AVN INSTITUTE OF ENGINEERING & TECHNOLOGY
Accredited by NAAC & NBA

# OPERATING SYSTEMS
# LAB MANUAL

## DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

## II B.TECH – II SEM

**Academic Year 2021-22**

**R18 Regulation**

**Prepared by**

Dr. P Meena Kumari

Associate Professor

OPERATING SYSTEMS LAB MANUAL

# INDEX

# OPERATING SYSTEMS LAB MANUAL

## Course Objectives:

- To implement the scheduling algorithms.
- To write programs in Linux environment using the I/O system calls of UNIX/LINUX operating system.
- To implement deadlock avoidance
- To implement synchronization problems using semaphores using UNIX/LINUX system calls.
- To implement IPC mechanism using Pipes.FIFOs , Message Queues , Shared Memory
- To implement memory management techniques.

## Course Outcomes:

1. Able to implement c programs for different CPU scheduling algorithms.
2. Able to implement c programs for file and directory I/O system calls
3. Able to implement c programs for prevention and avoidance of deadlocks.
4. Able to implement c programs for process synchronization using semaphore and IPC mechanisms Using system calls.
5. Able to develop c programs for paging and segmentation technique

## 1. Write C programs to simulate the following CPU Scheduling algorithms:

### a) FCFS CPU SCHEDULING ALGORITHM

**DESCRIPTION:**

For FCFS scheduling algorithm, read the number of processes/jobs in the system, their CPU burst times. The scheduling is performed on the basis of arrival time of the processes irrespective of their other parameters. Each process will be executed according to its arrival time. Calculate the waiting time and turnaround time of each of the processes accordingly.

**ALGORITHM:**

Step 1: Start

Step 2: Define a structure process with elements p,bt,wt,tat.

Step 3: Read the Processes details p, & bt

Step 4: Initialize
wt[0]=avgwt=0;
avgtat=tat[0]=bt[0];

Step 5: for i=1 to i<n do till step6

Step 6: wt[i]=wt[i-1]+bt[i-1];
tat[i]=wt[i]+bt[i];
avgwt=avgwt+wt[i];
avgtat=avgtat+tat[i];

Step7: for i=0 to n do step 8

Step8: Print the output with the FCFS Fashion and Calculating bt,wt,&tat

Step 9: End

**PROGRAM:** FCFS CPU SCHEDULING ALGORITHM

```
#include<stdio.h>
int main( )
{
char  p[10][10];
int bt[10],wt[10],tat[10],i,n;
float  avgwt,avgtat;
printf("enter no of processes:");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("enter process %d name:\t",i+1);
scanf("%s",p[i]);
printf("enter burst time\t");
scanf("%d",&bt[i]);
}
wt[0]=avgwt=0;
avgtat=tat[0]=bt[0];
for(i=1;i<n;i++)
{
wt[i]=wt[i-1]+bt[i-1];
tat[i]=wt[i]+bt[i];
avgwt=avgwt+wt[i];
avgtat=avgtat+tat[i];
}
printf("p_name\t B_time\t w_time\t turnarounftime\n");
for(i=0;i<n;i++)
printf("%s\t%d\t%d\t%d\n",p[i],bt[i],wt[i],tat[i]);
printf("\navg waiting time=%f", avgwt/n);
printf("\navg tat time=%f\n", avgtat/n);
return  0; }
```

**OUTPUT:**

```
student@NNRG310:~/oslab$ cc fcfs.c
student@NNRG310:~/oslab$ ./a.out
enter no of processes:    3
enter process 1 name:    P1
enter burst time    24
enter process 2 name:    P2
enter burst time    3
enter process 3 name:    P3
enter burst time    3
p_name       B_time       w_time       turnarounftime
P1      24      0       24
P2      3       24      27
P3      3       27      30

avg waiting time=17.000000
avg tat time=27.000000
student@NNRG310:~/oslab$
```

## b) SJF CPU SCHEDULING ALGORITHM

**DESCRIPTION:**

For SJF(Shortest Job First) scheduling algorithm, read the number of processes/jobs in the system, their CPU burst times. Arrange all the jobs in order with respect to their burst times. There may be two jobs in queue with the same execution time, and then FCFS approach is to be performed. Each process will be executed acco ding to the length of its burst time. Then calculate the waiting time and turnaround time of each of the processes accordingly.

**ALGORITHM:**

Step 1: Start
Step 2: Define a structure process with elements p,bt,wt,tat
Step3: Read process name P,burst time bt of the process
Step4: for i=0 to n go to step 6
Step:5 for j=0;j< i do

```
if(bt[i]<bt[j])
{
temp=bt[i];
bt[i]=bt[j];
bt[j]=temp;
k=p[i];
p[i]=p[j];
p[j]=k;
}
```

Step6: else avgwt=wt[0]=0;
avgtat=tat[0]=bt[0];
Step 7: for i=1;i<n do
wt[i]=wt[i-1]+bt[i-1];
tat[i]=wt[i]+bt[i];

avgwt=avgwt+wt[i];

avgtat=avgtat+tat[i];

Step 8: Print the output with the SJF Fashion and Calculating Pid,bt,wt,&tat

Step 9: End

**PROGRAM:** SJF CPU SCHEDULING ALGORITHM

```
#include<stdio.h>
int main()
{
int i,j,k,n,temp;
int p[10],bt[10],wt[10],tat[10];
float avgtat,avgwt;
printf("enter no of processes: \t");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("enter process name:\t");
scanf("%d",&p[i]);
printf("enter burst time \t");
scanf("%d",&bt[i]);
}
for(i=0;i<n;i++)
{
for(j=0;j<i;j++)
{
if(bt[i]<bt[j])
{
temp=bt[i];
bt[i]=bt[j];
bt[j]=temp;
```

```
k=p[i];
p[i]=p[j];
p[j]=k;
}
}
}
avgwt=wt[0]=0;
avgtat=tat[0]=bt[0];
for(i=1;i<n;i++)
{
wt[i]=wt[i-1]+bt[i-1];
tat[i]=wt[i]+bt[i];
avgwt=avgwt+wt[i];
avgtat=avgtat+tat[i];
}
printf("p_name\t B_time\t w_time\t turnarounftime\n");
for(i=0;i<n;i++)
printf("%d\t%d\t%d\t%d\n",p[i],bt[i],wt[i],tat[i]);
printf("\navg waiting time=%f\n", avgwt/n);
printf("avg tat time=%f\n", avgtat/n);
 }
```

**OUTPUT:**

```
student@NNRG310:~/oslab$ cc sjf.c
student@NNRG310:~/oslab$  ./a.out
enter no of processes:     4
enter process name:        1
enter burst time    6
enter process name:        2
enter burst time    8
```

enter process name: 3

enter burst time 7

enter process name: 4

enter burst time 3

p_name B_time w_time turnarounftime

4 3 0 3

1 6 3 9

3 7 9 16

2 8 16 24

avg waiting time=7.000000

avg tat time=13.000000

student@NNRG310:~/oslab$

### c) ROUND ROBIN CPU SCHEDULING ALGORITHM

**DESCRIPTION:**

For round robin scheduling algorithm, read the number of processes/ jobs in the system, their CPU burst times, and the size of the time slice. Time slices are assigned to each process in equal portions and in circular order, handling all processes execution. This allows every process to get an equal chance.

