROCESS

1. Start
2. Attach necessary libraries and declare str[100] character array
3. Create the shared memory segment or use an already created shared memory segment using shmget() system call
4. Attach the Writer process to the created memory segment using shmat() system call
5. Read data from user using either read() system call or using scanf() to the str[100]
6. Write the read input from str[100] to the shared segment using strcpy() function
7. Stop writer process

READER PROCESS

1. Start
2. Attach necessary libraries and declare str[100] character array
3. Use the already created shared memory segment by writer process using shmget() system call
4. Attach the reader process to the memory segment using shmat() system call
5. Print the data present in the shared memory
6. Stop the reader process

PROGRAM -2 (READER PROCESS)

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

void main()
{
  char str[100];
  int shmid=shmget((key_t)2366,1024,0666);
  printf("Key of shared memory is %d\n",shmid);
  void* shm=shmat(shmid,NULL,0);
  printf("Process is attached at %p \n ",shm);

  printf("\nValue read by reader process : %s",(char*)shm);
}
```

OUTPUT-2

```
Key of shared memory is 0
Process is attached at 0x7f42560db000
Value written by writer process : hai
```

RESULT

Successfully implemented IPC using shared memory

PROGRAM

```
#include<stdio.h>
#define BS 5
int buf[BS],in=0,out=0;
int m=1,f=0,e=BS; //m=semaphore, f=no of full blocks,  e=empty blocks

int wait(int s)
{
  while(s<=0);
  s--;
  return s;
}
int signal(int s)
{
  s++;
  return s;
}

void producer()
{
  int itemp;
  printf("Enter an integer value which is produced :");
  scanf("%d",&itemp);

  m=wait(m);
  e=wait(e);
  buf[in]=itemp;
  in=(in+1)%BS;
  printf("Producer PRODUCED item:%d\n",itemp);
  f=signal(f);
  m=signal(m);      }

void consumer()
{
  int itemc;
  m=wait(m);
  f=wait(f);
  itemc=buf[out];
  out=(out+1)%BS;
  printf("Consumer CONSUMED item:%d\n",itemc);
  e=signal(e);
  m=signal(m);  }
```

EXPERIMENT NO .06

PRODUCER-CONSUMER PROBLEM USING SEMAPHORE

AIM

Write a C Program to solve producer consumer problem using semaphore.

ALGORITHM

1. Start
2. Initialize BS=5 ,buf[Bs],out=in=0,e=BS,f=0 and m=1
3. Create a wait() function that decrements the value of an integer if its value is positive otherwise make a busy waiting and returns an integer variable
    3.1 if s<=0 then
        Go to line 3.1
    3.2 Set s–
    3.3 Return s
4. Create a signal() function that increments the value of an integer and returns that integer variable
    3.1 Set s++
    3.2 Return s
5. Create a producer() function , which produces an itemp and then add it to the buffer only if semaphore m is available and e!=0
    5.1 Decrement value of m and e using wait()
    5.2 Declare itemp
    5.3 Read item from user and store it in itemp
    5.4 Set buf[in]=itemp and in=(in+1)%BS and print item produced
    5.5 Increment value of f and m using signal()
6. Create a consumer() function , which consumes an itemc from buf[] in out position only if semaphore m is available and f!=0
    6.1 Decrement value of m and f using wait()
    6.2 Declare itemc
    6.3 Set items=buf[out] and out=(out+1)%BS
    6.4 print item consumed
    6.5 Increment value of e and m using signal()
7. Inside main function print “1.Produce an item 2.Consumes an item 3.Exit”
8. Declare choice
9. Read choice from user
10. If choice==1
    10.1 if m==1 && e!=0 then call produce() to produce an item
        Otherwise print “Producer is waiting..”

```
void main()
{
  printf("---PRODUCER-CONSUMER---\n");
  int choice;
  do{
    printf("\n***OPERATIONS***\n");
    printf("1.Produce an item\n2.Consume an item\n3.Exit\n");
    printf("Enter your choice : ");
    scanf("%d",&choice);
    switch(choice){
      case 1:
        if(m==1 && e!=0)
          producer();
        else
          printf("\nProducer is waiting....");
        break;
      case 2:
        if(m==1 && f!=0)
          consumer();
        else
          printf("\nConsumer is waiting....");
        break;
      case 3:
        printf("\nThank you\nExiting....");
        break;
      default:
        printf("Enter a valid choice.!!");
    }
  }while(choice!=3);
}
```

OUTPUT

```
---PRODUCER-CONSUMER---

***OPERATIONS***
1.Produce an item
2.Consume an item
3.Exit
Enter your choice : 1
Enter an integer value which is produced :100
Producer PRODUCED item:100
```

11. If choice==2
    11.1 if m==1 && f!=0 then call consumer() to consume an item
         Otherwise print “consumer is waiting.”
12. If choice ==3 then print exiting and go to line 15
13. If choice is not 1,2,3 print invalid choice
14. If choice!=3 go to line 9
15. Stop

OPERATIONS
1.Produce an item
2.Consume an item
3.Exit
Enter your choice : 1
Enter an integer value which is produced :200
Producer PRODUCED item:200

OPERATIONS
1.Produce an item
2.Consume an item
3.Exit
Enter your choice : 2
Consumer CONSUMED item:100

OPERATIONS
1.Produce an item
2.Consume an item
3.Exit
Enter your choice : 2
Consumer CONSUMED item:200

OPERATIONS
1.Produce an item
2.Consume an item
3.Exit
Enter your choice : 2

Consumer is waiting....
OPERATIONS
1.Produce an item
2.Consume an item
3.Exit
Enter your choice : 3

Thank you
Exiting....

RESULT

Successfully solved producer consumer problem using semaphore.

PROGRAM-A

```
#include<stdio.h>

void main()
{
  int nb,nf,temp;
  printf("Enter the no of blocks : ");
  scanf("%d",&nb);
  printf("Enter the no of files : ");
  scanf("%d",&nf);

  int frag[nf+1],b[nb+1],f[nf+1];
  int bf[nb+1],ff[nf+1];

  printf("Enter the size of the blocks:\n");
  for(int i=1;i<=nb;i++)
    {
      printf("Block.%d : ",i);
      scanf("%d",&b[i]);
    }
  printf("Enter the size of the files:\n");
  for(int i=1;i<=nf;i++)
    {
      printf("File.%d : ",i);
      scanf("%d",&f[i]);
    }

  for(int i=1;i<=nf;i++)
    {
      temp=-1;
      for(int j=1;j<=nb;j++)
        {
          if(bf[j]!=1)
          {
            if(b[j]>=f[i]){
              temp=j;
              break;
              }
          }
        }
      if(temp!=-1){
        frag[i]=b[temp]-f[i];
```

EXPERIMENT NO .07

MEMORY ALLOCATION METHODS

a. First Fit

AIM

Write a C Program to implement First Fit memory allocation method for fixed partition

ALGORITHM

1. Start
2. Declare nb,nf,temp
3. Read no of blocks in nb and no of files in nf
4. Declare frag[nf+1], b[nb+1] , f[nf+1], bf[nb+1] , ff[nf+1] //we are using + 1 because we are starting from 1 to nb not 0
5. Read the size of each block and store it in b[i] where i=1 to nb
6. Read the size of each file and store it in f[i] where i=1 to nf
7. Now take each file and assign temp=-1 and check if its size is less or equal to the size of each blocks and that block is not already assigned if yes then assign that block no to temp and break out of loop and set frag[i] to difference b/w block and file , set ff[i] to temp and bf[i] to 1 Otherwise repeat the process from each blocks(nb times) and at last if temp==-1 then set frag[i] and ff[i] to -1
    - 7.1 for i from 1 to nf
        - 7.1.1 Set temp=-1
        - 7.1.2 for j from 1 to nb
            - If bf[j]!=1 and b[j]>=f[i] then set temp=j and break
        - 7.1.3 if temp!=-1 then
            - Set frag[i]=b[temp]-f[i]
            - Set ff[i] = temp
            - Set bf[temp]=1
        - 7.1.4 Otherwise
            - Set frag[i]=-1 and ff[i]=-1
8. Display each files number,size ,the block they are assigned,its size and fragmentation
    - 8.1 for i from 1 to nf
        - 8.1.1 Display i,f[i],ff[i],b[ff[i]],frag[i]
9. Stop

