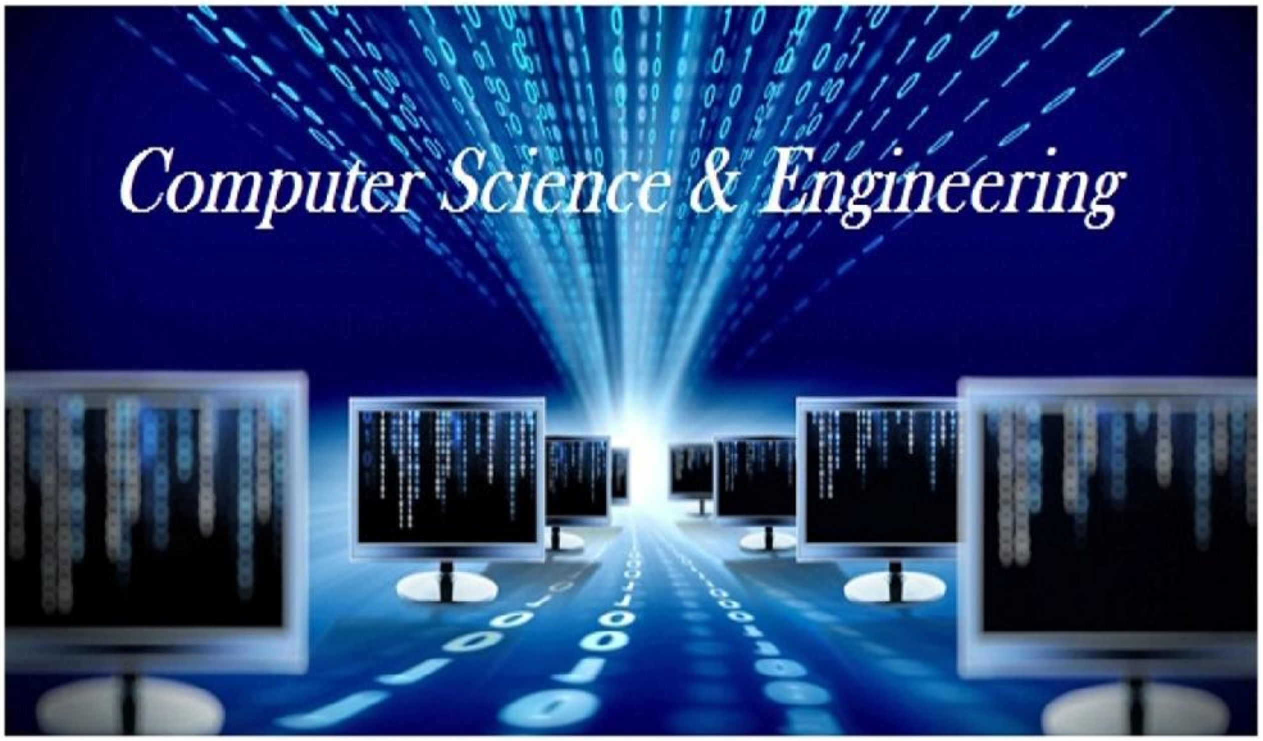
Varuvan Vadivelan Institute of Technology

Dharmapuri – 636 703

LAB MANUAL

Regulation	**: 2013**
Branch	**: B.E. - CSE**
Year & Semester	**: II Year / IV Semester**

CS6413-OPERATING SYSTEM LABORATORY



ANNA UNIVERSITY CHENNAI

REGULATION -2013

CS 6413 – OPERATING SYSTEMS LABORATORY

LIST OF EXPERIMENTS:

1. Basics of UNIX commands.
2. Shell programming
3. Implementation of CPU scheduling. a) Round Robin b) SJF c) FCFS d) Priority
4. Implement all file allocation strategies
5. Implement Semaphores
6. Implement ll File Organization Techniques a
7. Implement Bankers algorithm for Dead Lock Avoidance
8. Implement an Algorithm for Dead Lock Detection
9. Implement the all page replacement algorithms a) FIFO b) LRU c) LFU
10. Implement Shared memory and IPC
11. Implement Paging Technique f memory management.
12. Implement Threading & Synchronization Applications

Total hours: 45

INDEX

Operating System

Introduction

A computer system can be divided into 4 components:

- ***Hardware*** *(CPU, memory, input/output devices, etc.),*
- ***Operating system****,*
- ***System programs*** *(word processors, spread sheets, accounting software's, compilers,)*
- ***Application programs****.*

In 1960's definition of an operating system is ***"software that controls the hardware".*** However, today, due to microcode we need a better definition. We see an operating system as the programs that make the hardware useable. In brief, an operating system is the set of programs that controls a computer.

An Operating system is software that creates a relation between the User, Software and Hardware. It is an interface between the all. All the computers need basic software known as an Operating System (OS) to function.

The OS acts as an interface between the User, Application Programs, Hardware and the System Peripherals. The OS is the first software to be loaded when a computers starts up. The entire application programs are loaded after the OS.

Types of Operating System (Based of No. of user):

1. **Single User:** If the single user ***Operating System*** is loaded in computer's memory; the computer would be able to handle one user at a time.

   Ex: MS-Dos, MS-Win 95-98, Win-ME

1. **Multi user:** If the multi-user ***Operating System*** is loaded in computer's memory; the computer would be able to handle more than one user at a time.

   Ex: UNIX, Linux, XENIX

2. **Network:** If the network ***Operating System*** is loaded in computer's memory; the computer would be able to handle more than one computer at time.

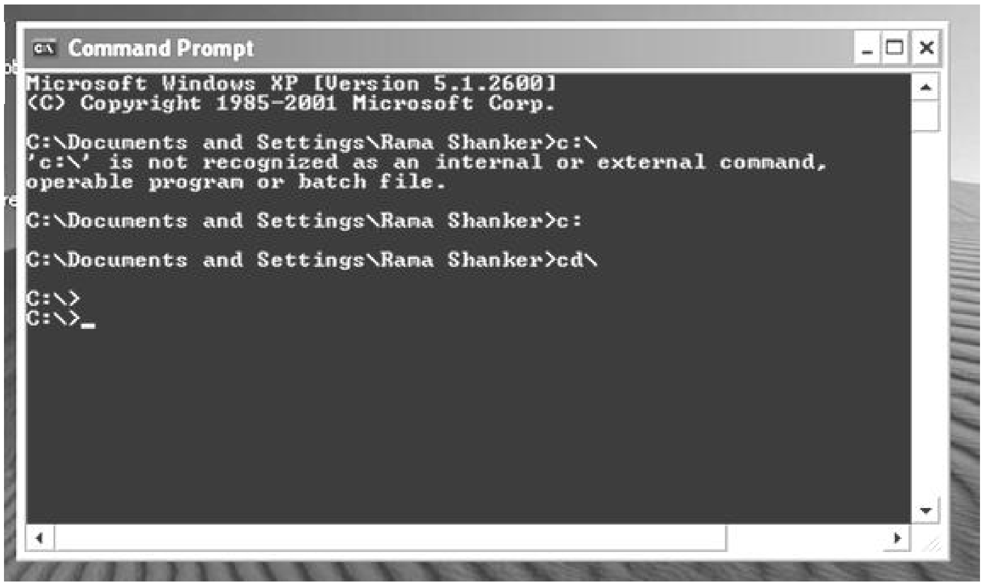
   Ex: Novel Netware, Win-NT, Win-2000-2003

Command Prompt Interface:

Operating System provides a text based interface called command prompt. From the command prompt commands can be issued to perform file and disk management and to run program. Results of these commands are presented to the user as text message.

C:\>-

The command prompt can be an alphabet followed by one colon (:), one back slash (\), one greater than sign (>) and one blinking element called cursor (_).



```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Rama Shanker>c:\
'c:\' is not recognized as an internal or external command,
operable program or batch file.

C:\Documents and Settings\Rama Shanker>c:

C:\Documents and Settings\Rama Shanker>cd\

C:\>
C:\>_
```

Where C:		represents the Drive letter (Current Drive)
	\	represents the current folder / Directory
	>	represents the end of the Prompt and
	_	blinking element (represents the Cursor)

Operating Systems Types

Single- And Multi-Tasking Operating Systems

A single-tasking system can only run one program at a time, while a multi-tasking operating system allows more than one program to be running in concurrency. This is achieved by time-sharing, dividing the available processor time between multiple processes that are each interrupted repeatedly in time slices by a task-scheduling subsystem of the operating system.

Multi-tasking may be characterized in preemptive and co-operative types. In preemptive multitasking, the operating system slices the CPU time and dedicates a slot to each of the programs. Unix-like operating systems, e.g., Solaris, Linux, as well as AmigaOS support preemptive multitasking.