**ALGORITHM:**

Step 1: Start

Step 2: Define a structure process with elements st,bt,wt,tat,n,tq

Step 3: Read i,n.tq

Step 4: Read the Processes details n & bt

Step 5: for i=0 to i<n do st[i]=bt[i]

Step 6: for i=0,count=0;i<n; do till step 7

Step 7: check if(st[i]>tq)

```
st[i]=st[i]-tq;
else
if(st[i]>=0)
{
temp=st[i];
st[i]=0;
}
sq=sq+temp;
tat[i]=sq;
```

Step 8: if (n= =count) break;

Step 9: else wt[i]=tat[i]-bt[i];

```
avgwt=avgwt+wt[i];
avgtat=avgtat+tat[i];
```

Step 10: Print the output with the RoundRobin Fashion and Calculating Pid,bt,wt,&tat

Step 11: End

**PROGRAM:** ROUND ROBIN CPU SCHEDULING ALGORITHM

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
int p[10],st[10],bt[10],wt[10],tat[10],n,tq;
int i,count=0,temp,sq=0;
float avgwt=0.0,avgtat=0.0;
system("clear");
printf("Enter number of processes:\t");
scanf("%d",&n);

for(i=0;i<n;i++)
{
printf("enter process number:\t");
scanf("%d",&p[i]);
printf("enter burst time:\t");
scanf("%d",&bt[i]);
st[i]=bt[i];
}

printf("Enter time quantum:");
scanf("%d",&tq);
while(1)
{
for(i=0,count=0;i<n;i++)
{
temp=tq;
if(st[i]==0)
{
count++;
```

```
continue;
}
if(st[i]>tq)
st[i]=st[i]-tq;
else
if(st[i]>=0)
{
temp=st[i];
st[i]=0;
}
sq=sq+temp;
tat[i]=sq;
}
if(n==count)
break;
}
for(i=0;i<n;i++)
{
wt[i]=tat[i]-bt[i];
avgwt=avgwt+wt[i];
avgtat=avgtat+tat[i];
}
printf("P_NO\t B_T\t W_T\t TAT\n");
for(i=0;i<n;i++)
printf("%d\t %d\t %d\t %d\t\n",i+1,bt[i],wt[i],tat[i]);
printf("Avg wait time is %f\n Avg turn around time is %f\n",avgwt/n,avgtat/n);
}
```

**OUTPUT:**

```
student@NNRG310:~/oslab$ cc rr.c
student@NNRG310:~/oslab$ ./a.out
Enter number of processes:    3
enter process number:   1
enter burst time: 24
enter process number:   2
enter burst time: 3
enter process number:   3
enter burst time: 3
Enter time quantum:4
P_NO  B_T   W_T   TAT
1     24    6     30
2     3     4     7
3     3     7     10
Avg wait time is 5.666667
Avg turn around time is 15.666667
student@NNRG310:~/oslab$
```

### d) PRIORITY CPU SCHEDULING ALGORITHM

**DESCRIPTION:**

For priority scheduling algorithm, read the number of processes/jobs in the system, their CPU burst times, and the priorities. Arrange all the jobs in order with respect to their priorities. There may be two jobs in queue with the same priority, and then FCFS approach is to be performed. Each process will be executed according to its priority. Calculate the waiting time and turnaround time of each of the processes accordingly.

**ALGORITHM:**

```
Step1:    Start
Step2:    Define a structure process with elements p,bt,wt,tatSte
Step3:    Read the Processes details pid, & bt
Step4:    for i=0 to i<n do still step5
Step5:    for j=0 to j<n do till step 6
Step6:    if(pr[i]>pr[j])
          temp=p[i];
          p[i]=p[j];
          p[j]=temp;
          temp=bt[i];
          bt[i]=bt[j];
          bt[j]=temp;
          temp=pr[i];
          pr[i]=pr[j];
          pr[j]=temp;
Step7:    initialize avgwt=wt[0]=0;
          avgtat=tat[0]=bt[0];
Step8:    for i=1;i<n do till step 9
Step9:    wt[i]=wt[i-1]+bt[i-1];
          tat[i]=wt[i]+bt[i];
```

avgwt=avgwt+wt[i];

avgtat=avgtat+tat[i];

Step10: Print the output with the FCFS Fashion and Calculating Pid,bt,wt,&tat

Step11 : End

**PROGRAM:** PRIORITY CPU SCHEDULING ALGORITHM

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
int i,j,n,temp;
int p[10],pr[10],bt[10],wt[10],tat[10];
float avgtat,avgwt;
system ("clear");
printf("enter no of processes:\t");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("enter process number:\t");
scanf("%d",&p[i]);
printf("enter burst time:\t");
scanf("%d",&bt[i]);
printf("enter priority:\t");
scanf("%d",&pr[i]);
}
for(i=0;i<n;i++)
{
for(j=i+1;j<n;j++)
{
if(pr[i]<pr[j])
```

```
{
temp=p[i];
p[i]=p[j];
p[j]=temp;
temp=bt[i];
bt[i]=bt[j];
bt[j]=temp;
temp=pr[i];
pr[i]=pr[j];
pr[j]=temp;
}
}
}
avgwt=wt[0]=0;
avgtat=tat[0]=bt[0];
for(i=1;i<n;i++)
{
wt[i]=wt[i-1]+bt[i-1];
tat[i]=wt[i]+bt[i];
avgwt=avgwt+wt[i];
avgtat=avgtat+tat[i];
}
printf("p_name\t B_time\t w_time\t turnarounftime\n");
for(i=0;i<n;i++)
printf("%d\t%d\t%d\t%d\n",p[i],bt[i],wt[i],tat[i]);
printf("\navg waiting time=%f\n", avgwt/n);
printf("avg tat time=%f\n", avgtat/n);
}
```

**OUTPUT:**

```
student@NNRG310:~/oslab$ cc priority.c
student@NNRG310:~/oslab$ ./a.out
enter no of processes:    5
enter process number:   1
enter burst time:  10
enter  priority:      3
enter process number:   2
enter burst time:  1
enter  priority:      1
enter process number:   3
enter burst time:  2
enter  priority:      4
enter process number:   4
enter burst time:   1
enter  priority:      5
enter process number:   5
enter burst time:  5
enter  priority:      2
p_name        B_time        w_time        turnarounftime
4        1        0        1
3        2        1        3
1        10       3        13
5        5        13       18
2        1        18       19
```

avg waiting time=7.000000

avg tat time=10.800000

student@NNRG310:~/oslab$

**2. Write programs using the I/O system calls of UNIX/LINUX operating system**

**a) open ( ) system call**

**DESCRIPTION:**

Used to open the file for reading, writing or both. This function returns the file descriptor or in case of an error -1. The number of arguments that this function can have is two or three. The third argument is used only when creating a new file. When we want to open an existing file only two arguments are used.