```
        ff[i]=temp;
        bf[temp]=1;
      }
      else{
        frag[i]=-1;
        ff[i]=-1;
      }
    }
  printf("File_no\tFile_Size\tBlock_no\tBlock_size\tFragment\n");
  for(int i=1;i<=nf;i++)
    printf("%d\t\t%d\t\t\t%d\t\t\t%d\t\t\t%d\n",i,f[i],ff[i],b[ff[i]],frag[i]);

}
```

OUTPUT

```
Enter the no of blocks : 5
Enter the no of files : 4
Enter the size of the blocks:
Block.1 : 300
Block.2 : 200
Block.3 : 100
Block.4 : 400
Block.5 : 150
Enter the size of the files:
File.1 : 100
File.2 : 300
File.3 : 150
File.4 : 350
File_no  File_Size   Block_no    Block_size  Fragment
1          100          1             300          200
2          300          4             400          100
3          150          2             200          50
4          350         -1             0            -1
```

RESULT

Successfully implemented First Fit memory allocation method for fixed partition

PROGRAM-B

```
#include<stdio.h>

void main()
{
  int nb,nf,temp;
  printf("Enter the no of blocks : ");
  scanf("%d",&nb);
  printf("Enter the no of files : ");
  scanf("%d",&nf);

  int frag[nf+1],b[nb+1],f[nf+1];
  int bf[nb+1],ff[nf+1];

  printf("Enter the size of the blocks:\n");
  for(int i=1;i<=nb;i++)
    {
      printf("Block.%d : ",i);
      scanf("%d",&b[i]);
    }
  printf("Enter the size of the files:\n");
  for(int i=1;i<=nf;i++)
    {
      printf("File.%d : ",i);
      scanf("%d",&f[i]);
    }

  for(int i=1;i<=nf;i++)
    {
      temp=-1;
      for(int j=1;j<=nb;j++)
        {
          if(bf[j]!=1)
          {
            if(b[j]>=f[i]){
              if(temp==-1 || (b[j]-f[i])<(b[temp]-f[i]))
                temp=j;
              }
          }
        }
      if(temp!=-1){
        frag[i]=b[temp]-f[i];
```

b. Best Fit

AIM

Write a C Program to implement Best Fit memory allocation method for fixed partition

ALGORITHM

1. Start
2. Declare nb,nf,temp
3. Read no of blocks in nb and no of files in nf
4. Declare frag[nf+1], b[nb+1] , f[nf+1], bf[nb+1] , ff[nf+1] //we are using + 1 because we are starting from 1 to nb not 0
5. Read the size of each block and store it in b[i] where i=1 to nb
6. Read the size of each file and store it in f[i] where i=1 to nf
7. Now take each file and assign temp=-1 and check if that block is not already assigned if not then check if file size is less than or equal to block size if yes then check if temp is still -1 or this block size difference with the file is **less** than old block assigned in temp if yes make this block as temp and repeat the process of checking with the same file with all the blocks

    Check if temp not equal to -1 then set frag[i] to difference b/w block and file , set ff[i] to temp and bf[i] to 1 Otherwise set frag[i] and ff[i] to -1

    7.1 for i from 1 to nf

        7.1.1 Set temp=-1

        7.1.2 for j from 1 to nb

            If bf[j]!=1 and b[j]>=f[i] then

                If temp==-1 or b[j]-f[i]<b[temp]-f[i] then

                    set temp=j and repeat from 7.1.2

        7.1.3 if temp!=-1 then

            Set frag[i]=b[temp]-f[i]

            Set ff[i] = temp

            Set bf[temp]=1

        7.1.4 Otherwise

            Set frag[i]=-1 and ff[i]=-1

8. Display each files number,size ,the block they are assigned,its size and fragmentation

    8.1 for i from 1 to nf

        8.1.1 Display i,f[i],ff[i],b[ff[i]],frag[i]

9. Stop

```
          ff[i]=temp;
        bf[temp]=1;
      }
      else{
        frag[i]=-1;
        ff[i]=-1;
      }
    }
  printf("File_no\tFile_Size\tBlock_no\tBlock_size\tFragment\n");
  for(int i=1;i<=nf;i++)
    printf("%d\t\t%d\t\t\t%d\t\t\t%d\t\t\t%d\n",i,f[i],ff[i],b[ff[i]],frag[i]);
}
```

OUTPUT

```
Enter the no of blocks : 5
Enter the no of files : 4
Enter the size of the blocks:
Block.1 : 300
Block.2 : 200
Block.3 : 100
Block.4 : 400
Block.5 : 150
Enter the size of the files:
File.1 : 100
File.2 : 300
File.3 : 150
File.4 : 350
File_no   File_Size   Block_no    Block_size  Fragment
1           100          3             100          0
2           300          1             300          0
3           150          5             150          0
4           350          4             400          50
```

RESULT

Successfully implemented Best Fit memory allocation method for fixed partition

PROGRAM-C

```
#include<stdio.h>

void main()
{
  int nb,nf,temp;
  printf("Enter the no of blocks : ");
  scanf("%d",&nb);
  printf("Enter the no of files : ");
  scanf("%d",&nf);

  int frag[nf+1],b[nb+1],f[nf+1];
  int bf[nb+1],ff[nf+1];

  printf("Enter the size of the blocks:\n");
  for(int i=1;i<=nb;i++)
    {
      printf("Block.%d : ",i);
      scanf("%d",&b[i]);
    }
  printf("Enter the size of the files:\n");
  for(int i=1;i<=nf;i++)
    {
      printf("File.%d : ",i);
      scanf("%d",&f[i]);
    }

  for(int i=1;i<=nf;i++)
    {
      temp=-1;
      for(int j=1;j<=nb;j++)
        {
          if(bf[j]!=1)
          {
            if(b[j]>=f[i]){
              if(temp==-1 || (b[j]-f[i])>(b[temp]-f[i]))
                temp=j;
              }
          }
        }
      if(temp!=-1){
        frag[i]=b[temp]-f[i];
```

c. Worst Fit

AIM

Write a C Program to implement Worst Fit memory allocation method for fixed partition

ALGORITHM

1. Start
2. Declare nb,nf,temp
3. Read no of blocks in nb and no of files in nf
4. Declare frag[nf+1], b[nb+1] , f[nf+1], bf[nb+1] , ff[nf+1] //we are using + 1 because we are starting from 1 to nb not 0
5. Read the size of each block and store it in b[i] where i=1 to nb
6. Read the size of each file and store it in f[i] where i=1 to nf
7. Now take each file and assign temp=-1 and check if that block is not already assigned if not then check if file size is less than or equal to block size if yes then check if temp is still -1 or this block size difference with the file is **greater** than old block assigned in temp if yes make this block as temp and repeat the process of checking with the same file with all the blocks

    Check if temp not equal to -1 then set frag[i] to difference b/w block and file , set ff[i] to temp and bf[i] to 1 Otherwise set frag[i] and ff[i] to -1

    7.1 for i from 1 to nf

        7.1.1 Set temp=-1

        7.1.2 for j from 1 to nb

            If bf[j]!=1 and b[j]>=f[i] then

                If temp==-1 or b[j]-f[i]>b[temp]-f[i] then

                    set temp=j and repeat from 7.1.2

        7.1.3 if temp!=-1 then

            Set frag[i]=b[temp]-f[i]

            Set ff[i] = temp

            Set bf[temp]=1

        7.1.4 Otherwise

            Set frag[i]=-1 and ff[i]=-1

8. Display each files number,size ,the block they are assigned,its size and fragmentation

    8.1 for i from 1 to nf

        8.1.1 Display i,f[i],ff[i],b[ff[i]],frag[i]

9. Stop