Single- And Multi-User Operating Systems

Single-user operating systems have no facilities to distinguish users, but may allow multiple programs to run in tandem. A multi-user operating system extends the basic concept of multi-tasking with facilities that identify processes and resources, such as disk space, belonging to multiple users, and the system permits multiple users to interact with the system at the same time. Time-sharing operating systems schedule tasks for efficient use of the system and may also include accounting software for cost allocation of processor time, mass storage, printing, and other resources to multiple users.

Distributed Operating Systems

A distributed operating system manages a group of distinct computers and makes them appear to be a single computer. The development of networked computers that could be linked and communicate with each other gave rise to distributed computing. Distributed computations are carried out on more than one machine. When computers in a group work in cooperation, they form a distributed system.

Embedded Operating Systems

Embedded operating systems are designed to be used in embedded computer systems. They are designed to operate on small machines like PDAs with less autonomy. They are able to operate with a limited number of resources. They are very compact and extremely efficient by design. Windows CE and “***Minix 3***”are some examples of embedded operating systems.

Real-Time Operating Systems

A real-time operating system is an operating system that guarantees to process events or data by a specific moment in time. A real-time operating system may be single- or multi-tasking, but when multitasking, it uses specialized scheduling algorithms so that a deterministic nature of behavior is achieved. An event-driven system switches between tasks based on their priorities or external events while time-sharing operating systems switch tasks based on clock interrupts

Process Scheduling

Processes are the Small Programs those are executed by the user according to their Request. CPU Executes all the Process according to Some Rules or Some Schedule. Scheduling ist hat in which each process have Some Amount of Time of CPU. Scheduling Provides Time of CPU to the Each Process.

Types of Process Scheduling

1. FCFS Scheduling Algorithm

The First Come First Served (FCFS) Scheduling Algorithm is the simplest one. In this algorithm the set of ready processes is managed as FIFO (first-in-first-out) Queue. The processes are serviced by the CPU until completion in order of their entering in the FIFO queue.

A process once allocated the CPU keeps it until releasing the CPU either by terminating or requesting I/O. For example, interrupted process is allowed to continujre running after interrupt handling is done with.

2. SJF Scheduling Algorithm

The Shortest Job First Scheduling Algorithm chooses the process that has the smallest next CPU burst.

3. SRTF: Shortest Remaining Time First

This is the preemptive version of SJF. The currently executing process will be preempted from the CPU if a process with a shorter CPU burst time is arrived.

4. Round Robin Scheduling

This scheduling algorithm is designed especially for time sharing systems. It is similar to FCFS scheduling, but preemption is added to switch between processes.

EX NO: 1

DATE:

PROCESS SYSTEM CALLS

AIM:

To write c program to implement the Process system calls.

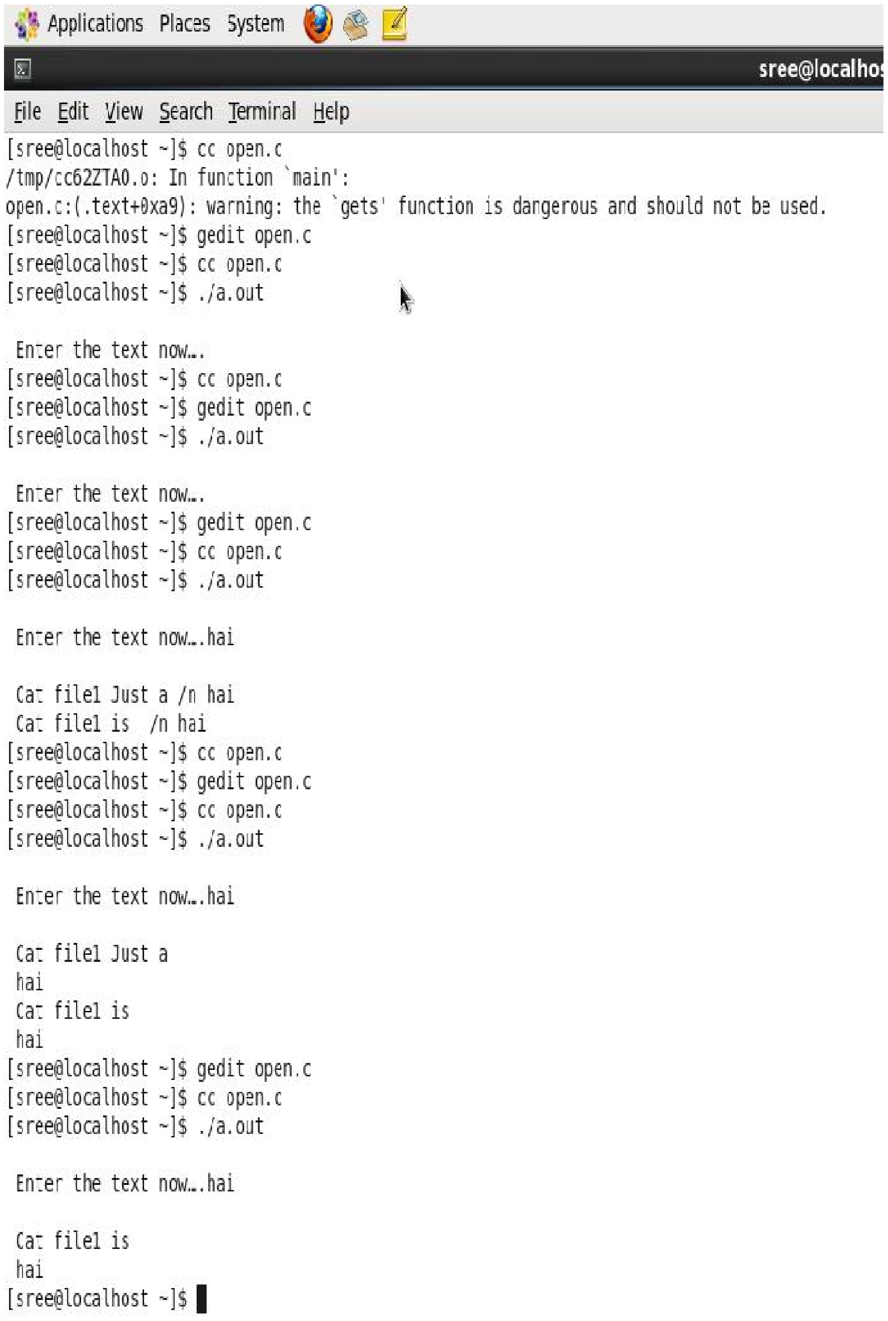
ALGORITHM:

1. Start the program.
2. Declare the pid and get the pid by using the getpid() method.
3. Create a child process by calling the fork() system call
4. Check if(pid==0) then print the child process id and then print the parent process value. Otherwise print
5. Stop the program

PROGRAM: (PROCESS SYSTEM CALLS)

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
void main(int argc,char *arg[])
{
int pid; pid=fork();
if(pid<0)
{
printf("fork failed");
exit(1);
}
else if(pid==0)
{
execlp("whoami","ls",NULL);
exit(0);
}
else
{
printf("\n Process id is -%d\n",getpid());
wait(NULL);
exit(0);
}
}
```

OUTPUT:


Applications Places System
sree@localhos
File Edit View Search Terminal Help

```
[sree@localhost ~]$ cc open.c
/tmp/cc62ZTA0.o: In function `main':
open.c:(.text+0xa9): warning: the `gets' function is dangerous and should not be used.
[sree@localhost ~]$ gedit open.c
[sree@localhost ~]$ cc open.c
[sree@localhost ~]$ ./a.out

 Enter the text now...
[sree@localhost ~]$ cc open.c
[sree@localhost ~]$ gedit open.c
[sree@localhost ~]$ ./a.out

 Enter the text now...
[sree@localhost ~]$ gedit open.c
[sree@localhost ~]$ cc open.c
[sree@localhost ~]$ ./a.out

 Enter the text now...hai

 Cat file1 Just a /n hai
 Cat file1 is  /n hai
[sree@localhost ~]$ cc open.c
[sree@localhost ~]$ gedit open.c
[sree@localhost ~]$ cc open.c
[sree@localhost ~]$ ./a.out

 Enter the text now...hai

 Cat file1 Just a
 hai
 Cat file1 is
 hai
[sree@localhost ~]$ gedit open.c
[sree@localhost ~]$ cc open.c
[sree@localhost ~]$ ./a.out