**PROGRAM:** using open ( ) system call

```
//using open() system call
#include<stdio.h>
#include<fcntl.h>
#include<errno.h>
extern int errno;
int main()
{
int fd=open("f3.txt",O_RDONLY | O_CREAT);
printf("fd=%d\n",fd);
if (fd==-1)
{
printf("error Number %d\n",errno);
perror("program");
}
return 0;
}
```

**OUTPUT:**

```
student@NNRG310:~/oslab$ cc open.c
student@NNRG310:~/oslab$ ./a.out
fd=3
```

**b) read ( ) system call**

**DESCRIPTION:**

size_t read (int fd, void* buf, size_t cnt);

From the file indicated by the file descriptor fd, the read() function reads cnt bytes of input into the memory area indicated by buf. A successful read() updates the access time for the file.

**PROGRAM:** using read ( ) system call

```
// read system Call read.c file
#include<stdio.h>
#include <fcntl.h>
#include<stdlib.h>
#include <unistd.h>
int main()
{
  int fd,sz;
  char *c = (char *) calloc(100, sizeof(char));

  fd = open("f3.txt", O_RDONLY);
  if (fd==-1)
   {
   perror("r1");
   exit(1);
   }
  sz=read(fd,c,13);
  printf("called read(%d, c, 10). returned that" " %d bytes were read.\n",
        fd, sz);
  c[sz] = '\0';
  printf("Those bytes are as follows: %s\n", c);
  return 0; }
```

**OUTPUT:**

ca> f3.txt

From the file indicated by the file descriptor fd, the read() function reads cnt bytes of input into the memory area indicated by buf.

student@NNRG310:~/oslab$ cc read.c

student@NNRG310:~/oslab$ ./a.out

called read(3, c, 10). returned that 13 bytes were read.

Those bytes are as follows: From the file

**c) write ( ) system call**

**DESCRIPTION:**

size_t write (int fd, void* buf, size_t cnt);
Writes cnt bytes from buf to the file or socket associated with fd. If cnt is zero, write ( ) simply returns 0 without attempting any other action.

**PROGRAM:** using write ( ) system call

```
// C program to illustrate
// write system Call
#include<stdio.h>
#include <fcntl.h>
#include<stdlib.h>
#include <unistd.h>
#include<string.h>
int main( )
{
  int sz;

  int fd = open("f4.txt", O_WRONLY | O_CREAT | O_TRUNC, 0644);
  if (fd ==-1)
  {
     perror("r1");
     exit(1);
  }
  sz = write(fd, "hello linux", strlen("hello linux"));
 printf("called write(%d, \"hello linux\", %d). It returned %d\n", fd,
         strlen("hello linux"), sz);
  close(fd);
  return 0;
}
```

**OUTPUT:**

```
student@NNRG310:~/oslab$ cc write.c
student@NNRG310:~/oslab$ ./a.out
called write(3, "hello linux", 11). It returned 11
student@NNRG310:~/oslab$ cat f4.txt
hello linux
```

**d) close ( ) system call**

**DESCRIPTION:**

int close(int fd);

Tells the operating system you are done with a file descriptor and Close the file which pointed by fd.

**PROGRAM:** using close ( ) system call

```
// C program to illustrate close system Call
#include<stdio.h>
#include <fcntl.h>
#include<stdlib.h>
#include <unistd.h>
int main()
{
    int fd1 = open("f3.txt", O_RDONLY);
    if (fd1==-1)
    {
      perror("c1");
      exit(1);
    }
    printf("opened the fd = % d\n", fd1);
    // Using close system  Call
    if (close(fd1)==-1)
    {
      perror("c1");
      exit(1);
```

```
}
printf("closed the fd.\n");
return 0;
}
```

**OUTPUT:**

```
student@NNRG310:~/oslab$ ./a.out
opened the fd = 3
closed the fd.
```

**e) fcntl ( ) system call**

**DESCRIPTION:**

The fcntl system call is the access point for several advanced operations on file descriptors. The first argument to fcntl is an open file descriptor, and the second is a value that indicates which operation is to be performed. For some operations, fcntl takes an additional argument. We'll describe here one of the most useful fcntl operations, file locking

**PROGRAM:** using fcntl ( ) system call

```
#include <fcntl.h>

#include <stdio.h>

#include <string.h>

#include <unistd.h>


int main (int argc, char* argv[])

{

 char* file = argv[1];

 int fd;

 struct flock lock;

 printf ("opening %s\n", file);

 /* Open a file descriptor to the file. */

 fd = open (file, O_WRONLY);

 printf ("locking\n");

 /* Initialize the flock structure. */

 memset (&lock, 0, sizeof(lock));

 lock.l_type = F_WRLCK;
```

```
 /* Place a write lock on the file. */
 fcntl (fd, F_SETLKW, &lock);
 printf ("locked; hit Enter to unlock... ");
 /* Wait for the user to hit Enter. */
 getchar ();
 printf ("unlocking\n");
 /* Release the lock. */
 lock.l_type = F_UNLCK;
 fcntl (fd, F_SETLKW, &lock);

 close (fd);
 return 0;
}
```

**OUTPUT:**

Terminal-1

```
student@NNRG310:~/oslab$ cc lock.c
student@NNRG310:~/oslab$ ./a.out f4.txt
opening f4.txt
locking
locked; hit Enter to unlock...
unlocking
student@NNRG310:~/oslab$
```

Terminal-2

```
student@NNRG310:~/oslab$ ./a.out f4.txt
opening f4.txt
locking
locked; hit Enter to unlock...
```

**f ) seek ( ) system call**

**DESCRIPTION:**

The lseek() function allows the file offset to be set beyond the end of the file (but this does not change the size of the file). If data is later written at this point, subsequent reads of the data in the gap (a "hole") return null bytes ('\0') until data is actually written into the gap.

**PROGRAM:** using seek ( ) system call

```
#include <unistd.h>

#include <fcntl.h>

#include <sys/types.h>

#include<stdio.h>

int main()

{

        int file=0;

        if((file=open("f4.txt",O_RDONLY)) < -1)

                return 1;

        char buffer[19];

        if(read(file,buffer,19) != 19) return 1;

        printf("%s\n",buffer);

        if(lseek(file,10,SEEK_SET) < 0) return 1;

        if(read(file,buffer,19) != 19) return 1;

        printf("%s\n",buffer);

        return 0;

}
```

**OUTPUT:**

Cat> f4.txt

lseek is a system call that is used to change the location of the read/write pointer of a file descriptor. The location can be set either in absolute or relative terms.

student@NNRG310:~/oslab$ cc lseek.c

student@NNRG310:~/oslab$ ./a.out

lseek is a system c

system call that i

student@NNRG310:~/oslab$

**g ) stat ( ) system call**

**DESCRIPTION:**

int stat(const char *path, struct stat *buf);

These functions return information about a file. No permissions are required on the file itself, but — in the case of stat() and lstat() — execute (search) permission is required on all of the directories in path that lead to the file.