```
          ff[i]=temp;
        bf[temp]=1;
      }
      else{
        frag[i]=-1;
        ff[i]=-1;
      }
    }
  printf("File_no\tFile_Size\tBlock_no\tBlock_size\tFragment\n");
  for(int i=1;i<=nf;i++)
    printf("%d\t\t%d\t\t\t%d\t\t\t%d\t\t\t%d\n",i,f[i],ff[i],b[ff[i]],frag[i]);
}
```

OUTPUT

```
Enter the no of blocks : 5
Enter the no of files : 4
Enter the size of the blocks:
Block.1 : 300
Block.2 : 200
Block.3 : 100
Block.4 : 400
Block.5 : 150
Enter the size of the files:
File.1 : 100
File.2 : 300
File.3 : 150
File.4 : 350
File_no  File_Size   Block_no    Block_size  Fragment
1          100          4           400          300
2          300          1           300          0
3          150          2           200          50
4          350          -1          0            -1
```

RESULT

Successfully implemented Worst Fit memory allocation method for fixed partition

PROGRAM-A

```
#include <stdio.h>

void FIFO(int pg[], int n, int frame)
{
    int pagefault = 0, hit = 0, k = 0, flag;
    int temp[frame];
    printf("\nValues\t");

    for (int i = 1; i <= frame; i++)
        printf("Frame%d\t", i);

    printf("\n");
    for (int i = 0; i < frame; i++)
    {
        temp[i] = -1;
    }

    for (int i = 0; i < n; i++)
    {
        flag= 0;
        printf("\n%d =>\t", pg[i]);
        for (int j = 0; j < frame; j++)
        {
            if (temp[j] == pg[i])
          {      flag= 1;
                 hit++;
                printf("\tHit!!!");
            }
        }
        if (flag == 0)
        {
            temp[k] = pg[i];
            k = (k + 1) % frame;
            pagefault++;
            for (int j = 0; j < frame; j++)
            {
                if (temp[j] != -1)
                    printf("%d\t", temp[j]);
            }
        }
    }
```

EXPERIMENT NO .08

PAGE REPLACEMENT ALGORITHMS

a. FIFO

AIM

Write a C Program to implement FIFO page replacement algorithm

ALGORITHM

1. Start
2. Declare n, frame
3. Read no of sequences in n and no of frames in frame
4.Declare pg[n]
5. Read each request and store it in pg[i]
6. Use another function FIFO and pass pg[],n,frame as argument
//Inside FIFO function
7.Declare pagefault,hit,k,flag,temp[frame]
8. Initialise pagefault=0,hit=0,k=0
9. Add -1 to all entries in temp[] indicating initially all frames are free
10. Now take each request ie pg[i] then set flag=0 then check if pg[i] is already present in temp if yes set flag to 1 , hit++ and print “Hit” otherwise set temp[k] to pg[i] where k denotes the frame in which oldest request is present and set k to k+1%frame and pagefault++ and then print all entries of temp which are not -1
    10.1 for i from 0 to n-1
        10.1.1 Set flag=0 and print pg[i]
        10.1.2 for j from 0 to frame-1
            If temp[j]==pg[i] then set hit++,flag=1 and print hit
        10.1.3 if flag==0 then
            Set temp[k]=pg[i] and k=(k+1)%frame
            For j from 0 to frame-1
                If temp[j]!=-1 then print temp[j]
11. Display Total page fault =pagefault
12. Display total page hit as hit
13. Display hit ratio as hit/n
14. Stop

```
    printf("\nTotal Page Faults: %d\n", pagefault);
    printf("\nTotal Page hits = %d\n", hit);
    printf("Hit ratio = %f",hit/(float)n);
}
void main()
{
    int n, frame;
    printf("Enter the Length of sequence: ");
    scanf("%d", &n);
    int pg[n];
    printf("Enter the Sequence: ");
    for (int i = 0; i < n; i++)
        scanf("%d", &pg[i]);
    printf("Enter the number of frames: ");
    scanf("%d", &frame);
    FIFO(pg,n,frame);
}
```

OUTPUT

```
Enter the Length of sequence: 15
Enter the Sequence: 7 0 1 2 0 3 0 4 2 3 0 3 1 2 0
Enter the number of frames: 3
Values  Frame1  Frame2  Frame3
7 =>    7
0 =>    7   0
1 =>    7   0   1
2 =>    2   0   1
0 =>        Hit!!!
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 =>        Hit!!!
1 =>    0   1   3
2 =>    0   1   2
0 =>        Hit!!!

Total Page Faults: 12
Total Page hits = 3
Hit ratio = 0.2
```

RESULT

Successfully implemented FIFO page replacement algorithm

PROGRAM-B

```
#include <stdio.h>

int findLRU(int time[], int n)
{
    int minimum = time[0], pos = 0;
    for (int i = 1; i < n; i++) {
        if (time[i] < minimum)
        {
            minimum = time[i];
            pos = i;
        }
    }
    return pos;
}

void LRU(int pg[], int n, int frame)
{
    int temp[frame], counter = 0, time[10], flag1, flag2,pos;
    int hit=0, pagefault = 0;
    printf("\nValues\t");

    for (int i = 1; i <= frame; i++)
        printf("Frame%d\t", i);

    printf("\n");
    for (int i = 0; i < frame; i++)
        temp[i] = -1;

    for (int i = 0; i < n; i++)
    {
        flag1 = flag2 = 0;
        printf("%d =>\t", pg[i]);
        for (int j = 0; j < frame; j++)
        {
            if (temp[j] == pg[i])
            {
                counter++;
                time[j] = counter;
                flag1 = flag2 = 1;
                hit++;
                printf("\tHit!!!\n");
```

b. LRU

AIM

Write a C Program to implement LRU page replacement algorithm

ALGORITHM

1. Start
2. Declare n, frame
3. Read no of sequences in n and no of frames in frame
4.Declare pg[n]
5. Read each request and store it in pg[i]
6. Use another function LRU and pass pg[],n,frame as argument
//Inside LRU function
7.Declare pagefault,hit, temp[frame],counter,time[frame],flag1.flag2,pos
8. Initialise pagefault=0,hit=0,counter=0
9. Add -1 to all entries in temp[] indicating initially all frames are free
10. Now take each request as pg[i] and set flage1=flage2=0 and check if pg[i] already in temp if yes set counter++,time[j]=counter,flag1=flag2=1,hit++ and break
Otherwise there are 2 options

If flag1==0 and free frame exist then set temp[j]=pg[i] counter++,time[j]=counter , flag2=1 ,pagefault++ and print each temp[k] which is not -1 and break

If flag2 still 0 means no free frame then first find least recently used frame using a function findLRU and store it in pos and set temp[pos]=pg[i], counter++, time[pos]=counter, pagefault++ and print each temp[k] which is not -1

10.1 for i from 0 to n-1

10.1.1 Set flag1=flag2=0 and print pg[i]

10.1.2 for j from 0 to frame-1

If temp[j]==pg[i] then set counter++ ,time[j]=counter hit++ ,flag1=flag2=1 and break

10.1.3 if flag1==0 then

For j from 0 to frame-1

If temp[j]==-1 then set counter++ ,time[j]=counter pagefault++ ,flag2=1 and print temp[k] where k=0 to frame-1 and temp[k]!=-1 and break