 Enter the text now...hai

 Cat file1 is
 hai
[sree@localhost ~]$ 
```

RESULT:

Thus the process system call program was executed and verified successfully.

EX.NO:2

DATE:

IO SYSTEM CALLS

AIM:

To write a ‘c’ program for I/O system calls.

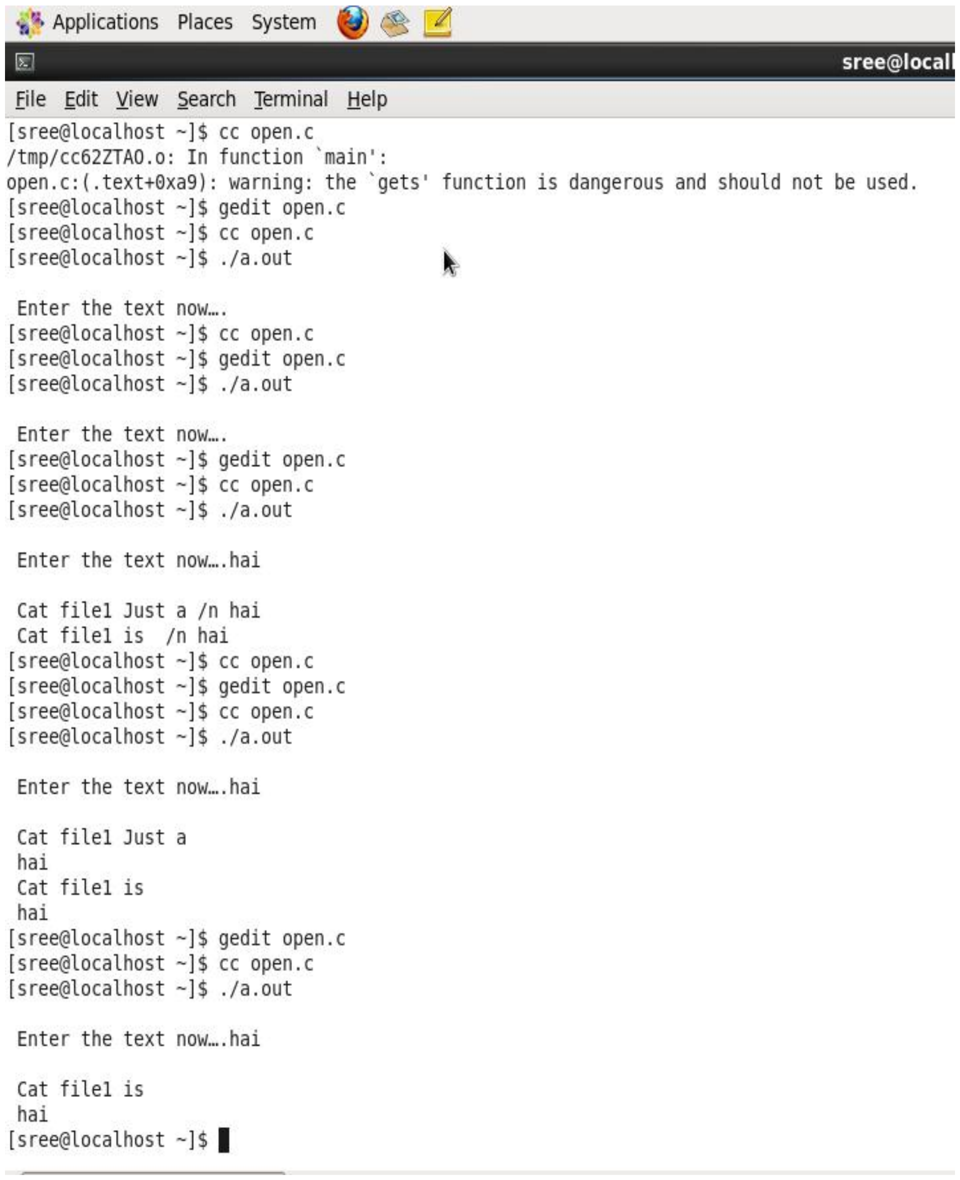
ALGORITHM:

1. Start the program.
2. open a file for O_RDWR for R/W,O_CREATE for creating a file , O_TRUNC for truncate a file
3. Using getchar(), read the character and stored in the string[] array
4. The string [] array is write into a file close it.
5. Then the first is opened for read only mode and read the characters and displayed It and close the file
6. Stop the program

PROGRAM :(IO SYSTEM CALLS)

```
#include<stdio.h>

#include<unistd.h>

#include<string.h>

#include<fcntl.h>

main( )

{

int fd[2];

char buf1[25]= "just a test\n";

char buf2[50];

fd[0]=open("file1",O_RDWR);

fd[1]=open("file2",O_RDWR);

write(fd[0], buf1, strlen(buf1));

printf("\n Enter the text now….");

scanf("\n %s",buf1);


printf("\n Cat file1 is  \n hai");

write(fd[0], buf1, strlen(buf1));

lseek(fd[0], SEEK_SET, 0);

read(fd[0], buf2, sizeof(buf1));

write(fd[1], buf2, sizeof(buf2));

close(fd[0]);

close(fd[1]);

printf("\n");

return 0;

}
```

OUTPUT:


Applications Places System
sree@locall
File Edit View Search Terminal Help

```
[sree@localhost ~]$ cc open.c
/tmp/cc62ZTA0.o: In function `main':
open.c:(.text+0xa9): warning: the `gets' function is dangerous and should not be used.
[sree@localhost ~]$ gedit open.c
[sree@localhost ~]$ cc open.c
[sree@localhost ~]$ ./a.out

 Enter the text now….
[sree@localhost ~]$ cc open.c
[sree@localhost ~]$ gedit open.c
[sree@localhost ~]$ ./a.out

 Enter the text now….
[sree@localhost ~]$ gedit open.c
[sree@localhost ~]$ cc open.c
[sree@localhost ~]$ ./a.out

 Enter the text now….hai

 Cat file1 Just a /n hai
 Cat file1 is  /n hai
[sree@localhost ~]$ cc open.c
[sree@localhost ~]$ gedit open.c
[sree@localhost ~]$ cc open.c
[sree@localhost ~]$ ./a.out

 Enter the text now….hai

 Cat file1 Just a
 hai
 Cat file1 is
 hai
[sree@localhost ~]$ gedit open.c
[sree@localhost ~]$ cc open.c
[sree@localhost ~]$ ./a.out

 Enter the text now….hai

 Cat file1 is
 hai
[sree@localhost ~]$ █
```

RESULT:

Thus the I/O system call program was executed and verified successfully.

EX.NO:3

DATE:

FIRST COME FIRST SERVE SCHEDULING

AIM:

To write the program to implement CPU & scheduling algorithm for first come first serve scheduling.

ALGORITHM:

1. Start the program.
2. Get the number of processes and their burst time.
3. Initialize the waiting time for process 1 and 0.
4. Process for(i=2;i<=n;i++),wt.p[i]=p[i-1]+bt.p[i-1].
5. The waiting time of all the processes is summed then average value time is calculated.
6. The waiting time of each process and average times are displayed
7. Stop the program

PROGRAM :(FIRST COME FIRST SERVE SCHEDULING)

```
#include<stdio.h>

struct process

 {

int pid;

 int bt;

 int wt,tt;

 }p[10];

int main()

{

      int i,n,totwt,tottt,avg1,avg2; clrscr();

            printf("enter the no of process \n"); scanf("%d",&n);
      for(i=1;i<=n;i++)

      {

        p[i].pid=i;

         printf("enter the burst time n"); scanf("%d",&p[i].bt);

      }

p[1].wt=0;

p[1].tt=p[1].bt+p[1].wt;

i=2;

while(i<=n)

      {

           p[i].wt=p[i-1].bt+p[i-1].wt; p[i].tt=p[i].bt+p[i].wt;

           i ++;

      }

i=1;

totwt=tottt=0;

printf("\n processid \t bt\t wt\t tt\n"); while(i<=n){
```

```
printf("\n\t%d \t%d \t%d \t%d",p[i].pid,p[i].bt,p[i].wt,p[i].tt);
totwt=p[i].wt+totwt;

tottt=p[i].tt+tottt;

i++;}

avg1=totwt/n; avg2=tottt/n; printf("\navg1=%d \t avg2=%d
\t",avg1,avg2); getch();

return 0;

}
```

OUTPUT:

```
enter the no of process      3

enter the burst time         2

enter the burst time         4

enter the burst time         6

Process sid    bt    wt    tt

     1         2     0     2

      2        4     2     6

      3        6     6     12


avg1=2               avg2=6
```

RESULT:

Thus the FIFO process scheduling program was executed and verified successfully.

EX.NO:4

DATE:

SHORTEST JOB FIRST SCHEDULING

AIM:

To write a program to implement cpu scheduling algorithm for shortest job first scheduling.

ALGORITHM:

1. Start the program. Get the number of processes and their burst time.
2. Initialize the waiting time for process 1 as 0.
3. The processes are stored according to their burst time.
4. The waiting time for the processes are calculated a follows:
   for(i=2;i<=n;i++).wt.p[i]=p[i=1]+bt.p[i-1].
5. The waiting time of all the processes summed and then the average time is calculate
6. The waiting time of each processes and average time are displayed.
7. Stop the program.

<u>PROGRAM:</u> (SHORTEST JOB FIRST SCHEDULING)

```
#include<stdio.h>

#include<conio.h>

struct process

{

      int pid;

      int bt;

       int wt;

       int tt;

      }p[10],temp;

      int main()

{

      int i,j,n,totwt,tottt;

      float avg1,avg2;

     clrscr();

      printf("\nEnter the number of process:\t");

      scanf("%d",&n);

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

{

      p[i].pid=i;

      printf("\nEnter the burst time:\t");

      scanf("%d",&p[i].bt);

   }

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

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

  {

      if(p[i].bt>p[j].bt)

  {

      temp.pid=p[i].pid;
```

```
       p[i].pid=p[j].pid;
       p[j].pid=temp.pid;
      temp.bt=p[i].bt;p[i].bt=p[j].bt;
     p[j].bt=temp.bt;
}}}
      p[1].wt=0;
      p[1].tt=p[1].bt+p[1].wt;
      i=2;
      while(i<=n){
      p[i].wt=p[i-1].bt+p[i-1].wt;
      p[i].tt=p[i].bt+p[i].wt;
      i++;
   }
    i=1;
    totwt=tottt=0;
    printf("\nProcess id \tbt \twt \ttt");
    while(i<=n){
    printf("\n\t%d \t%d \t%d t%d\n",p[i].pid,p[i].bt,p[i].wt,p[i].tt);
    totwt=p[i].wt+totwt;
    tottt=p[i].tt+tottt;
    i++;
   }  avg1=totwt/n;
      avg2=tottt/n;
      printf("\nAVG1=%f\t AVG2=%f",avg1,avg2);
      getch();
      return 0; }
```

OUTPUT:

```
enter the number of process    3
enter the burst time: 2
enter the burst time: 4
enter the burst time: 6
processid      bt      wt      tt
1              2       0       2
2              4       2       6
3              6       6       12

AVG1=2.000000       AVG2=6.000000
```

RESULT:

Thus the SJF program was executed and verified successfully

EX.NO:5

DATE:

PRIORITY SCHEDULING

AIM:

To write a 'C' program to perform priority scheduling.

ALGORITHM:

1. Start the program.
2. Read burst time, waiting time, turn the around time and priority.
3. Initialize the waiting time for process 1 and 0.
4. Based up on the priority process are arranged
5. The waiting time of all the processes is summed and then the average waiting time
6. The waiting time of each process and average waiting time are displayed based on the priority.
7. Stop the program.

<u>PROGRAM:</u> (PRIORITY SCHEDULING)

```
#include<stdio.h>
#include<conio.h>
struct process
{
      int pid;
       int bt;
       int wt;
       int tt;
       int prior;
 }
      p[10],temp;
      int main()
     {
           int i,j,n,totwt,tottt,arg1,arg2;
           clrscr();
           printf("enter the number of process");
           scanf("%d",&n);
      for(i=1;i<=n;i++)
     {
                p[i].pid=i;
          printf("enter the burst time");
                scanf("%d",&p[i].bt);
                printf("\n enter the priority");
                scanf("%d",&p[i].prior);
      }
      for(i=1;i<n;i++)
{
      for(j=i+1;j<=n;j++)
      {
                 if(p[i].prior>p[j].prior)
           {
                       temp.pid=p[i].pid;
                        p[i].pid=p[j].pid;
                       p[j].pid=temp.pid;
                       temp.bt=p[i].bt;
                       p[i].bt=p[j].bt;
                       p[j].bt=temp.bt;
                       temp.prior=p[i].prior;
                       p[i].prior=p[j].prior;
                       p[j].prior=temp.prior;
                 }
        }
        }
      p[i].wt=0;
      p[1].tt=p[1].bt+p[1].wt;
      i=2;
      while(i<=n)
{
         p[i].wt=p[i-1].bt+p[i-1].wt;
        p[i].tt=p[i].bt+p[i].wt;
         i++;
   }
      i=1;
      totwt=tottt=0;
      printf("\n process to \t bt \t wt \t tt");
      while(i<=n)
{
```

```
printf("\n%d\t %d\t %d\t %d\t",p[i].pid,p[i].bt,p[i].wt,p[i].tt);
     totwt=p[i].wt+totwt;
         tottt=p[i].tt+tottt;
    i++;
}
    arg1=totwt/n;
    arg2=tottt/n;
   printf("\n arg1=%d \t arg2=%d\t",arg1,arg2);
   getch();
   return 0;
    }
```

OUTPUT:

```
enter the no of process:3

enter the burst time:2

enter the priority:3

enter the burst time:4

enter the priority:1

 enter the burst time:6

enter the priority:2

process to      bt      wt      tt

        1       4       0       4       4

        2       6       4       10      14

        3       2       10      12      22


        avg1=4              avg2=8
```

RESULT:

Thus the priority scheduling program was executed and verified successfully

EX.NO:6

DATE:

ROUND ROBIN SCHEDULING

AIM:

To write a program to implement cpu scheduling for Round Robin Scheduling.

ALGORITHM:

1. Get the number of process and their burst time.
2. Initialize the array for Round Robin circular queue as ‘0’.
3. The burst time of each process is divided and the quotients are stored on the round Robin array.
4. According to the array value the waiting time for each process and the average time are calculated as line the other scheduling.
5. The waiting time for each process and average times are displayed.
6. Stop the program.

PROGRAM :(ROUND ROBIN SCHEDULING)

```
#include<stdio.h>
#include<conio.h>
 struct process
{
      int pid,bt,tt,wt;
 };
  int main()
     {
      struct process x[10],p[30];
      int i,j,k,tot=0,m,n;
      float wttime=0.0,tottime=0.0,a1,a2;
      clrscr();
      printf("\nEnter the number of process:\t");
      scanf("%d",&n);
      for(i=1;i<=n;i++){
      x[i].pid=i;
      printf("\nEnter the Burst Time:\t");
      scanf("%d",&x[i].bt);
      tot=tot+x[i].bt;
       }
      printf("\nTotal Burst Time:\t%d",tot);
              p[0].tt=0;
       k=1;
      printf("\nEnter the Time Slice:\t");
      scanf("%d",&m);
      for(j=1;j<=tot;j++)
{
           for(i=1;i<=n;i++)
```

```
{
                if(x[i].bt !=0)
{
            p[k].pid=i;
      if(x[i].bt-m<0)
{
      p[k].wt=p[k-1].tt;
      p[k].bt=x[i].bt;
      p[k].tt=p[k].wt+x[i].bt;
       x[i].bt=0;
        k++;
  }
      else
{
      p[k].wt=p[k-1].tt;
      p[k].tt=p[k].wt+m;
      x[i].bt=x[i].bt-m;
      k++;
 }
 }
 }
  }
      printf("\nProcess id \twt \ttt");
      for(i=1;i<k;i++){
       printf("\n\t%d \t%d \t%d",p[i].pid,p[i].wt,p[i].tt);
      wttime=wttime+p[i].wt;
      tottime=tottime+p[i].tt;
      a1=wttime/n;
      a2=tottime/n;
```

```
}
printf("\n\nAverage Waiting Time:\t%f",a1);
printf("\n\nAverage TurnAround Time:\t%f",a2);
getch();
return 0;
}
```

OUTPUT:

```
enter the no of process3
enter the burst time3
enter the burst time5
enter the burst time7
total burst time : 15
enter the time slice: 2
process id    wt    tt
        1     0     2
        2     2     4
        3     4     6
        1     6     7

process id    wt    tt
        2     7     9
        3     9     11
        2     11    12
        3     12    14
        3     14    15
avg waiting time: 21.666666
avg turnaround time: 26.666666
```

RESULT:

Thus the Round Robin scheduling program was executed and verified successfully.

EX.NO:7

DATE:

PIPE PROCESSING

AIM :

To write a program for create a pope processing

ALGORITHM:

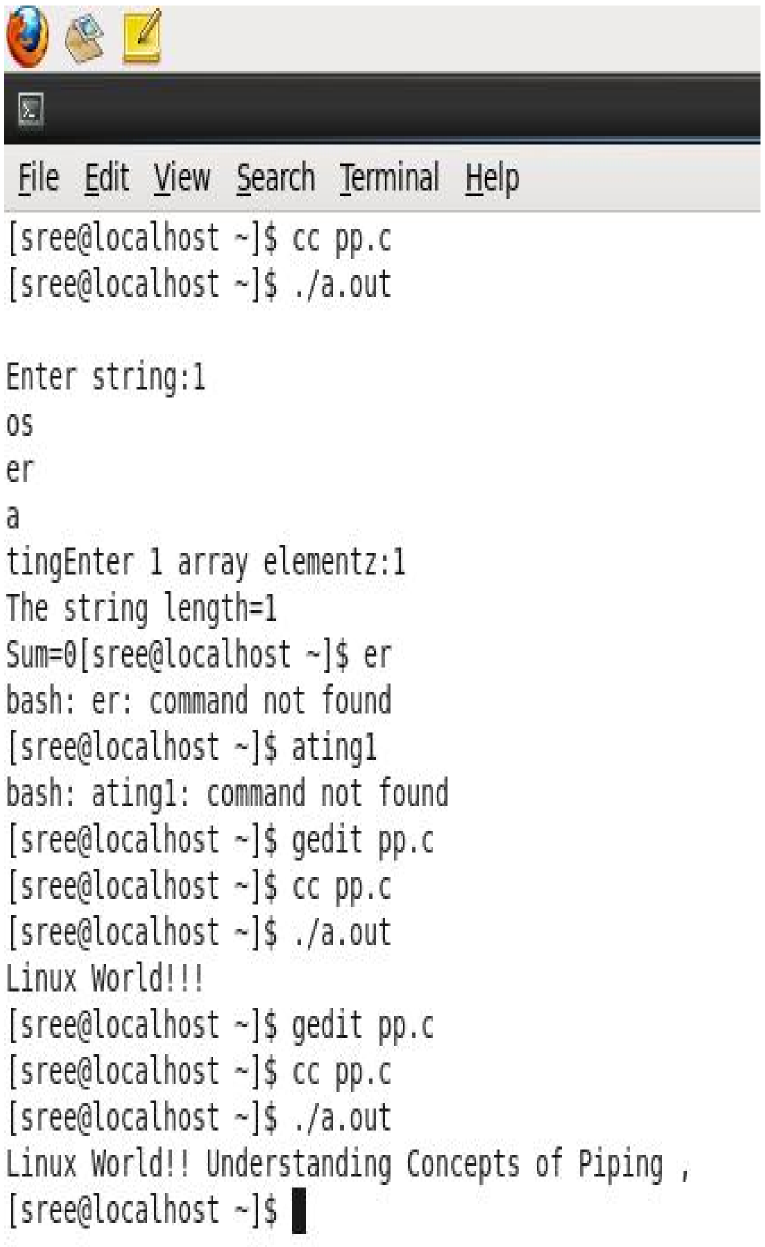
1. Start the program.
2. Declare the variables.
3. Read the choice.
4. Create a piping processing using IPC.
5. Assign the variable lengths
6. "*strcpy*" the message lengths.
7. To join the operation using IPC .
8. Stop the program

.

PROGRAM :(PIPE PROCESSING)

```
#include <unistd.h>

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

#define MSG_LEN 64

int main(){

   int     result;

   int     fd[2];

   char    message[MSG_LEN];

   char    recvd_msg[MSG_LEN];

   result = pipe (fd);

//Creating a pipe//fd[0] is for reading and fd[1] is for writing

   if (result < 0)

{

       perror("pipe ");

       exit(1);

   }

   strncpy(message,"Linux World!! ",MSG_LEN);

   result=write(fd[1],message,strlen(message));

   if (result < 0)

{

       perror("write");

            exit(2);

   }

   strncpy(message,"Understanding ",MSG_LEN);

   result=write(fd[1],message,strlen(message));

   if (result < 0)

{
      perror("write");
          exit(2);
  }
  strncpy(message,"Concepts of ",MSG_LEN);
  result=write(fd[1],message,strlen(message));
  if (result < 0)
{
      perror("write");
        exit(2);
 }
 strncpy(message,"Piping ", MSG_LEN);
 result=write(fd[1],message,strlen(message));
 if (result < 0)
{
      perror("write");
       exit(2);
 }
 result=read (fd[0],recvd_msg,MSG_LEN);
 if (result < 0)
{
       perror("read");
      exit(3);
 }
 printf("%s\n",recvd_msg);
 return 0;}
```

OUTPUT:



```
[sree@localhost ~]$ cc pp.c
[sree@localhost ~]$ ./a.out

Enter string:1
os
er
a
tingEnter 1 array elementz:1
The string length=1
Sum=0[sree@localhost ~]$ er
bash: er: command not found
[sree@localhost ~]$ ating1
bash: ating1: command not found
[sree@localhost ~]$ gedit pp.c
[sree@localhost ~]$ cc pp.c
[sree@localhost ~]$ ./a.out
Linux World!!!
[sree@localhost ~]$ gedit pp.c
[sree@localhost ~]$ cc pp.c
[sree@localhost ~]$ ./a.out
Linux World!! Understanding Concepts of Piping ,
[sree@localhost ~]$ 
```

RESULT:

Thus the Piping process using IPC program was executed and verified successfully

EX.NO:8

DATE:

PRODUCER-CONSUMER PROBLEM USING SEMOPHERES

AIM:

To implement producer/consumer problem using semaphore.

ALGORITHM:

1. Declare variable for producer & consumer as pthread-t-tid produce tid consume.
2. Declare a structure to add items, semaphore variable set as struct.
3. Read number the items to be produced and consumed.
4. Declare and define semaphore function for creation and destroy.
5. Define producer function.
6. Define consumer function.
7. Call producer and consumer.
8. Stop the execution.

PROGRAM: (PRODUCER-CONSUMER PROBLEM)

```
#include<stdio.h>
 void main()
{
   int buffer[10], bufsize, in, out, produce, consume, choice=0;
   in = 0;
   out = 0;
   bufsize = 10;
    while(choice !=3)
      {
          printf("\n1. Produce  \t 2. Consume \t3. Exit");
          printf("\nEnter your choice:  =");
           scanf("%d", &choice);
            switch(choice)
           {
           case 1:    if((in+1)%bufsize==out)
            printf("\nBuffer is Full");
                else
              {
                printf("\nEnter the value: ");
                scanf("%d", &produce);
                buffer[in] = produce;
                in = (in+1)%bufsize;
                }
                 break;
                 case 2:    if(in == out)
                 printf("\nBuffer is Empty");
                else
               {
```

```
          consume = buffer[out];
          printf("\nThe consumed value is %d", consume);
         out = (out+1)%bufsize;
          }
           break;
       }    }    }
```

OUTPUT:

```
 1. Produce      2. Consume      3. Exit
Enter your choice: 2
Buffer is Empty
1. Produce      2. Consume      3. Exit
Enter your choice: 1
Enter the value: 100
1. Produce      2. Consume      3. Exit
Enter your choice: 2
The consumed value is 100
1. Produce      2. Consume      3. Exit
Enter your choice: 3
```

RESULT:

Thus the producer consumer program was executed and verified successfully

EX.NO:9

DATE:

FIRST FIT MEMORY MANAGEMENT

AIM:

To implement first fit, best fit algorithm for memory management.

ALGORITHM:

1. Start the program.
2. Get the segment size, number of process to be allocated and their corresponding size.
3. Get the options. If the option is ‘2’ call first fit function.
4. If the option is ‘1’ call best fit function. Otherwise exit.
5. For first fit, allocate the process to first possible segment which is free and set the personnel slap as ‘1’. So that none of process to be allocated to segment which is already allocated and vice versa.
6. For best fit, do the following steps,.
7. Sorts the segments according to their sizes.
8. Allocate the process to the segment which is equal to or slightly greater than the process size and set the flag as the ‘1’ .So that none of the process to be allocated to the segment which is already allocated and vice versa. Stop the program.
9. Stop the program

PROGRAM: (FIRST FIT MEMORY MANAGEMENT)

```
#include<stdio.h>
#include<conio.h>
#define max 25
void main()
{
 int frag[max],b[max],f[max],i,j,nb,nf,temp,highest=0;
 static int bf[max],ff[max];
 clrscr();
 printf("\n\tMemory Management Scheme - First Fit");
 printf("\nEnter the number of blocks:");
 scanf("%d",&nb);
 printf("Enter the number of files:");
 scanf("%d",&nf);
 printf("\nEnter the size of the blocks:-\n");
 for(i=1;i<=nb;i++)
 {
   printf("Block %d:",i);
   scanf("%d",&b[i]);
 }
  printf("Enter the size of the files :-\n");
  for(i=1;i<=nf;i++)
 {
  printf("File %d:",i);
  scanf("%d",&f[i]);
  }
 for(i=1;i<=nf;i++)
 {
```