**PROGRAM:** using stat ( ) system call

```
#include <unistd.h>
#include <stdio.h>
#include <sys/stat.h>
#include <sys/types.h>
 int main(int argc, char **argv)
{
    if(argc!=2)
        return 1;
    struct stat fileStat;
    if(stat(argv[1],&fileStat) < 0)
        return 1;
    printf("Information for %s\n",argv[1]);
    printf("...............................................-\n");
    printf("File Size: \t\t%ld bytes\n",fileStat.st_size);
    printf("Number of Links: \t%d\n",fileStat.st_nlink);
    printf("File inode: \t\t%lu\n",fileStat.st_ino);
```

```
    printf("File Permissions: \t");
    printf( (S_ISDIR(fileStat.st_mode)) ? "d" : "-");
    printf( (fileStat.st_mode & S_IRUSR) ? "r" : "-");
    printf( (fileStat.st_mode & S_IWUSR) ? "w" : "-");
    printf( (fileStat.st_mode & S_IXUSR) ? "x" : "-");
    printf( (fileStat.st_mode & S_IRGRP) ? "r" : "-");
    printf( (fileStat.st_mode & S_IWGRP) ? "w" : "-");
    printf( (fileStat.st_mode & S_IXGRP) ? "x" : "-");
    printf( (fileStat.st_mode & S_IROTH) ? "r" : "-");
    printf( (fileStat.st_mode & S_IWOTH) ? "w" : "-");
    printf( (fileStat.st_mode & S_IXOTH) ? "x" : "-");
    printf("\n\n");
    return 0;
}
```

**OUTPUT:**

```
student@NNRG310:~/oslab$ cc stat.c
student@NNRG310:~/oslab$ ./a.out read.c
Information for read.c
---------------------------
File Size:          455 bytes
Number of Links: 1
File inode:         794292
File Permissions:  -rw-rw-r--
```

**h ) opendir ( ), readdir ( )system calls**

**DESCRIPTION:**

The opendir( ) function opens a directory stream corresponding to the directory named by the d_name argument. The directory stream is positioned at the first entry. The readdir( ) function returns a pointer to a structure representing the directory entry at the current position in the directory stream specified by the argument dir, and positions the directory stream at the next entry

**PROGRAM:** using opendir ( ), readdir ( ) system calls

```
#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:**

```
student@NNRG310:~/oslab$ cc dirsystemcalls.c
student@NNRG310:~/oslab$ ./a.out
                    f1.txt
                    f2.txt
                    .
                    fcfs.c
                    system calls programs.docx
                    sjf.c
                    a.out
                    .~lock.system calls programs.docx#
                    read.c
                    write.c
                    close.c
                    rr.c
                    dirsystemcalls.c
                    priority.c
                    f4.txt
                    f3.txt
                    ..
                    open.c
```

**3. Write a C program to simulate Bankers Algorithm for Deadlock Avoidance and Prevention.**

**ALGORITHM:**

Safety Algorithm

1. Work and Finish be the vector of length m and n respectively, Work=Available and Finish[i] =False.
2. Find an i such that both
   a. Finish[i] =False
   b. Need<=Work

   If no such I exists go to step 4.
3. work=work+Allocation, Finish[i] =True;
4. if Finish[1]=True for all I, then the system is in safe state.

**PROGRAM:**

```
#include<stdio.h>
int main()
 {
 int process,resource,instance,j,i,k=0,count1=0,count2=0;
 int avail[10] , max[10][10], allot[10][10],need[10][10],completed[10];

   printf("\n\t\t Enter No. of Process: ");
   scanf("%d",&process);
   printf("\n\t\tEnter No. of Resources: ");
   scanf("%d",&resource);
   for(i=0;i<process;i++)
   completed[i]=0;
   printf("\n\t Enter No. of Available Instances: ");
   for(i=0;i<resource;i++)
    {
```

```
     scanf("%d",&instance);
     avail[i]=instance;
    }
   printf("\n\tEnter Maximum No. of instances of resources that a
Process need:\n");
   for(i=0;i<process;i++)
    {
     printf("\n\t For P[%d]",i);
     for(j=0;j<resource;j++)
         {
          printf("\t");
          scanf("%d",&instance);
          max[i][j]=instance;
         }
   }
    printf("\n\t Enter no. of instances already allocated to process of a
resource:\n");
    for(i=0;i<process;i++)
     {
      printf("\n\t For P[%d]\t",i);
      for(j=0;j<resource;j++)
         {
            scanf("%d",&instance);
            allot[i][j]=instance;
            need[i][j]=max[i][j]-allot[i][j];      //calculating Need of each
process
         }   }
   printf("\n\n \t Safe Sequence is:- \t");
    while(count1!=process)
    {
    count2=count1;
    for(i=0;i<process;i++)
```

```
    {
      for(j=0;j<resource;j++)
        {
          if(need[i][j]<=avail[j])
           {
               k++;
           }
        }
if(k==resource  &&  completed[i]==0  )
        {
          printf("P[%d]\t",i);
          completed[i]=1;
          for(j=0;j<resource;j++)
           {
             avail[j]=avail[j]+allot[i][j];
           }
           count1++;
         }
        k=0;        }

 if(count1==count2)
        {
        printf("\t\t Stop ..After this.....Deadlock \n");
        break;

 }
 }
   return 0;
}
```

**OUTPUT:**

```
student@NNRG310:~/oslab$ cc rr.c
student@NNRG310:~/oslab$ ./a.out
Enter number of processes:    3
enter process number:   1
enter burst time: 24
enter process number:   2
enter burst time: 3
enter process number:   3
enter burst time: 3
Enter time quantum:4
P_NO  B_T   W_T   TAT
1     24    6     30
2     3     4     7
3     3     7     10
Avg wait time is 5.666667
Avg turnaround time is 15.666667
student@NNRG310:~/oslab$
```

**4. Write a C program to implement the Producer - Consumer problem using semaphores using UNIX/LINUX system calls**

**DESCRIPTION:**

The producer consumer problem is a synchronization problem. There is a fixed size buffer and the producer produces items and enters them into the buffer. The consumer removes the items from the buffer and consumes them. A producer should not produce items into the buffer when the consumer is consuming an item from the buffer and vice versa. So the buffer should only be accessed by the producer or consumer at a time.

**PROGRAM:** Producer – Consumer problem using semaphores

```
#include<stdio.h>
#include<stdlib.h>
int mutex=1,full=0,empty=3,x=0;
int main()
{
        int n;
        void producer();
        void consumer();
        int wait(int);
        int signal(int);
        printf("\n1.Producer\n2.Consumer\n3.Exit");
        while(1)
        {
                printf("\nEnter your choice:");
                scanf("%d",&n);
```

```
        switch(n)
        {
            case 1:     if((mutex==1)&&(empty!=0))
                            producer();
                        else
                            printf("Buffer is full!!");
                        break;
            case 2:     if((mutex==1)&&(full!=0))
                            consumer();
                        else
                            printf("Buffer is empty!!");
                        break;
            case 3:
                        exit(0);
                        break;
        }
    }

    return 0;
}

int wait(int s)
{
    return (--s);
}
```

```
int signal(int s)
{
        return(++s);
}
void producer()
{
        mutex=wait(mutex);
        full=signal(full);
        empty=wait(empty);
        x++;
        printf("\nProducer produces the item %d",x);
        mutex=signal(mutex);
}
void consumer()
{
        mutex=wait(mutex);
        full=wait(full);
        empty=signal(empty);
        printf("\nConsumer consumes item %d",x);
        x--;
        mutex=signal(mutex);
}
```

**OUTPUT:**