```
            break;
        }
    }
    if (flag1 == 0)
    {
        for (int j = 0; j < frame; j++)
        {
            if (temp[j] == -1)
            {
                counter++;
                pagefault++;
                temp[j] = pg[i];
                time[j] = counter;
                flag2 = 1;
                for (int k = 0; k < frame; k++)
                {
                    if (temp[k] != -1)
                        printf("%d\t", temp[k]);
                }
                printf("\n");
                break;
            }
        }
    }
    if (flag2 == 0)
    {
        pos = findLRU(time, frame);
        counter++;
        pagefault++;
        temp[pos] = pg[i];
        time[pos] = counter;
        for (int j = 0; j < frame; j++)
        {
            if (temp[j] != -1)
                printf("%d\t", temp[j]);
        }
        printf("\n");
    }
}
printf("\nTotal Page Faults = %d\n", pagefault);
printf("Total Page hits = %d\n", hit);
printf("Hit ratio = %f",hit/(float)n);
}
```

10.1.4 if flag2==0 then find LRU frame using findLRU()

//inside findLRU()

For i from 0 to n-1

If time[i]<minimum set minimum=time[i] and pos=i

Set temp[pos]=pg[i] , counter++,pagefault++,time[pos]=counter

Print each temp[k] where k =0 to frame-1 and temp[k]!=-1

11. Display Total page fault =pagefault
12. Display total page hit as hit
13. Display hit ratio as hit/n
14. Stop

```
void main()
{
    int n, frame;
    printf("Enter the Length of sequence: ");
    scanf("%d", &n);
    int pg[n];
    printf("Enter the Sequence: ");
    for (int i = 0; i < n; i++)
        scanf("%d", &pg[i]);
    printf("Enter the number of frames: ");
    scanf("%d", &frame);
    LRU(pg,n,frame);
}
```

OUTPUT

```
Enter the Length of sequence: 20
Enter the Sequence: 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
Enter the number of frames: 4
Values  Frame1  Frame2  Frame3  Frame4
7 =>    7
0 =>    7   0
1 =>    7   0   1
2 =>    7   0   1   2
0 =>        Hit!!!
3 =>    3   0   1   2
0 =>        Hit!!!
4 =>    3   0   4   2
2 =>        Hit!!!
3 =>        Hit!!!
0 =>        Hit!!!
3 =>        Hit!!!
2 =>        Hit!!!
1 =>    3   0   1   2
2 =>        Hit!!!
0 =>        Hit!!!
1 =>        Hit!!!
7 =>    7   0   1   2
0 =>        Hit!!!
1 =>        Hit!!!
Total Page Faults = 8
Total Page hits = 12
Hit ratio = 0.600000
```

RESULT

Successfully implemented LRU page replacement algorithm

PROGRAM-C

```
#include <stdio.h>
void main()
{
    int f, p;
    int pages[50], frame[10], hit = 0, count[50], time[50];

    int page, flag, least, minTime, temp;

    printf("Enter no of pages : ");
    scanf("%d", &p);
    printf("Enter no of frames : ");
    scanf("%d", &f);
    for (int i = 0; i < f; i++)
    {
        frame[i] = -1;
    }
    for (int i = 0; i < 50; i++)
    {
        count[i] = 0;
    }
    printf("Enter page no : \n");
    for (int i = 0; i < p; i++)
    {
        scanf("%d", &pages[i]);
    }


    for (int i = 1; i <= f; i++)
        printf("Frame%d\t", i);


    printf("\n");
    for (int i = 0; i < p; i++)
    {
        count[pages[i]]++;
        time[pages[i]] = i;
        flag = 1;
        least = frame[0];
        for (int j = 0; j < f; j++)
        {
```

c. LFU

AIM

Write a C Program to implement LFU page replacement algorithm

ALGORITHM

1. Start
2. Declare f,p indicating no of frames and no of pages
3. Declare pages[50],frame[10],count[50],time[50]
4. Declare hit and initialise as 0
5. Declare page,flag,least,minTime,temp
6. Read no of pages and frames in p and f
7. Input all entries of the frame[] as -1 indicating initially all frames are free
8. Set all entries of count[] as 0 indicating all request page initially has 0 frequency
9.Read each page no request on page[i]
10.Now for each page no , set its count++ ,time =i where i is its arrival ,flag to 0 and least as first frame and do the following
    10.1 For each of the frame check if page already there or any free frame is there then
        If page present then
            Set hit++
        Set flag=0,frame[j]=page[i] and break
        If count of least frame is greater than current one then set least to current frame
    10.2 if flag==1 indicating replacement needed then
        Set minTime=50
        For each frame
            If count of frame == count of least and frame's time < minTime
                Then set temp=current frame j and minTime =its time
        Set count of temp frame as 0 and frame[temp]=page[i]
    10.3 for each frame j if it is not -1 print it
11. Display page fault as p-hit,hit as hit, hit ratio as hit/p
12. Stop

```
    if (frame[j] == -1 || frame[j] == pages[i])
        {
            if (frame[j] != -1)
            {
                hit++;
            }
            flag = 0;
            frame[j] = pages[i];
            break;
        }
        if (count[least] > count[frame[j]])
        {
            least = frame[j];
        }
      }
      if (flag)
      {
        minTime = 50;
        for (int j = 0; j < f; j++)
        {
            if (count[frame[j]] == count[least] && time[frame[j]] < minTime)
            {
                temp = j;
                minTime = time[frame[j]];
            }
        }
        count[frame[temp]] = 0;
        frame[temp] = pages[i];
      }
      for (int j = 0; j < f; j++)
      {
       if(frame[j]!=-1)
        printf("%d \t\t", frame[j]);
      }
      printf("\n");
   }
    printf("Page faults = %d\n", p-hit);
  printf("Page hit = %d\n", hit);
  printf("Hit Ratio = %f\n", hit/(float)p);
}
```

OUTPUT

```
Enter no of pages : 10
Enter no of frames : 3
Enter page no :
2 3 4 2 1 3 7 5 4 3
Frame1  Frame2  Frame3
2
2       3
2       3       4
2       3       4
2       1       4
2       1       3
2       7       3
2       7       5
2       4       5
2       4       3
Page faults = 9
Page hit = 1
Hit Ratio = 0.100000
```

RESULT

Successfully implemented LFU page replacement algorithm

PROGRAM

```
#include<stdio.h>
void main()
{
   int ninst, np;
   printf("Enter the number of processes: ");
   scanf("%d", &np);
   printf("Enter the number of resource instances: ");
   scanf("%d", &ninst);

   // Inputting Allocation & Max Matrix
   int allocmat[np][ninst], max[np][ninst], Available[ninst],need[np][ninst];;

   printf("Input the Allocation Matrix:\n");
   for(int i = 0; i < np; i++)
   {
      for(int j = 0; j < ninst; j++)
      {
         scanf("%d", &allocmat[i][j]);
      }
   }

   printf("Input the Max Matrix:\n");
   for(int i = 0; i < np; i++)
   {
      for(int j = 0; j < ninst; j++)
      {
         scanf("%d", &max[i][j]);
      }
   }

   printf("\nInput the Available Matrix:\n");
   for(int i = 0; i < ninst; i++)
   {
      scanf("%d", &Available[i]);
   }

   // Calculate need matrix
   printf("\nNeed Matrix:\n");
   for(int i = 0; i < np; i++)
   {   printf("\t\n");
      for(int j = 0; j < ninst; j++)
```

EXPERIMENT NO .09

BANKER'S ALGORITHM

AIM

Write a C Program to implement the banker's algorithm for deadlock avoidance.

ALGORITHM

1. Start
2. Read the no of processes in np and resources in ninst
3. Declare allocation matrix , max matrix , need matrix of size np*ninst and available array of size ninst
4. Read each entries of the allomat[np][ninst] matrix and store it in the allocmat[i][j]
5. Read each entries of the max[np][ninst] matrix and store it in themax[i][j]
6. Read each entries of the available[ninst] array and store it in the available[i]
7. Calculate each entries of the need matrix and print it
    7.1 for i from 0 to np-1
        7.1.1 for j from 0 to ninst-1
            Set need[i][j]=max[i][j]-allocmat[i][j]
            Display need[i][j]
8. Declare flag,completed[np],safeseq[np],index=0 ,checker=0
9. Set all the entries of completed[np] to 0 because initially no process has completed its execution
10. Check if current available is enough to meet the need of any process , if yes add the process to safeseq[np] , set completed[i] to 1 and add its allocmat[i][ninst] to available[ninst] otherwise break and repeat the process . The process is repeated for np*np times and for each of these checking is done for utmost ninst times
    10.1 for c from 0 to np-1
        10.1.1 for i from 0 to np-1
            Set flag=0
            For j from 0 to ninst-1 check if need[i][j]>available[j] if yes
                Set flag= 1 and break
            If flag==0 then
                safeseq[index++]=i
                For k from 0 to ninst-1 set Available[k]+=allocmat[i][k]
                Set completed[i]=1 and checker++
11. If checker ==np Display "Safe Sequence"
    11.1. For i from 0 to np-1
        Display safeseq[i]