```
for(j=1;j<=nb;j++)
 {
    if(bf[j]!=1)    //if bf[j] is not allocated
      {
         temp=b[j]-f[i];
         if(temp>=0)

if(highest<temp)
{

ff[i]=j;

highest=temp;
 }
  }
 }
 frag[i]=highest;
 bf[ff[i]]=1;
 highest=0;
}
printf("\nFile_no:\tFile_size :\tBlock_no:\tBlock_size:\tFragement");
for(i=1;i<=nf;i++)
printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",
i,f[i],ff[i],b[ff[i]],frag[i]);
getch();
}
```

OUTPUT:

```
Enter the number of blocks: 3

Enter the number of files: 2


Enter the size of the blocks:-

Block 1: 5

Block 2:    2

Block 3: 7


Enter the size of the files:-

File 1: 1

File 2: 4


OUTPUT

File No         File Size       Block No        Block Size      Fragment

1               1               3               7               6

2               4               1               5               1
```

RESULT:

Thus the First Bit and Best Fit program was executed and verified successfully.

EX.NO:10

DATE:

FILE MANIPULATION-I

AIM:

To write a program for file manipulation for displays the file and directory in memory

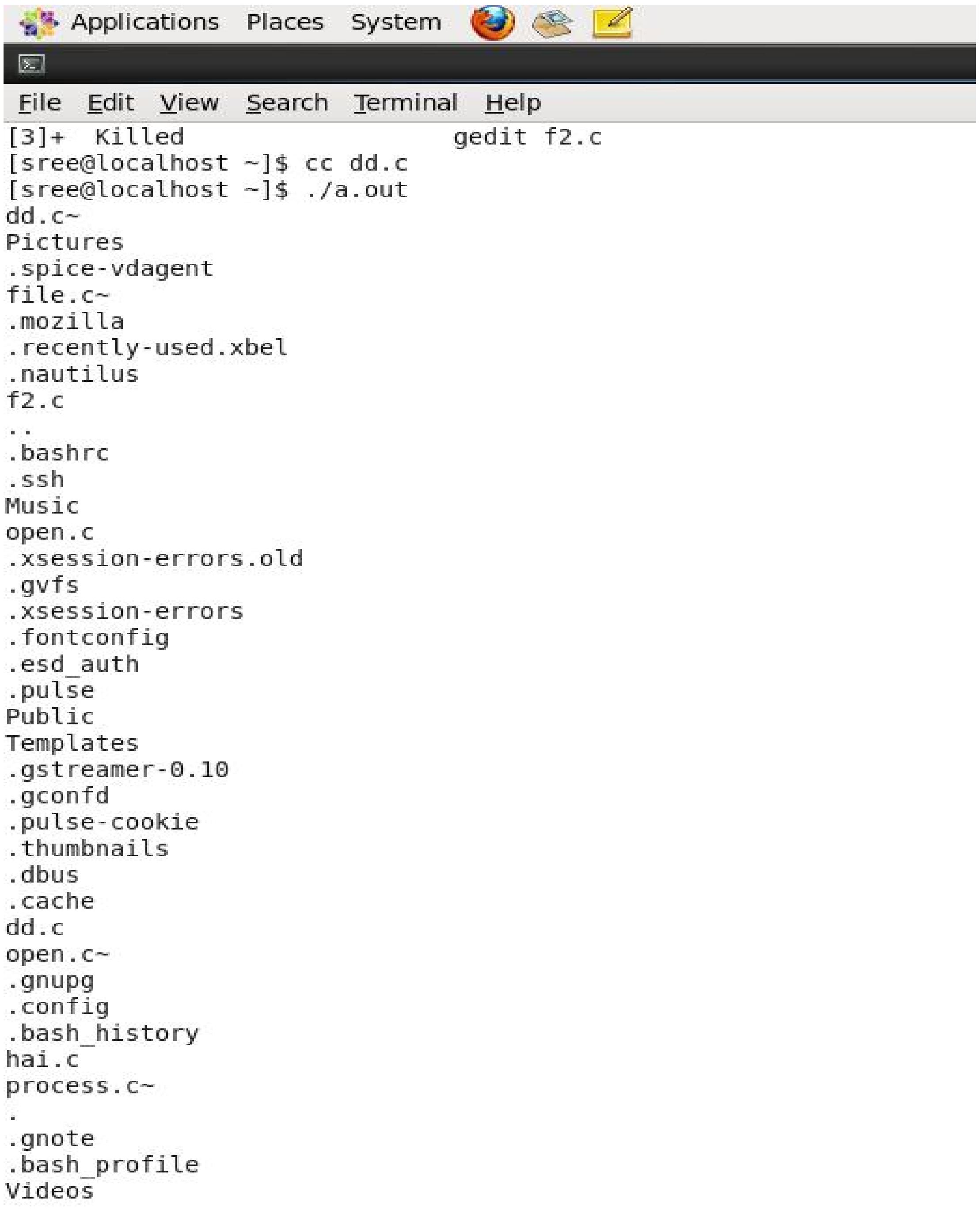
ALGORITHM:

1. Start the program.
2. Use the pre defined function list out the files in directory..
3. Main function is used to check the file present in the directory or not.
4. Using the file pointer in the file to that the argument is less than a times means print
5. By using if loop check in file, open two means print error
6. Stop the program.

<u>PROGRAM:</u> (FILE MANIPULATION-I)

```
#include <dirent.h>
#include <stdio.h>
int main(void)
{
    DIR *d;
    struct dirent *dir;
    d = opendir(".");
    if (d)
    {
        while ((dir = readdir(d)) != NULL)
        {
            printf("%s\n", dir->d_name);
        }
        closedir(d);
    }
   return(0);
}
```

OUTPUT:


Applications Places System
File Edit View Search Terminal Help

```
[3]+  Killed                  gedit f2.c
[sree@localhost ~]$ cc dd.c
[sree@localhost ~]$ ./a.out
dd.c~
Pictures
.spice-vdagent
file.c~
.mozilla
.recently-used.xbel
.nautilus
f2.c
..
.bashrc
.ssh
Music
open.c
.xsession-errors.old
.gvfs
.xsession-errors
.fontconfig
.esd_auth
.pulse
Public
Templates
.gstreamer-0.10
.gconfd
.pulse-cookie
.thumbnails
.dbus
.cache
dd.c
open.c~
.gnupg
.config
.bash_history
hai.c
process.c~
.
.gnote
.bash_profile
Videos
```

RESULT:

Thus the file management program was executed and verified successfully.

EX.NO:11

DATE:

FILE MANIPULATION-II

AIM:

To write a program performs file manipulation.

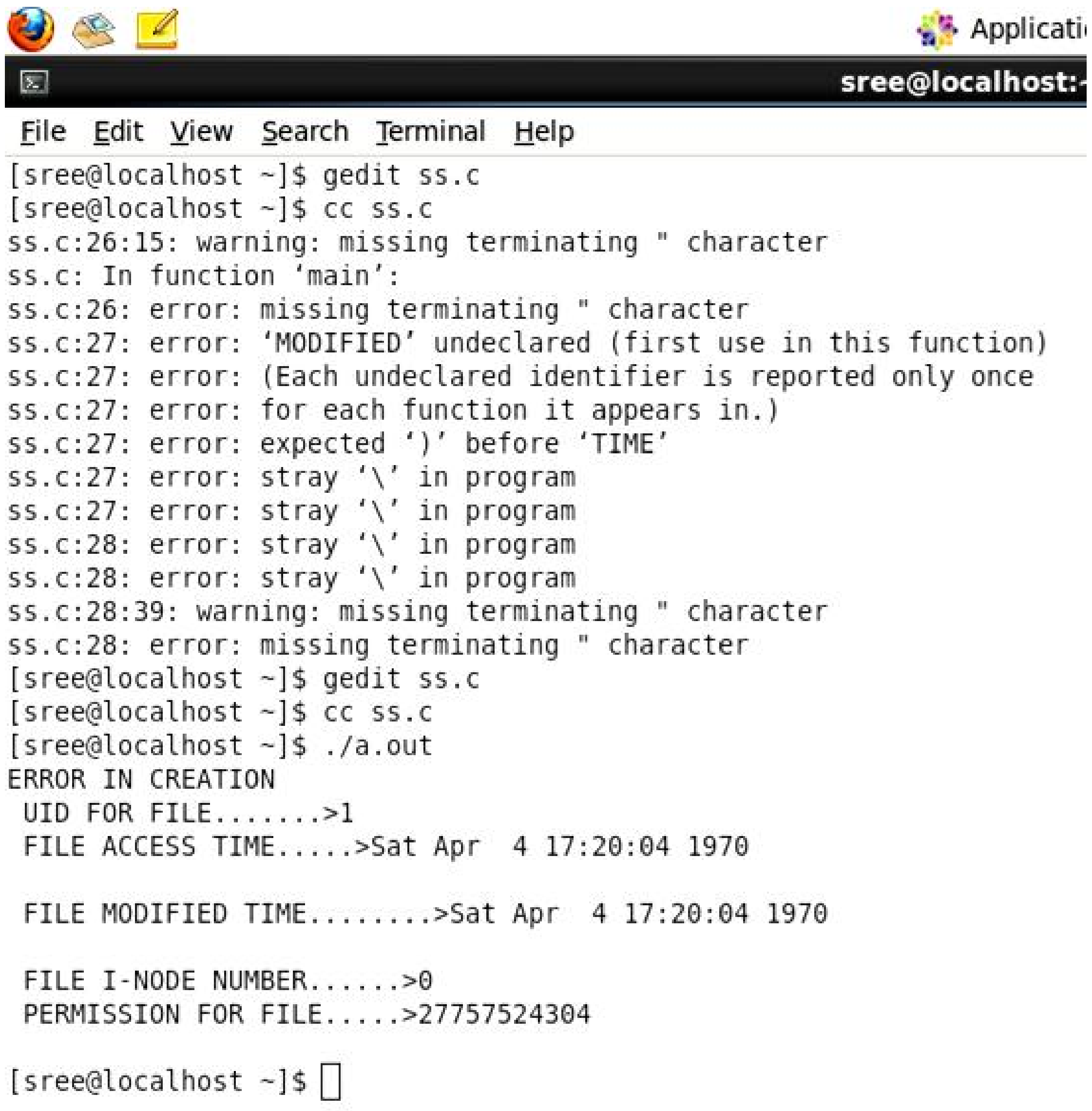
ALGORITHM:

1. Start the program.
2. Declare the arguments for file open and file create.
3. print the file in directory otherwise display the error message error in creation
4. if check the files in directory
5. close the files and directory
6. Stop the program.

<u>PROGRAM :</u>(FILE MANIPULATION-II)