```
student@NNRG310:~/oslab$  cc  pc.c
student@NNRG310:~/oslab$   ./a.out
1.Producer
2.Consumer
3.Exit
Enter your choice:1
Producer produces the item 1
Enter your choice:2
Consumer consumes item 1
Enter your choice:2
Buffer is empty!!
Enter your choice:1
Producer produces the item 1
Enter your choice:1
Producer produces the item 2
Enter your choice:2
Consumer consumes item 2
Enter your choice:2
Consumer consumes item 1
Enter your choice:2
Buffer is empty!!
Enter your choice:3
student@NNRG310:~/oslab$
```

**5. Write C programs to illustrate the following IPC mechanisms**

**a) Pipes b) FIFOs c) Message Queues d) Shared Memory**

**a) Pipes**

**DESCRIPTION:**

Pipe is a communication medium between two or more related or interrelated processes. It can be either within one process or a communication between the child and the parent processes. Communication can also be multi-level such as communication between the parent, the child and the grand-child, etc. Communication is achieved by one process writing into the pipe and other reading from the pipe. To achieve the pipe system call, create two files, one to write into the file and another to read from the file.

**PROGRAM:**

```
#include<stdio.h>
#include<unistd.h>
int main() {
  int pipefds[2];
  int returnstatus;
  char writemessages[2][20]={"Hi", "Hello"};
  char readmessage[20];
  returnstatus = pipe(pipefds);
  if (returnstatus == -1) {
    printf("Unable to create pipe\n");
    return 1;
  }
  printf("Writing to pipe - Message 1 is %s\n", writemessages[0]);
  write(pipefds[1], writemessages[0], sizeof(writemessages[0]));
```

```
    read(pipefds[0], readmessage, sizeof(readmessage));
    printf("Reading from pipe â€“ Message 1 is %s\n", readmessage);
    printf("Writing to pipe - Message 2 is %s\n", writemessages[1]);
    write(pipefds[1], writemessages[1], sizeof(writemessages[0]));
    read(pipefds[0], readmessage, sizeof(readmessage));
    printf("Reading from pipe â€“ Message 2 is %s\n", readmessage);
    return 0;
}
```

**OUTPUT:**

```
student@NNRG310:~/oslab$ cc pipe.c
student@NNRG310:~/oslab$ ./a.out
Writing to pipe - Message 1 is  Hi
Reading from pipe – Message 1 is Hi
Writing to pipe - Message 2 is Hello
Reading from pipe – Message 2 is Hello
student@NNRG310:~/oslab$
```

**b) FIFOs**

**DESCRIPTION:**

Pipes were meant for communication between related processes. Can we use pipes for unrelated process communication, say, we want to execute client program from one terminal and the server program from another terminal? The answer is No. Then how can we achieve unrelated processes communication, the simple answer is Named Pipes. Even though this works for related processes, it gives no meaning to use the named pipes for related process communication.

**PROGRAM:**

**<u>fifoclient.c</u>**

```
#include<stdio.h>
#include<fcntl.h>
#include<stdlib.h>
#include<unistd.h>
int main()
{
 FILE *file1;
 int fifo_server,fifo_client;
 char str[256];
 char  *buf;
 int choice=1;
 printf("Choose the request to be sent to server from options below");
 printf("\n\t\t Enter 1 for O.S.Name \n \
               Enter 2 for Distribution \n \
               Enter 3 for Kernel version \n");
 scanf("%d",&choice);
fifo_server=open("fifo_server",O_RDWR);
if(fifo_server < 0) {
  printf("Error in opening file");
```

```
 exit(-1);
 }

write(fifo_server,&choice,sizeof(int));

fifo_client=open("fifo_client",O_RDWR);

if(fifo_client < 0) {
 printf("Error in opening file");
 exit(-1);
 }

buf=malloc(10*sizeof(char));
read (fifo_client,buf,10*sizeof(char));
printf("\n ***Reply from server is %s***\n",buf);
close(fifo_server);
close(fifo_client);
return 0;
}
```

**fifoserver.c**

**PROGRAM:**

```
#include<stdio.h>
#include<fcntl.h>
#include<unistd.h>
int main( )
{
FILE *file1;
int fifo_server,fifo_client;
int choice;
char *buf;
fifo_server = open("fifo_server",O_RDWR);
if(fifo_server<1) {
 printf("Error opening file");
 }
read(fifo_server,&choice,sizeof(int));
sleep(10);
fifo_client = open("fifo_client",O_RDWR);
if(fifo_server<1) {
printf("Error opening file");
 }
switch(choice) {
case 1:
 buf="Linux";
 write(fifo_client,buf,10*sizeof(char));
 printf("\n Data sent to client \n");
 break;
case 2:

 buf="Fedora";
 write(fifo_client,buf,10*sizeof(char));
```

```
 printf("\nData sent to client\n");
 break;
case 3:
 buf="2.6.32";
 write(fifo_client,buf,10*sizeof(char));
 printf("\nData sent to client\n");
}
close(fifo_server);
close(fifo_client);
}
```

**OUTPUT:**

**TERMINAL-I**

```
student@NNRG310:~/oslab$ cc fifoclient.c
student@NNRG310:~/oslab$ ./a.out
Choose the request to be sent to server from options below
Enter 1 for O.S.Name
Enter 2 for Distribution
Enter 3 for Kernel version
1
***Reply from server is Linux***
```

**TERMINAL-II**

```
student@NNRG310:~/oslab$ cc  fifoserver.c
student@NNRG310:~/oslab$ ./a.out
Data sent to client
```

c) **Message Queues:**

**DESCRIPTION:**

A message queue is a linked list of messages stored within the kernel and identified by a message queue identifier. A new queue is created or an existing queue opened by msgget().New messages are added to the end of a queue by msgsnd(). Every message has a positive long integer type field, a non-negative length, and the actual data bytes (corresponding to the length), all of which are specified to msgsnd() when the message is added to a queue. Messages are fetched from a queue by msgrcv(). We don’t have to fetch the messages in a first-in, first-out order. Instead, we can fetch messages based on their type field.

**PROGRAM:**

**Sender.c**

```
#include<stdio.h>
#include<sys/ipc.h>
#include<sys/msg.h>
#include<sys/types.h>
#include<stdlib.h>
#define SIZE 2000
void main()
{
int mfd,mfd2,mfd3;
struct
{
double mtype;
char mtext[2000];
}s1,s2,s3;
if((mfd=msgget(1000,IPC_CREAT|0666))==-1)
{
perror("msgget:");
exit(1);
}
s1.mtype=1;
sprintf(s1.mtext,"%s","Hi friends... My name is message1");
if(msgsnd(mfd,&s1,1000,0)==-1)
```

```
{
perror("msgsnd");
exit(1);
}
if((mfd2=msgget(1000,IPC_CREAT|0666))==-1)
{
perror("msgget:");
exit(1);
}
s2.mtype=1;
sprintf(s2.mtext,"%s","Hi friends... My name is message2");
if(msgsnd(mfd2,&s2,1000,0)==-1)
{
perror("msgsnd");
exit(1);
}

if((mfd3=msgget(1000,IPC_CREAT|0666))==-1)
{
perror("msgget:");
exit(1);
}
s3.mtype=1;
sprintf(s3.mtext,"%s","Hi friends... My name is message3");
if(msgsnd(mfd3,&s3,1000,0)==-1)
{
perror("msgsnd");
exit(1);
}
printf("Your message has been sent successfully...\n");
printf("Please visit another (receiver's) terminal...\n");
printf("Thank you.... For using LINUX\n");
}
```

**Output:**