```
    {
        need[i][j] = max[i][j] - allocmat[i][j];
        printf("%d\t", need[i][j]);
    }
}



int flag, completed[np], safeseq[np],checker=0, index = 0;
for(int i = 0; i < np; i++)
{
    completed[i] = 0;
}

for(int c = 0; c < np; c++)
{
    for(int i = 0; i < np; i++)
    {
        if(completed[i] == 0)
        {
            flag = 0;
            for(int j = 0; j < ninst; j++)
            {
                if(need[i][j] > Available[j])
                {
                    flag = 1;
                    break;
                }
            }
            if(flag == 0)
            {
                safeseq[index++] = i;
                for(int k = 0; k < ninst; k++)
                {
                    Available[k] += allocmat[i][k];
                }
                completed[i] = 1;
                checker++
            }
        }
    }
}
```

12 Otherwise display “Deadlock occurs”
13.Stop

```
if(checker==np){
    printf("\n\nSafe Sequence is:\n\t");
    for(int i = 0; i < np - 1; i++)
    {
        printf("P%d -> ", safeseq[i]);
    }
    printf("P%d\n", safeseq[np - 1]);
 }
  else{
          printf("\nDeadlock occurs");
          }
}
```

OUTPUT

```
Enter the number of processes: 5
Enter the number of resource instances: 3
Input the Allocation Matrix:
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2
Input the Max Matrix:
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3

Input the Available Matrix:
3 3 2

Need Matrix:
7   4   3
1   2   2
6   0   0
0   1   1
4   3   1

Safe Sequence is:
    P1 -> P3 -> P4 -> P0 -> P2
```

RESULT

Successfully implemented Banker’s algorithm for deadlock avoidance.

PROGRAM

```
#include<stdio.h>
void main()
{
    int ninst, np;
    printf("Enter the number of processes: ");
    scanf("%d", &np);
    printf("Enter the number of resources : ");
    scanf("%d", &ninst);
  printf("\n");
    int totres[ninst];
    for(int i=0;i<ninst;i++)
      {
      printf("Total no of instances of Resource R%d: ",i+1);
      scanf("%d",&totres[i]);
      }

    // Inputting Allocation & Max Matrix
    int allocmat[np][ninst], Available[ninst],need[np][ninst];;

    printf("\nInput the Allocation Matrix:\n");
    for(int i = 0; i < np; i++)
    {
        for(int j = 0; j < ninst; j++)
        {
            scanf("%d", &allocmat[i][j]);
        }
    }

    printf("\nInput the need Matrix:\n");
    for(int i = 0; i < np; i++)
    {
        for(int j = 0; j < ninst; j++)
        {
            scanf("%d", &need[i][j]);
        }
    }

  //Calculating available array
  for(int j=0;j<ninst;j++)
    {
      Available[j]=totres[j];
```

EXPERIMENT NO .10

DEADLOCK DETECTION ALGORITHM

AIM

Write a C Program to implement deadlock detection algorithm

ALGORITHM

1. Start
2. Read the no of processes in np and resources in ninst
3. Declare totres[ninst]
4. Read total no of instances of each resources and store it in totres[i]
5. Declare allocation matrix , need matrix of size np*ninst and available array of size ninst
6. Read each entries of the allomat[np][ninst] matrix and store it in the allocmat[i][j]
7. Read each entries of the need[np][ninst] matrix and store it in the need[i][j]
8. Calculate the available no of instances of each resources
    8.1 for j from 0 to ninst-1
        8.1.1 Set Available[j]=totres[j]
        8.1.2 for i from 0 to np-1
            Set Available[j] - = allocmat[i][j]
9. Declare flag,completed[np],safeseq[np],index=0 ,checker=0
10. Set all the entries of completed[np] to 0 because initially no process has completed its execution
11. Check if current available is enough to meet the need of any process , if yes add the process to safeseq[np] , set completed[i] to 1 and add its allocmat[i][ninst] to available[ninst] otherwise break and repeat the process . The process is repeated for np*np times and for each of these checking is done for utmost ninst times
    11.1 for c from 0 to np-1
        11.1.1 for i from 0 to np-1
            Set flag=0
            For j from 0 to ninst-1 check if need[i][j]>available[j] if yes
                Set flag= 1 and break
            If flag==0 then
                safeseq[index++]=i
                For k from 0 to ninst-1 set Available[k]+=allocmat[i][k]
                Set completed[i]=1 and checker++
12. If checker ==np Display “No deadlock and Safe Sequence is”
    12.1. For i from 0 to np-1

```
                Display safeseq[i]
        for(int i = 0; i< np; i++)
       Available[j]-=allocmat[i][j];
  }


  int flag, completed[np], safeseq[np],checker=0, index = 0;
  for(int i = 0; i < np; i++)
  {
     completed[i] = 0;
  }

  for(int c = 0; c < np; c++)
  {
     for(int i = 0; i < np; i++)
     {
        if(completed[i] == 0)
        {
           flag = 0;
           for(int j = 0; j < ninst; j++)
           {
              if(need[i][j] > Available[j])
              {
                 flag = 1;
                 break;
              }
           }
           if(flag == 0)
           {
              safeseq[index++] = i;
              for(int k = 0; k < ninst; k++)
              {
                 Available[k] += allocmat[i][k];
              }
              completed[i] = 1;
            checker++;
           }
        }
     }
  }
if(checker==np){
  printf("\nNo Deadloack\nSafe Sequence is:\n\t");
  for(int i = 0; i < np - 1; i++)
  {   printf("P%d -> ", safeseq[i]); }
```

13. Otherwise display “Deadlock Detected”
14. .Stop

```
    printf("P%d\n", safeseq[np - 1]);
 }
  else{printf("\nDeadlock Detected");}
}
```

OUTPUT

```
Enter the number of processes: 3
Enter the number of resources : 3

Total no of instances of Resource R1: 1
Total no of instances of Resource R2: 2
Total no of instances of Resource R3: 4

Input the Allocation Matrix:
0 0 1
1 3 6
9 5 1

Input the need Matrix:
1 0 2
2 0 9
1 1 0

Deadlock Detected
```

RESULT

Successfully implemented a C program to detect deadlock.

PROGRAM-A