```
#include<stdio.h>

#include<sys/stat.h>

#include<time.h>

main(int ag,char*arg[])

{

      char buf[100];

      struct stat s;

      int fd1,fd2,n;

      fd1=open(arg[1],0);

      fd2=creat(arg[2],0777);

      stat(arg[2],&s);

      if(fd2==-1)


      printf("ERROR IN CREATION");

      while((n=read(fd1,buf,sizeof(buf)))>0)

      {

                if(write(fd2,buf,n)!=n)

{

                     close(fd1);

                     close(fd2);

              }

       }

     printf("\t\n UID FOR FILE.......>%d \n FILE ACCESS
     TIME.....>%s \n FILE MODIFIED TIME........>%s \n FILE I-NODE
     NUMBER......>%d \n PERMISSION FOR
     FILE.....>%o\n\n",s.st_uid,ctime(&s.st_atime),ctime(&s.st_mt
     ime),s.st_mode);

       close(fd1);

       close(fd2);

}
```

OUTPUT:



```
[sree@localhost ~]$ gedit ss.c
[sree@localhost ~]$ cc ss.c
ss.c:26:15: warning: missing terminating " character
ss.c: In function ‘main’:
ss.c:26: error: missing terminating " character
ss.c:27: error: ‘MODIFIED’ undeclared (first use in this function)
ss.c:27: error: (Each undeclared identifier is reported only once
ss.c:27: error: for each function it appears in.)
ss.c:27: error: expected ‘)’ before ‘TIME’
ss.c:27: error: stray ‘\’ in program
ss.c:27: error: stray ‘\’ in program
ss.c:28: error: stray ‘\’ in program
ss.c:28: error: stray ‘\’ in program
ss.c:28:39: warning: missing terminating " character
ss.c:28: error: missing terminating " character
[sree@localhost ~]$ gedit ss.c
[sree@localhost ~]$ cc ss.c
[sree@localhost ~]$ ./a.out
ERROR IN CREATION
 UID FOR FILE.......>1
 FILE ACCESS TIME.....>Sat Apr  4 17:20:04 1970

 FILE MODIFIED TIME........>Sat Apr  4 17:20:04 1970

 FILE I-NODE NUMBER......>0
 PERMISSION FOR FILE.....>27757524304

[sree@localhost ~]$ 
```

RESULT:

Thus the File Manipulation II program was executed and verified successfully.

EX NO: 12

DATE:

SIMULATE PAGE REPLACEMENT ALGORITHMS FIFO

AIM:

To Simulate FIFO page replacement algorithms.

ALGORITHM:

1. Start the program
2. Read the number of frames
3. Read the number of pages
4. Read the page numbers
5. Initialize the values in frames to -1
6. Allocate the pages in to frames in First in first out order.
7. Display the number of page faults.
8. Stop the program

PROGRAM: **(SIMULATE PAGE REPLACEMENT ALGORITHMS FIFO)**

```
#include<stdio.h>
#include<conio.h>
int i,j,nof,nor,flag=0,ref[50],frm[50],pf=0,victim=-1;
void main()
{
 clrscr();
 printf("\n \t\t\t FIFI PAGE REPLACEMENT ALGORITHM");
 printf("\n Enter no.of frames....");
 scanf("%d",&nof);
 printf("Enter number of Pages.\n");
 scanf("%d",&nor);
 printf("\n Enter the Page No...");
 for(i=0;i<nor;i++)
 scanf("%d",&ref[i]);
 printf("\nThe given Pages are:");
 for(i=0;i<nor;i++)
 printf("%4d",ref[i]);
 for(i=1;i<=nof;i++)
 frm[i]=-1;
 printf("\n");
 for(i=0;i<nor;i++)
 {
  flag=0;
  printf("\n\t page no %d->\t",ref[i]);
  for(j=0;j<nof;j++)
   {
    if(frm[j]==ref[i])
    {
     flag=1;
     break;
    }}
     if(flag==0)
   {
    pf++;
    victim++;
    victim=victim%nof;
    frm[victim]=ref[i];
    for(j=0;j<nof;j++)
    printf("%4d",frm[j]);
   }  }
 printf("\n\n\t\t No.of pages faults...%d",pf);
 getch();
}
```

OUTPUT:

```
FIFO PAGE REPLACEMENT ALGORITHM


Enter no.of frames....4

Enter number of reference string..

6


 Enter the reference string..

5 6 4 1 2 3


The given reference string:

      ...................................... 5 6 4 1  2   3


       Reference np5->             5  -1  -1  -1

       Reference np6->             5   6  -1  -1

       Reference np4->             5   6  4   -1

       Reference np1->             5   6  4   1

       Reference np2->             2   6  4   1

       Reference np3->             2   3  4   1

       No.of pages faults...6
```

RESULT:

Thus the program FIFO page replacement was successfully executed.

EX NO:13

DATE :

SIMULATE PAGE REPLACEMENT ALGORITHMS: LRU

AIM:

To Simulate LRU page replacement algorithms

ALGORITHM:

1. Start
2. Read the number of frames
3. Read the number of pages
4. Read the page numbers
5. Initialize the values in frames to -1
6. Allocate the pages in to frames by selecting the page that has not been used for the longest period of time.
7. Display the number of page faults.
8. stop

PROGRAM: (SIMULATE PAGE REPLACEMENT ALGORITHMS: LRU)

```
 #include<stdio.h>
 #include<conio.h>
 int i,j,nof,nor,flag=0,ref[50],frm[50],pf=0,victim=-1;
 int recent[10],lrucal[50],count=0;
 int lruvictim();
 void main()
 {
clrscr();
 printf("\n\t\t\t LRU PAGE REPLACEMENT ALGORITHM");
printf("\n Enter no.of Frames....");
  scanf("%d",&nof);
  printf(" Enter no.of reference string..");
  scanf("%d",&nor);
  printf("\n Enter reference string..");
  for(i=0;i<nor;i++)
  scanf("%d",&ref[i]);
  printf("\n\n\t\t LRU PAGE REPLACEMENT ALGORITHM ");
 printf("\n\t The given reference string:");
   printf("\n……………………………..");
   for(i=0;i<nor;i++)
   printf("%4d",ref[i]);
   for(i=1;i<=nof;i++)
   {
     frm[i]=-1;
     lrucal[i]=0;
   }

   for(i=0;i<10;i++)
```

```
recent[i]=0;
printf("\n");
for(i=0;i<nor;i++)
{
  flag=0;
  printf("\n\t Reference NO %d->\t",ref[i]);
  for(j=0;j<nof;j++)
  {

     if(frm[j]==ref[i])
    {
            flag=1;
            break;
    }
  }

  if(flag==0)
  {
    count++;
    if(count<=nof)
    victim++;
    else
    victim=lruvictim();
    pf++;
    frm[victim]=ref[i];
    for(j=0;j<nof;j++)
    printf("%4d",frm[j]);
  }
  recent[ref[i]]=i;
```