```
student@NNRG310:~/oslab$  cc mqsender.c
student@NNRG310:~/oslab$ ./a.out
Your message has been sent successfully...
Please visit another (receiver's) terminal...
Thank you.... For using LINUX
student@NNRG310:~/oslab$
```

**Receiver.c**

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/ipc.h>
#include<sys/msg.h>
#include<sys/types.h>
#define SIZE 40
void main()
{
int mfd,mfd2,mfd3;
struct
{
long mtype;
char mtext[6];
}s1,s2,s3;
if((mfd=msgget(1000,0))==-1)
{
perror("msgget");
exit(1);
}
if(msgrcv(mfd,&s1,SIZE,0,IPC_NOWAIT|MSG_NOERROR)==-1)
{
perror("msgrcv");
exit(1);
}
printf("Message from client is :%s\n",s1.mtext);
if((mfd2=msgget(1000,0))==-1)
{
perror("msgget");
exit(1);
}
if(msgrcv(mfd2,&s2,SIZE,0,IPC_NOWAIT|MSG_NOERROR)==-1)
{
perror("msgrcv");
exit(1);
}
printf("Message from client is :%s\n",s2.mtext);
if((mfd3=msgget(1000,0))==-1)
{
perror("msgget");
```

```
exit(1);
}
if(msgrcv(mfd3,&s3,SIZE,0,IPC_NOWAIT|MSG_NOERROR)==-1)
{
perror("msgrcv");
exit(1);
}
printf("Message from sender is :%s\n",s3.mtext);
}
```

**Output:**

```
student@NNRG310:~/oslab$ cc mqclient.c

student@NNRG310:~/oslab$ ./a.out

Message from client is :Hi friends... My name is message1

Message from client is :Hi friends... My name is message2

Message from client is :Hi friends... My name is message3

student@NNRG310:~/oslab$
```

### d) Shared Memory:

**DESCRIPTION:**

Inter Process Communication through shared memory is a concept where two or more process can access the common memory. And communication is done via this shared memory where changes made by one process can be viewed by another process.
The problem with pipes, fifo and message queue – is that for two process to exchange information. The information has to go through the kernel.

- Server reads from the input file.
- The server writes this data in a message using either a pipe, fifo or message queue.
- The client reads the data from the IPC channel,again requiring the data to be copied from kernel's IPC buffer to the client's buffer.
- Finally the data is copied from the client's buffer.

**PROGRAM:**

shwriter.c

```
#include <stdio.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/stat.h>
int main ( )
{
  int segment_id;
  char bogus;
  char* shared_memory;
  struct shmid_ds shmbuffer;
  int segment_size;
  const int shared_segment_size = 0x6400;
```

```
  /* Allocate a shared memory segment. */
  segment_id = shmget (IPC_PRIVATE, shared_segment_size, IPC_CREAT
| IPC_EXCL | S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP);
  /* Attach the shared memory segment. */
  printf("Shared memory segment ID is %d\n", segment_id);
  shared_memory = (char*) shmat (segment_id, 0, 0);
  printf ("shared memory attached at address %p\n", shared_memory);
  /* Determine the segment's size. */
  /*
  shmctl (segment_id, IPC_STAT, &shmbuffer);
  segment_size  =               shmbuffer.shm_segsz;
  printf ("segment size: %d\n", segment_size);
  */
  /* Write a string to the shared memory segment. */
  sprintf (shared_memory, "Hello, world.");
  /* Detach the shared memory segment. */
  shmdt (shared_memory);
  printf("Wrote Hello World to the segment\n");
}
```

**PROGRAM:**

<u>shreader.c</u>

```
#include <stdio.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/stat.h>
int main ()
{
  int segment_id;
  char bogus;
  char* shared_memory;
  struct shmid_ds shmbuffer;
```

```
  int segment_size;
  const int shared_segment_size = 0x6400;
  printf("Enter the shared memory id: ");
  scanf("%d", &segment_id);
  /* Reattach the shared memory segment, at a different address. */
  shared_memory = (char*) shmat (segment_id, (void*) 0x5000000, 0);
  printf ("shared memory reattached at address %p\n", shared_memory);
  /* Print out the string from shared memory. */
  printf ("The contents of the shared memory is:\n%s\n",
shared_memory);
  /* Detach the shared memory segment. */
  shmdt (shared_memory);
  return 0;
}
```

**OUTPUT:**

**Terminal-I**

```
student@NNRG310:~/oslab$ cc shwriter.c
student@NNRG310:~/oslab$ ./a.out
Shared memory segment ID is 3047442
shared memory attached at address 0xb7f2a000
Wrote Hello World to the segment
student@NNRG310:~/oslab$
```

**Terminal-II**

```
student@NNRG310:~/oslab$ cc shreader.c
student@NNRG310:~/oslab$ ./a.out
Enter the shared memory id: 3047442
shared memory reattached at address 0x5000000
The contents of the shared memory is:
Hello, world.
student@NNRG310:~/oslab$
```

6. **Write C programs to simulate the following memory management techniques**

   **a) Paging b) Segmentation**

**a) Paging**

**PROGRAM:**

```
#include<stdio.h>
int main()
{
 int ms, ps, nop, np, rempages, i, j, x, y, pa, offset;
int s[10], fno[10][20];

 printf("\nEnter the memory size -- ");
 scanf("%d",&ms);
printf("\nEnter the page size -- ");
 scanf("%d",&ps);
nop = ms/ps;
printf("\nThe no. of pages available in memory are -- %d ",nop);
printf("\nEnter number of processes -- ");
scanf("%d",&np);
rempages = nop;
for(i=1;i<=np;i++)
 {
printf("\nEnter no. of pages required for p[%d]-- ",i);
 scanf("%d",&s[i]);
 if(s[i] >rempages)
{
 printf("\nMemory is Full");
 break;
}
rempages = rempages - s[i];
```

```
printf("\nEnter pagetable for p[%d] --- ",i);
for(j=0;j<s[i];j++)
 scanf("%d",&fno[i][j]);    }
 printf("\nEnter Logical Address to find Physical Address ");
printf("\nEnter process no. and pagenumber and offset -- ");
scanf("%d %d %d",&x,&y, &offset);

if(x>np || y>=s[i] || offset>=ps)
 printf("\nInvalid Process or Page Number or offset");
 else
{
pa=fno[x][y]*ps+offset;
printf("\nThe Physical Address is -- %d",pa);
 }
return 0;
}
```

**OUTPUT:**

```
student@NNRG310:~/oslab$ ./a.out

Enter the  memory  size  --  1000

Enter the page size -- 100

The no. of pages available in memory are -- 10

Enter number of processes -- 3

Enter no. of pages required for p[1]-- 4

Enter pagetable for p[1] --- 8  6 9 5

Enter no. of pages required for p[2]-- 5

Enter pagetable for p[2] --- 1 4 5 7 3

Enter no. of pages required for p[3]-- 5
```

Memory is Full

Enter Logical Address to find Physical Address

Enter process no. and pagenumber and offset -- 2 3 60

The Physical Address is – 760

student@NNRG310:~/oslab$

### b) Segmentation

**ALGORITHM:**

Step1 : Start the program.