```
#include <stdio.h>
void main()
{
   int n,flag[50], len[20], start[20], t[20], bno[20][20];
   int blockalloc[50]={0};

   printf("Enter no.of files:");
   scanf("%d", &n);
   for (int i = 0; i < n; i++)
   {
      printf("\nEnter the starting block of file.%d : ", i + 1);
      scanf("%d", &start[i]);
      printf("Enter length of file%d : ", i + 1);
      scanf("%d", &len[i]);
      flag[i]=1;
      for(int j=start[i];j<start[i]+len[i];j++)
        if(blockalloc[j]==1)
        {
          printf("Block.%d already allocated\n",j);
          flag[i]=0;
          break;
        }
     if(flag[i]){
      for(int j=start[i];j<start[i]+len[i];j++){
        blockalloc[j]=1;
      }
      t[i] = start[i];
      for (int j = 0; j < len[i]; j++)
         bno[i][j] = t[i]++;
      printf("File allocated successfully\n");
     }
   }
   printf("FileNo\tStart\tlength\n");
   for (int i = 0; i < n; i++)
     if(flag[i])
      printf("%d\t\t %d \t\t%d\n", i + 1, start[i], len[i]);

  int choice,search;
  do{
   printf("\nEnter file No:");
   scanf("%d", &search);
```

EXPERIMENT NO .11

FILE ALLOCATION STRATEGIES

a. Sequential or Contiguous Allocation

AIM

Write a C Program to implement Sequential file allocation strategy

ALGORITHM

1. Start
2. Declare n,flag[50],len[20], start[20], t[20], bno[20][20] and blockalloc[50]={0}
3. Read no of files in n
4.Now read each of the files start block and length and allocate it in block if possible for that use a loop
For i from 0 to n-1
    4.1 Read start block of file and length to start[i] and len[i]
    4.2 Set flag[i]=1
    4.3 Check if the needed blocks for this file is already allocated if yes set flag[i]=0 and break
        For j from start[i] to start[i]+len[i]-1
            If blockalloc[j]==1 then set flag[i]=0 and break
    4.4. If flag[i]==1 means blocks needed are free and allocate it
        For j from start[i] to start[i]+len[i]-1
            Set blockalloc[j]=1
        Set t[i]=start[i]
        For j from 0 to len[i]
            Set bno[i][j]=t[i]++
        Print file allocated successfully
5. Display each files fileno as i ,start block address as start[i] and length as len[i] where i =0 to n-1 and flag[i]==1
6.Declare choice and search
7.Read searching file no in search
8. If flag[search-1]==1 then display its length and blocks assigned
   Otherwise print file was not allocated
9. Read choice and if choice==1 go to line 7
10. Stop

```
    if(flag[search-1]){
      printf("File No : %d\n", search);
      printf("Length : %d\n", len[search - 1]);
      printf("blocks occupied:");
      for (int i = 0; i < len[search - 1]; i++)
          printf("%4d", bno[search - 1][i]);
    }
    else
      printf("File%d not allocated \n",search);
    printf("\nDo you want to search more (1/0) : ");scanf("%d",&choice);
  }while(choice);
}
```

OUTPUT

```
Enter no.of files:3

Enter the starting block of file.1 : 3
Enter length of file1 : 4
File allocated successfully

Enter the starting block of file.2 : 4
Enter length of file2 : 2
Block.4 already allocated

Enter the starting block of file.3 : 8
Enter length of file3 : 2
File allocated successfully

FileNo  Start   length
1         3         4
3         8         2

Enter file No:1
File No : 1
Length : 4
blocks occupied:   3   4   5   6
Do you want to search more (1/0) : 1

Enter file No:2
File2 not allocated

Do you want to search more (1/0) : 0
```

RESULT

Successfully implemented Sequential file allocation strategy

PROGRAM-B

```
#include <stdio.h>

struct file {
    char fname[10];
    int start, size, block[10];
    //int flag;
} f[10];

void main() {
    int n, allocated[10];

    // Initialize allocated array to 0
    for (int i = 0; i < 10; i++) {
        allocated[i] = 0;
    }

    // Get the number of files
    printf("Enter no. of files:");
    scanf("%d", &n);

    // Get the details of each file
    for (int i = 0; i < n; i++) {
        //f[i].flag=1;
        printf("\nEnter file name:");
        scanf("%s", f[i].fname);
        printf("Enter starting block:");
        scanf("%d", &f[i].start);
        f[i].block[0] = f[i].start;
        printf("Enter no.of blocks:");
        scanf("%d", &f[i].size);
        printf("Enter block numbers:\n");
        for (int j = 1; j <= f[i].size; j++) {
           x: scanf("%d", &f[i].block[j]);

            // Check if the block is already allocated
            if (allocated[f[i].block[j]] == 1) {
                printf("Block %d is already allocated-->enter another block\n", f[i].block[j]);
                //f[i].flag=0;
                goto x;
               // break;
            }
```

b. Linked List Allocation

AIM

Write a C Program to implement Linked list file allocation strategy

ALGORITHM

1. Start
2. Create an array structure file with fname[10],start,size,block[10] as f[10]
3. Declare n,allocated[10]
4. Set all entries of allocated[] as 0
5. Read no of files in n
6. Now for each of the file read its name,start,length and the block they need
   If the blocks are already allocated try to read another free block from user
   For i from 0 to n-1
        6.1 Read f[i].fname, f[i]start, f[i].size,
        6.2 for j from 1 to size
                X: read f[i].block[j]
                If its allocated is 1 then print already allocated and goto X to read another block from the user
                Set allocated of f[i].block[j] to 1
7. Display each file's details
    For i from 0 to n-1
        7.1 Display f[i].fname, f[i].start, f[i].size
        7.2 Display each f[i].block[j] where j=1 to f[i].size
8.Stop

```
            // Mark the block as allocated
            allocated[f[i].block[j]] = 1;
        }
    }

    // Print the details of each file
    printf("File\tstart\tsize\tblock\n");
    for (int i = 0; i < n; i++) {
      //if(f[i].flag){
        printf("%s\t%d\t\t%d\t", f[i].fname, f[i].start, f[i].size);
        for (int j = 1; j <= f[i].size - 1; j++) {
            printf("%d--->", f[i].block[j]);
        }
        printf("%d", f[i].block[f[i].size]);
        printf("\n");
     // }
    }

}
```

OUTPUT

```
Enter no. of files:3

Enter file name:f1.txt
Enter starting block:3
Enter no.of blocks:3
Enter block numbers:
3
4
1

Enter file name:f2.txt
Enter starting block:6
Enter no.of blocks:2
Enter block numbers:
6
1
Block 1 is already allocated-->enter another block
5

Enter file name:f3.txt
Enter starting block:8
Enter no.of blocks:4
Enter block numbers:
8
6
Block 6 is already allocated-->enter another block
2
9
7

File    start   size    block
f1.txt   3       3          3--->4--->1
f2.txt   6       2          6--->5
f3.txt   8       4          8--->2--->9--->7
```

RESULT

Successfully implemented Linked list file allocation strategy

PROGRAM-C

```
#include <stdio.h>

struct file {
    char fname[10];
    int index, size, block[10];
    //int flag;
} f[10];

void main() {
    int n, allocated[10];

    // Initialize allocated array to 0
    for (int i = 0; i < 10; i++) {
        allocated[i] = 0;
    }

    // Get the number of files
    printf("Enter no. of files:");
    scanf("%d", &n);

    // Get the details of each file
    for (int i = 0; i < n; i++) {
        //f[i].flag=1;
        printf("\nEnter file name:");
        scanf("%s", f[i].fname);
        x: printf("Enter index block:");
         scanf("%d", &f[i].index);
        if(allocated[f[i].index])
        {
          printf("Index block is already allocated --try another\n");
          goto x;
        }
        allocated[f[i].index]=1;     //not allowing the index block to use as data block
        f[i].block[0] = f[i].index;
        printf("Enter no.of blocks:");
        scanf("%d", &f[i].size);
        printf("Enter block numbers:\n");
        for (int j = 1; j <= f[i].size; j++) {
           y: scanf("%d", &f[i].block[j]);

            // Check if the block is already allocated
```

c. Indexed File Allocation

AIM

Write a C Program to implement Indexed file allocation strategy

ALGORITHM

1. Start
2. Create an array structure file with fname[10],index,size,block[10] as f[10]
3. Declare n,allocated[10]
4. Set all entries of allocated[] as 0
5. Read no of files in n
6. Now for each of the file read its name,index block(allow to continue if index block is not already allocated otherwise read index block again) then set its allocated to 1 and f[i].block[0] to index then read length and the block they need
   If the blocks are already allocated try to read another free block from user
   For i from 0 to n-1
        6.1 Read f[i].fname, f[i]start, f[i].size,
        6.2 for j from 1 to size
                X: read f[i].block[j]
                If its allocated is 1 then print already allocated and goto X to read another block from the user
                Set allocated of f[i].block[j] to 1
        6.3 for j from 1 to f[i].size
                Display f[i].block[0] (index block)pointing to each f[i].block[j]
7. Display each file's details
    For i from 0 to n-1
        7.1 Display f[i].fname, f[i].index, f[i].size
8.Stop