```
   }
   printf("\n\n\t No.of page faults...%d",pf);
   getch();
}
int lruvictim()
{
  int i,j,temp1,temp2;
  for(i=0;i<nof;i++)
{
   temp1=frm[i];
   lrucal[i]=recent[temp1];
  }
  temp2=lrucal[0];
  for(j=1;j<nof;j++)
  {
   if(temp2>lrucal[j])
   temp2=lrucal[j];
  }
  for(i=0;i<nof;i++)
  if(ref[temp2]==frm[i])
  return i;
  return 0;
}
```

OUTPUT:

```
                LRU PAGE REPLACEMENT ALGORITHM
Enter no.of Frames....3
 Enter no.of reference string..6
 Enter reference string..
6 5 4 2 3 1

             LRU PAGE REPLACEMENT ALGORITHM
      The given reference string:
       ………………….  6   5   4   2   3   1

      Reference NO 6->            6  -1  -1
      Reference NO 5->            6   5  -1
      Reference NO 4->            6   5   4
      Reference NO 2->            2   5   4
      Reference NO 3->            2   3   4
      Reference NO 1->            2   3   1

      No.of page faults...6
```

RESULT:

Thus the process LRU page replacement was executed and verified successfully.

EX.NO:14

DATE:

SIMULATE PAGE REPLACEMENT ALGORITHMS: OPTIMAL

AIM:

To create program for optimal page replacement algorithms.

ALGORITHM:

1. Start the program
2. Read the number of frames
3. Read the number of pages
4. Read the page numbers
5. Initialize the values in frames to -1
6. Allocate the pages in to frames by selecting the page that will not be used for the longest period of time.
7. Display the number of page faults.
8. Stop the program

PROGRAM: (SIMULATE PAGE REPLACEMENT ALGORITHMS: OPTIMAL)

```
#include<stdio.h>
#include<conio.h>
int i,j,nof,nor,flag=0,ref[50],frm[50],pf=0,victim=-1;
int recent[10],optcal[50],count=0;
int optvictim();
void main()
{
  clrscr();
  printf("\n OPTIMAL PAGE REPLACEMENT ALGORITHN");
  printf("\n....................... ........");
  printf("\nEnter the no.of frames");
  scanf("%d",&nof);
  printf("Enter the no.of reference string");
  scanf("%d",&nor);
  printf("Enter the reference string");
  for(i=0;i<nor;i++)
  scanf("%d",&ref[i]);
  clrscr();
  printf("\n OPTIMAL PAGE REPLACEMENT ALGORITHM");
  printf("\n...............................");
  printf("\nThe given string");
  printf("\n...................\n");
  for(i=0;i<nor;i++)
  printf("%4d",ref[i]);
   for(i=0;i<nof;i++)
   {
       frm[i]=-1;
```

```
        optcal[i]=0;
    }
    for(i=0;i<10;i++)
        recent[i]=0;
    printf("\n");
    for(i=0;i<nor;i++)
    {
       flag=0;
       printf("\n\tref no %d ->\t",ref[i]);
       for(j=0;j<nof;j++)
       {
               if(frm[j]==ref[i])
               {
                  flag=1;
                  break;
               }
       }
       if(flag==0)
       {
               count++;
               if(count<=nof)
                   victim++;
               else
                   victim=optvictim(i);
               pf++;
               frm[victim]=ref[i];
               for(j=0;j<nof;j++)
                  printf("%4d",frm[j]);
       }
```

```
   }
   printf("\n Number of page faults: %d",pf);
   getch();
}
int optvictim(int index)
{
   int i,j,temp,notfound;
   for(i=0;i<nof;i++)
   {
      notfound=1;
      for(j=index;j<nor;j++)
              if(frm[i]==ref[j])
              {
                  notfound=0;
                  optcal[i]=j;
                  break;
              }
if(notfound==1)
             return i;
   }
  temp=optcal[0];
  for(i=1;i<nof;i++)
     if(temp<optcal[i])
             temp=optcal[i];
  for(i=0;i<nof;i++)
     if(frm[temp]==frm[i])
             return i;
 return 0;
}
```

OUTPUT:

```
OPTIMAL PAGE REPLACEMENT ALGORITHM


Enter no. of Frames....3

Enter no. of reference string..6


Enter reference string..

6 5 4 2 3 1


        OPTIMAL PAGE REPLACEMENT ALGORITHM

    The given reference string:

    ………………….  6  5  4  2  3  1


    Reference NO 6->        6  -1  -1

    Reference NO 5->        6   5  -1

    Reference NO 4->        6   5   4

    Reference NO 2->        2   5   4

    Reference NO 3->        2   3   4

    Reference NO 1->        2   3   1


    No.of page faults...6
```

RESULT:

Thus the process optimal page replacement was executed and verified successfully.

EX NO: 15

DATE:

SIMULATE ALGORITHM FOR DEADLOCK PREVENTION

AIM:

To Simulate Algorithm for Deadlock prevention

ALGORITHM:

1. Start the program
2. Attacking Mutex condition: never grant exclusive access. But this may not be possible for several resources.
3. Attacking preemption: not something you want to do.
4. Attacking hold and wait condition: make a process hold at the most 1 resource
5. At a time. Make all the requests at the beginning. Nothing policy. If you feel, retry.
6. Attacking circular wait: Order all the resources. Make sure that the requests are issued in the
7. Correct order so that there are no cycles present in the resource graph. Resources numbered 1 ... n.
8. Resources can be requested only in increasing
9. Order. i.e. you cannot request a resource whose no is less than any you may be holding.
10. Stop the program

PROGRAM: (SIMULATE ALGORITHM FOR DEADLOCK PREVENTION)

```
#include<stdio.h>
#include<conio.h>
int max[10][10], alloc[10][10], need[10][10];
int avail[10],i,j,p,r,finish[10]={0},flag=0;
int main()
{
clrscr( );
printf("\n\nSIMULATION OF DEADLOCK PREVENTION");
printf("Enter no. of processes, resources");
scanf("%d%d",&p,&r);printf("Enter allocation matrix");
for(i=0;i<p;i++)
for(j=0;j<r;j++)
scanf("%d",&alloc[i][j]);
printf("enter max matrix");
for(i=0;i<p;i++) /*reading the maximum matrix and availale matrix*/
for(j=0;j<r;j++)
scanf("%d",&max[i][j]);
printf("enter available matrix");
for(i=0;i<r;i++)
scanf("%d",&avail[i]);
for(i=0;i<p;i++)
for(j=0;j<r;j++)
need[i][j]=max[i][j]-alloc[i][j];
fun(); /*calling function*/
if(flag==0)
{
if(finish[i]!=1)
{
printf("\n\n Failing :Mutual exclusion");
for(j=0;j<r;j++)

{ /*checking for mutual exclusion*/

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

avail[j]=need[i][j];

}fun();

printf("\n By allocating required resources to process %d dead lock is
prevented ",i);

printf("\n\n lack of preemption");

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

{

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

avail[j]=need[i][j];

alloc[i][j]=0;

}

fun( );

printf("\n\n daed lock is prevented by allocating needed resources");

printf(" \n \n failing:Hold and Wait condition ");

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

{

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

avail[j]=need[i][j];

}

fun( );

printf("\n AVOIDING ANY ONE OF THE CONDITION, U CAN PREVENT DEADLOCK");

}

}

getch( );       return 0;

}

fun()

{

while(1)
```

```
{
for(flag=0,i=0;i<p;i++)
{
if(finish[i]==0)
{
for(j=0;j<r;j++)
{
if(need[i][j]<=avail[j])
continue;
else
break;
}
if(j==r)
{
for(j=0;j<r;j++)
avail[j]+=alloc[i][j];
flag=1;
finish[i]=1;
}
}
}
if(flag==0)
break;
}return 0;
}
```

OUTPUT:

```
SIMULATION OF DEADLOCK PREVENTION
Enter no. of processes, resources 3, 2
Enter allocation matrix 2 4 5
                        3 4 5
Enter max matrix4 3 4
                  5 6 1
Enter available matrix2
Failing: Mutual Exclusion
By allocating required resources to process dead is prevented
Lack of no preemption deadlock is prevented by allocating needed resources
Failing: Hold and Wait condition
```

RESULT:

Thus the program deadlock was executed successfully.