Step2 : Read the base address, number of segments, size of each segment, memorylimit.

Step3 : If memory address is less than the base address display “invalid memorylimit”.

Step4 : Create the segment table with the segment number and segment address and display it.

Step5 : Read the segment number and displacement.

Step6 : If the segment number and displacement is valid compute the real address and display the same.

Step7 : Stop the program.

**PROGRAM:**

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

int main()
{
int  b[20],l[20],n,i,pa,s,a,d;
printf("\nProgram for segmentation");
printf("\nEnter the number of segments:");
scanf("%d",&n);
printf("\nEnter the base address and limit register:");
for(i=1;i<=n;i++)
{
scanf("%d",&b[i]);
scanf("%d",&l[i]);
}
printf("\nEnter the logical address:");
scanf("%d",&d);
printf("\nEnter segment number:");
scanf("%d",&s);


for(i=1;i<=n;i++)
{
if(i==s)
{
if(d<l[i])
{
```

```
pa=b[i]+d;
a=b[i];
printf("\nPageNo.\t BaseAdd. PhysicalAdd. \n %d \t %d \t %d
\n",s,a,pa);
exit(0);
}
else
{
printf("\nPage size exceeds");
exit(0);
}
}
}
printf("\nInvalid segment");
return 0;
}
```

**OUTPUT**

```
student@NNRG310:~/oslab$ cc seg123.c
student@NNRG310:~/oslab$ ./a.out

Program for segmentation
Enter the number of segments:3

Enter the base address and limit register:
100 50
150 20
130 34

Enter the logical address:25
Enter segment number:1
PageNo.      BaseAdd. PhysicalAdd.
1            100          125
student@NNRG310:~/oslab$
```

**1. Simulate the following memory allocation algorithms**

**a) First-Fit b) Best-Fit c) Worst-fit**

**a) First-Fit**

**PROGRAM:**

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
int main()
{
      static int block[10],process[10];
      int frags[10], b[10], p[10];
      int i, j, nob, nop, temp;
      printf("\nEnter the Total Number of Blocks:\t");
      scanf("%d", &nob);
      printf("Enter the Total Number of process:\t");
      scanf("%d", &nop);
      printf("\nEnter the Size of the Blocks:\n");
      for(i = 0; i < nob; i++)
      {
       printf("Block size.:\t");
       scanf("%d", &b[i]);
      }
      printf("Enter the Size of the proces:\n");
      for(i = 0; i < nop; i++)
      {
       printf("proces size\t" );
       scanf("%d", &p[i]);
      }
      for(i = 0; i < nop; i++)
      {
```

```
        for(j = 0; j < nob; j++)
        {
                if(block[j] != 1)
                {
                        temp=abs(b[j]-p[i]);
                        if(temp >= 0)
                        {
                                process[i] = j;
                                break;
                        }
                }
        }

        frags[i] = temp;
        block[process[i]] = 1;
        }
        printf("\n process Number\tBlock Number\tBlock Size\tProcess
Size\tFragment");
        for(i = 0; i < nop; i++)
        {
        printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d", i, process[i],
b[process[i]],p[i], frags[i]);
        }
        printf("\n");
        return 0;
}
```

**Output:**

student@NNRG310:~/oslab$ cc firstfit.c

student@NNRG310:~/oslab$ ./a.out

Enter the Total Number of Blocks: 5

Enter the Total Number of process: 4

Enter the Size of the Blocks:

Block size.: 100

Block size.: 500

Block size.: 200

Block size.: 300

Block size.: 600

Enter the Size of the proces:

proces size 212

proces size 417

proces size 112

proces size 426

| process Number | Block Number | Block Size | Process Size | Fragment |
|---|---|---|---|---|
| 0 | 0 | 100 | 212 | 112 |
| 1 | 1 | 500 | 417 | 83 |
| 2 | 2 | 200 | 112 | 88 |
| 3 | 3 | 300 | 426 | 126 |

student@NNRG310:~/oslab$

**b) Best-Fit**

**PROGRAM:**

```
#include<stdio.h>
int main()
{
    int frags[20],b[20],p[20],i,j,nob,nop,temp,lowest=9999;
    static int block[20],process[20];
    printf("\n\t\t\tMemory Management Scheme - Best Fit");
    printf("\nEnter the number of blocks:\t");
    scanf("%d",&nob);
    printf("Enter the number of processes:\t");
    scanf("%d",&nop);
    printf("\nEnter the size of the blocks:-\n");
    for(i=0;i<nob;i++)
    {
  printf("Block size:\t");
  scanf("%d",&b[i]);
    }
    printf("\nEnter the size of the processes :-\n");
    for(i=0;i<nop;i++)
    {
  printf("Process size:\t");
  scanf("%d",&p[i]);
    }
    for(i=0;i<nop;i++)
    {
  for(j=0;j<nob;j++)
  {
      if(block[j]!=1)
      {
           temp=b[j]-p[i];
```

```
            if(temp>=0)
                if(lowest>temp)
                {
                    process[i]=j;
                    lowest=temp;
                }
        }
    }
    frags[i]=lowest;
    block[process[i]]=1;
    lowest=10000;
     }
     printf("\nProcess_no\tProcess_size\tBlock_no\tBlock_size\tFragment");
    for(i=0;i<nop && process[i]!=0;i++)
    printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,p[i],process[i],b[process[i]],fra
gs[i]);
}
```

**Output:**

```
student@NNRG310:~/oslab$  ./a.out
                    Memory Management Scheme - Best Fit
Enter the number of blocks:    5
Enter the number of processes:        4
Enter the size of the blocks:-
Block size:   100
Block size:   500
Block size:   200
Block size:   300
Block size:   600
Enter the size of the processes:-
Process size:        212
Process size:        417
Process size:        112
Process size:        426
Process_no  Process_size        Block_no      Block_size    Fragment
       0             212        3             300           88
       1             417        1             500           83
       2             112        2             200           88
       3             426        4             600           174
student@NNRG310:~/oslab$
```

c) **Worst-fit**

**PROGRAM:**

```
#include<stdio.h>

#define max 25
int main()
{
int frags[20],b[20],p[20],i,j,nob,nop,temp,highest=0;
static int block[20],process[20];
printf("\n\tMemory Management Scheme - Worst Fit");
printf("\nEnter the number of blocks:");
scanf("%d",&nob);
printf("Enter the number of files:");
scanf("%d",&nop);
printf("\nEnter the size of the blocks:-\n");
for(i=0;i<nob;i++)
{
printf("Block size :\t");
scanf("%d",&b[i]);
}
printf("Enter the size of the processes :-\n");
for(i=0;i<nop;i++)
{
printf("File %d:\t",i);
scanf("%d",&p[i]);
}

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

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