```
        if (allocated[f[i].block[j]] == 1) {
              printf("Block %d is already allocated-->enter another block\n", f[i].block[j]);
              //f[i].flag=0;
              goto y;
             // break;
           }

           // Mark the block as allocated
           allocated[f[i].block[j]] = 1;
       }
      printf("\n");
      for(int j=1;j<=f[i].size;j++)
       {
         printf("%d ----> %d\n",f[i].block[0],f[i].block[j]);
       }
    }

    // Print the details of each file
    printf("\nFile\tindex\tsize\n");
    for (int i = 0; i < n; i++) {
        printf("%s\t%d\t\t%d\n", f[i].fname, f[i].index, f[i].size);
    }

}
```

OUTPUT

```
Enter no. of files:3

Enter file name:f1.txt
Enter index block:4
Enter no.of blocks:2
Enter block numbers:
2
4
Block 4 is already allocated-->enter another block
1

4 ----> 2
4 ----> 1

Enter file name:f2.txt
Enter index block:1
Index block is already allocated --try another
Enter index block:7
Enter no.of blocks:3
Enter block numbers:
6
3
2
Block 2 is already allocated-->enter another block
9

7 ----> 6
7 ----> 3
7 ----> 9

Enter file name:f3.txt
Enter index block:8
Enter no.of blocks:1
Enter block numbers:
5
8 ----> 5

File    index   size
f1.txt    4         2
f2.txt    7         3
f3.txt    8         1
```

RESULT

Successfully implemented Indexed file allocation strategy

PROGRAM-A

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

void main()
{
  int  n, seekTime=0, diff;
  printf("Enter the size of Queue: ");
  scanf("%d", &n);
  int queue[n+1];
  printf("Enter the Queue: ");

  for(int i=1;i<=n;i++)
    scanf("%d",&queue[i]);

  printf("Enter the initial head position: ");
  scanf("%d", &queue[0]); /* head element */
  printf("\nMovement of Cylinders\n");

  for(int i=0;i<n;i++)
  {
    diff= abs(queue[i+1] - queue[i]);
    seekTime+= diff;
    printf("%d -> ", queue[i]);
  }
  printf("%d",queue[n]);
  printf("\nTotal Seek Time: %d", seekTime);
  printf("\nAverage Seek Time = %f",(float) seekTime/n);
}
```

OUTPUT

```
Enter the size of Queue: 5
Enter the Queue: 61 23 34 58 11
Enter the initial head position: 33

Movement of Cylinders
33 -> 61 -> 23 -> 34 -> 58 -> 11
Total Seek Time: 148
Average Seek Time = 29.600000
```

EXPERIMENT NO .12

DISK SCHEDULING ALGORITHMS

a. FCFS Disk Scheduling

AIM

Write a C Program to implement FCFS Disk scheduling algorithm

ALGORITHM

1. Start
2. Declare n,seekTime,diff
3. Set seekTime=0
4. Read the size of the queue in n
5. Declare queue[n+1]
6. Read each of the requested tracks from the user and store it in queue[i] where i=1 to n
7. Read the initial head position in queue[0]
8. Display “Movement of cylinder”
9. Now move the R/W head from initial position to consecutive elements in queue[n+1] in the same order they arrived and find the absolute difference b/w consecutive elements and add it to the total seekTime and display the requested track
    9.1 for i from 0 to n
        9.1.1 Set diff=abs(queue[i+1]-queue[i])
        9.1.2 Set seekTime+=diff
        9.1.3 Display queue[i]
10. Display Total seek time as seekTime
11. Display average seek time as (seekTime/n)
12. Stop

RESULT

Successfully implemented FCFS disk scheduling algorithm

PROGRAM-B

```
#include<stdio.h>

void main()
{
  int n,head,max,diff,temp,temp1=0,temp2=0,seek=0;
  int i,j,k;

  printf("Enter the max range of disk : ");
  scanf("%d",&max);

  printf("Enter the initial head position : ");
  scanf("%d",&head);

  printf("Enter the size of queue request : ");
  scanf("%d",&n);

 int queue[n+2],queue1[n+2],queue2[n+2];

  printf("Enter the queue of disk positions to be read\n");
  for(i=1;i<=n;i++)
  {
    scanf("%d",&temp);
          if(temp>=head){
     queue1[temp1]=temp;
     temp1++; }
   else{
     queue2[temp2]=temp;
     temp2++;
   }
  }

  for(i=0;i<temp1-1;i++)
  {
    for(j=0;j<temp1-i-1;j++)
    { if(queue1[j]>queue1[j+1]){
         temp=queue1[j];
                          queue1[j]=queue1[j+1];
                          queue1[j+1]=temp;
                        }
                }
         }
```

b. SCAN Disk Scheduling

AIM

Write a C Program to implement SCAN Disk scheduling algorithm

ALGORITHM

1. Start
2. Declare n,head,max,diff,i,j,k,seek,temp,temp1,temp2
3. Initialise seek=0,temp1=0,temp2=0
4. Read the max range in max
5. Read the initial position of head in head
6. Read the size of queue in n
7. Declare queue[n+2],queue1[n+2],queue2[n+2]
8. Read each requested track from user
    8.1 for i from 1 to n
        8.1.1 Read temp
        8.1.2 if temp>=head
            Set queue1[temp1++]=temp
        8.1.3 Otherwise
            Set queue2[temp2++]=temp
9.Sort queue1 using bubble sort in ascending order ie swap if queue1[j]>queue1[j+1]
10. Sort queue2 using bubble sort in descending order that is swap the element if queue2[j]<queue2[j+1]
11. Set queue[0]=head
12. Add each elements of queue1 to queue
    12.1 Set i=1
    12.2 for j from 0 to temp1-1
        Set queue[i]=queue1[j] and i++
13. Set queue[i]=max
14. Add each element of queue2 to queue
    14.1 Set i=temp1+2
    14.2 for j from 0 to temp2-1
        Set queue[i]=queue2[j] and i++
15. Now print each elements of queue in order and find absolute difference b/w consecutive elements and add it to seek
    15.1 for i from 0 to n+1
        Set diff=abs(queue[i+1]-queue[i])
        Set seek+=diff
        Display queue[i]
16. Display total seek time as seek
17. Display average seek time as seek/(float)n
18.Stop

```
  for(i=0;i<temp2-1;i++){
          for(j=0;j<temp2-i-1;j++){
            if(queue2[j]<queue2[j+1]){
              temp=queue2[j];
              queue2[j]=queue2[j+1];
              queue2[j+1]=temp;
    }
          }
  }

   queue[0]=head;
  for(i=1,j=0;j<temp1;i++,j++)
            queue[i]=queue1[j];

  queue[i]=max;

  for(i=temp1+2,j=0;j<temp2;i++,j++)
    queue[i]=queue2[j];


  for(i=0;i<n+1;i++){
    diff=abs(queue[i+1]-queue[i]);
    seek+=diff;
    printf(" %d -> ",queue[i]);
          }
          printf("%d",queue[n+1]);

  printf("\n Total seek time is %d\n",seek);
  printf("Average seek time is %f\n",seek/(float)n);

}
```

OUTPUT

```
Enter the max range of disk : 199
Enter the initial head position : 53
Enter the size of queue request : 8
Enter the queue of disk positions to be read
98 183 37 122 14 124 65 67
 53 ->  65 ->  67 ->  98 ->  122 ->  124 ->  183 ->  199 ->  37 -> 14
 Total seek time is 331
Average seek time is 41.375000
```

RESULT

Successfully implemented SCAN disk scheduling algorithm

PROGRAM-C

```
#include<stdio.h>

void main()
{
  int n,head,max,diff,temp,temp1=0,temp2=0,seek=0;
  int i,j,k;

  printf("Enter the max range of disk : ");
  scanf("%d",&max);

  printf("Enter the initial head position : ");
  scanf("%d",&head);

  printf("Enter the size of queue request : ");
  scanf("%d",&n);

 int queue[n+3],queue1[n+3],queue2[n+3];

  printf("Enter the queue of disk positions to be read\n");
  for(i=1;i<=n;i++)
  {
    scanf("%d",&temp);
          if(temp>=head){
     queue1[temp1]=temp;
     temp1++; }
   else{
     queue2[temp2]=temp;
     temp2++;
   }
  }

  for(i=0;i<temp1-1;i++)
  {
    for(j=0;j<temp1-i-1;j++)
    { if(queue1[j]>queue1[j+1]){
         temp=queue1[j];
                              queue1[j]=queue1[j+1];
                              queue1[j+1]=temp;
                            }
                  }
         }
```

c. C- SCAN Disk Scheduling

AIM

Write a C Program to implement C- SCAN Disk scheduling algorithm

ALGORITHM

1. Start
2. Declare n,head,max,diff,i,j,k,seek,temp,temp1,temp2
3. Initialise seek=0,temp1=0,temp2=0
4. Read the max range in max
5. Read the initial position of head in head
6. Read the size of queue in n
7. Declare queue[n+3],queue1[n+3],queue2[n+3]
8. Read each requested track from user
    8.1 for i from 1 to n
        8.1.1 Read temp
        8.1.2 if temp>=head
            Set queue1[temp1++]=temp
        8.1.3 Otherwise
            Set queue2[temp2++]=temp
9.Sort queue1 using bubble sort in ascending order ie swap if queue1[j]>queue1[j+1]
10. Sort queue2 using bubble sort in ascending order that is swap the element if queue2[j]>queue2[j+1]
11. Set queue[0]=head
12. Add each elements of queue1 to queue
    12.1 Set i=1
    12.2 for j from 0 to temp1-1
        Set queue[i]=queue1[j] and i++
13. Set queue[i]=max
14. Set queue[i+1]=0
15. Add each element of queue2 to queue
    15.1 Set i=temp1+3
    15.2 for j from 0 to temp2-1
        Set queue[i]=queue2[j] and i++
16. Now print each elements of queue in order and find absolute difference b/w consecutive elements and add it to seek
    16.1 for i from 0 to n+1
        Set diff=abs(queue[i+1]-queue[i]) and seek+=diff
        Display queue[i]
17. Display total seek time as seek
18. Display average seek time as seek/(float)n
19.Stop

```
  for(i=0;i<temp2-1;i++){
          for(j=0;j<temp2-i-1;j++){
            if(queue2[j]>queue2[j+1]){
              temp=queue2[j];
              queue2[j]=queue2[j+1];
              queue2[j+1]=temp;
    }
          }
  }

   queue[0]=head;
  for(i=1,j=0;j<temp1;i++,j++)
            queue[i]=queue1[j];

  queue[i]=max;
  queue[i+1]=0;

  for(i=temp1+3,j=0;j<temp2;i++,j++)
    queue[i]=queue2[j];

  for(i=0;i<n+2;i++){
    diff=abs(queue[i+1]-queue[i]);
    seek+=diff;
    printf(" %d -> ",queue[i]);
          }
          printf("%d",queue[n+2]);

  printf("\n Total seek time is %d\n",seek);
  printf("Average seek time is %f\n",seek/(float)n);

}
```

OUTPUT

```
Enter the max range of disk : 199
Enter the initial head position : 53
Enter the size of queue request : 8
Enter the queue of disk positions to be read
98 183 37 122 14 124 65 67
 53 ->  65 ->  67 ->  98 ->  122 ->  124 ->  183 ->  199 ->  0 ->  14 -> 37
 Total seek time is 382
Average seek time is 47.750000
```

RESULT

Successfully implemented C-SCAN disk scheduling algorithm

PROGRAM

```
#include<stdio.h>
 #include<stdlib.h>
 #include<fcntl.h>
 #include<errno.h>
 #include<string.h>
 #include<sys/types.h>

 void main()
 {
 int c,fd,sz;
 char *c1=(char*)calloc(100,sizeof(char));
  char str[10];

 do{
 printf("\n***CHOOSE OPERATION*** \n 1.Create\n 2.Open\n 3.Close\n 4.Write\n
5.Read\n 6.Exit\n");
   printf("Enter your choice : ");
 scanf("%d",&c);
 switch(c)
 {
 case 1:
   fd=creat("f1.txt",0777);
   printf("File created\n");
   printf("fd=%d\n",fd);
   close(fd);
   break;
 case 2:
   fd=open("f1.txt",O_RDONLY|O_CREAT,0777);
   printf("Opened the file with fd=%d\n",fd);
   close(fd);
   break;
case 3:
   fd=open("f1.txt",O_RDONLY|O_CREAT,0777);
   close(fd);
   printf("Closed the file with fd=%d\n",fd);
   break;
 case 4:
    fd=open("f1.txt",O_WRONLY|O_CREAT,0777);
     printf("\nEnter data :"); scanf("%s",str);
    sz=write(fd,str,strlen(str));
    printf("Data written to file\n\n");
```

EXPERIMENT NO .13

I/O SYSTEM CALLS OF LINUX

AIM

Write a C Program using the I/O system calls of the linux operating system.

ALGORITHM

1. Start
2. Attach necessary libraries to access I/O system call
3. Declare c,fd,sz ,c1 pointer
4.Display “1.Create,2Open 3.Close 4.Write 5.Read 6.Exit”
5. Read choice in c
6. If c==1 then create a file with f1.txt and print the file descriptor
7. If c==2 then open a file f1.txt in read only (if not present create one)
8. If c==3 then first open a file f1.txt or create one and then close its file descriptor
9. If c== 4 then open f1.txt in write only format or create one
    9.1 Read data from user and write it to the file f1.txt
    9.2 then open it in read only form and print the content
10. If c==5 then open file in read only or create one then read its content and print it
    10.1 Read content to c1 and sz will have last read position and add ‘\0’ to end indicating end of file line
    10.2 Display c1
11. If c==6 then print exiting
12 if c is not 1,2,3,4,5,6 then print to enter a valid choice
13.if c!=6 then go to line 5
14. Stop

```
 case 5:
    fd=open("f1.txt",O_RDONLY);
    sz=read(fd,c1,10);
    printf("Size of character string sz=%d\n",sz);
    printf("fd=%d\n",fd);
    c1[sz]='\0';
    printf("Content in the file is %s\n",c1);
    close(fd);
    break;
 case 6:
   printf("Exiting....");
   break;
default:
    printf("\nEnter a valid choice!!\n");
  }
 } while(c!=6);
}
```

OUTPUT

```
***CHOOSE OPERATION***
 1.Create
 2.Open
 3.Close
 4.Write
 5.Read
 6.Exit
Enter your choice : 1
File created
fd=3

***CHOOSE OPERATION***
 1.Create
 2.Open
 3.Close
 4.Write
 5.Read
 6.Exit
Enter your choice : 2
Opened the file with fd=3

***CHOOSE OPERATION***
 1.Create
```

```
 2.Open
 3.Close
 4.Write
 5.Read
 6.Exit
Enter your choice : 3
Closed the file with fd=3

***CHOOSE OPERATION***
 1.Create
 2.Open
 3.Close
 4.Write
 5.Read
 6.Exit
Enter your choice : 4
Enter data :hello
Data written to file

Size of character string sz=5
fd=4
Content in the file is hello

***CHOOSE OPERATION***
 1.Create
 2.Open
 3.Close
 4.Write
 5.Read
 6.Exit
Enter your choice : 5
Size of character string sz=5
fd=4
Content in the file is hello

***CHOOSE OPERATION***
 1.Create
 2.Open
 3.Close
 4.Write
 5.Read
 6.Exit
Enter your choice : 6
Exiting...
```

RESULT

Successfully implemented a c program using I/O system calls