```
{
if(block[j]!=1) //if bf[j] is not allocated
{
temp=b[j]-p[i];
if(temp>=0)
if(highest<temp)
{
process[i]=j;
highest=temp;
}
}
}
frags[i]=highest;
block[process[i]]=1;
highest=0;
}
printf("\nProcess_no:\tProcess_size
:\tBlock_no:\tBlock_size:\tFragement");
for(i=0;i<nop;i++)
printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,p[i],process[i],b[process[i]],frags
[i]);
return 0;
}
```

**Output**

```
student@NNRG310:~/oslab$ cc wrostfit.c
student@NNRG310:~/oslab$ ./a.out

  Memory Management Scheme - Worst Fit
Enter the number of blocks:5
Enter the number of files:4

Enter the size of the blocks:-
Block size :    100
Block size :    500
Block size :    200
Block size :    300
Block size :    600
Enter the size of the processes :-
File 0:  212
File 1:  417
File 2:  112
File 3:  426

Process_no:  Process_size :    Block_no:    Block_size:  Fragement
0                  212         4            600          388
1                  417         1            500          83
2                  112         3            300          188
3                  426         0            100          0
student@NNRG310:~/oslab$
```

**2. Simulate the following memory allocation algorithms**

**a) FCFS b) SCAN c) SSTF**

**a) FCFS**

**PROGRAM:**

```
#include<stdio.h>

int main()
{
 int a[20],b[20],n,i,thm[20],tot=0;
 float avgthm;

 printf("Enter head pointer position:\t");
 scanf("%d",&a[0]);
 b[0]=a[0];
 printf("\nEnter number of processes:\t");
 scanf("%d",&n);
 printf("\nEnter processes in request order\n");
 for(i=1;i<=n;i++)
 {
  scanf("%d",&a[i]);
 }
 for(i=0;i<n;i++)
 {
   thm[i]=(a[i+1]-a[i]);
  if(thm[i]<0)
  thm[i]=thm[i]*(-1);
 }
 for(i=0;i<n;i++)
 {
```

```
 b[i]=thm[i];
 tot=tot+thm[i];
 }
 avgthm=(float)tot/n;
 printf("\n\tTrack traversed \t Difference between tracks \n");
 for(i=0;i<n;i++)
 printf("\t%d\t%d\t=\t\t%d\n",a[i],a[i+1],b[i]);
 printf("\nTotal heam movents\t=%d \n Average Head
Movement\t=:%f",tot,avgthm);
 return 0;
}
```

**Output:**

```
student@NNRG310:~/oslab$ cc dfcfs.c
student@NNRG310:~/oslab$ ./a.out
Enter head pointer position:    53
Enter number of processes:      8
Enter processes in request order
98 183 37 122 14 124 65 67
  Track traversed     Difference between tracks
  53    98    =           45
  98    183   =           85
  183   37    =           146
  37    122   =           85
  122   14    =           108
  14    124   =           110
  124   65    =           59
  65    67    =           2

Total heam movents       =640
Average Head Movement =:80.000000
student@NNRG310:~/oslab$
```

**b) SCAN**

**PROGRAM:**

```
#include<stdio.h>

int main()
{
 int a[20],b[20],n,i,j,temp,p,s,m,x,t=0;

 printf("Enter head pointer position:");
 scanf("%d",&a[0]);
 s=a[0];
 printf("\nEnter previous head position:");
 scanf("%d",&p);
 printf("\nEnter max track limit:");
 scanf("%d",&m);
 printf("\nEnter number of processes:");
 scanf("%d",&n);
 printf("\nEnter processes in request order");
 for(i=1;i<=n;i++)
 {
  scanf("%d",&a[i]);
 }
 a[n+1]=m;
 a[n+2]=0;
 for(i=n+1;i>=0;i--)
 {
  for(j=0;j<=i;j++)
  {
   if(a[j]>a[j+1])
   {
    temp=a[j];
```

```
   a[j]=a[j+1];
   a[j+1]=temp;
  }
 }
}
for(i=1;i<=n+1;i++)
{
 if(s==a[i])
 x=i;
}
j=0;
if(s<p)
{
 for(i=x;i>0;i--)
 {
  t+=(a[i]-a[i-1]);
  b[j++]=a[i];
 }
 t+=a[x+1]-a[0];
 b[j++]=a[0];
 for(i=x+1;i<n+1;i++)
 {
  t+=(a[i+1]-a[i]);
  b[j++]=a[i];
 }
 b[j++]=a[i];
}
else
{
 for(i=x;i<n+2;i++)
 {
  t+=(a[i+1]-a[i]);
```

```
  b[j++]=a[i];
 }
 t+=a[n+2]-a[x-1];
 b[j++]=a[n+2];
 for(i=x-1;i>1;i--)
 {
  t+=(a[i]-a[i-1]);
  b[j++]=a[i];
 }
 b[j++]=a[i];
}
printf("\nProcessing order:");
for(i=0;i<=n+1;i++)
printf("%d->",b[i]);
printf("\n\nTotal Head Movement:%d",t);
return 0;
}
```

**Output- 01**

```
student@NNRG310:~/oslab$ cc dscan.c
student@NNRG310:~/oslab$ ./a.out
Enter head pointer position:53


Enter previous head position:60


Enter max track limit:199


Enter number of processes:8
```

Enter processes in request order

98 183 37 122 14 124 65 67

Processing order:53->37->14->0->65->67->98->122->124->183->

Total Head Movement:236

student@NNRG310:~/oslab$

**Output- 02**

student@NNRG310:~/oslab$ cc dscan.c

student@NNRG310:~/oslab$ ./a.out

Enter head pointer position:53

Enter previous head position:40

Enter max track limit:199

Enter number of processes:8

Enter processes in request order

98 183 37 122 14 124 65 67

Processing order:53->65->67->98->122->124->183->199->37->14->

Total Head Movement:331

student@NNRG310:~/oslab$

**c) SSTF**

**PROGRAM:**

```
#include<stdio.h>
struct di
{
int num;
int flag;
};
int main()
{
 int i,j,sum=0,n,min,loc,x,y;
 struct di d[20];
 int disk;
 int ar[20],a[20];

 printf("enter size of queue\t");
 scanf("%d",&n);
 printf("enter position of head\t");
 scanf("%d",&disk);
 printf("enter elements of disk queue:\t");
 for(i=0;i<n;i++)
 {
 scanf("%d",&d[i].num);   d[i]. flag=0;
 }
 for(i=0;i<n;i++)
 {                x=0; min=0;loc=0;
  for(j=0;j<n;j++)
  {
   if(d[j].flag==0)
   {
```

```
  if(x==0)
  {
  ar[j]=disk-d[j].num;
  if(ar[j]<0){ ar[j]=d[j].num-disk;}
  min=ar[j];loc=j;x++; }
  else
  {
  ar[j]=disk-d[j].num;
  if(ar[j]<0){ ar[j]=d[j].num-disk;}
   }
   if(min>ar[j]){  min=ar[j];  loc=j;}
  }
 }
    d[loc].flag=1;
    a[i]=d[loc].num-disk;
    if(a[i]<0){a[i]=disk-d[loc].num;}

    disk=d[loc].num;
}



for(i=0;i<n;i++)
{
sum=sum+a[i];
}
      printf("\nmovement of total cylinders %d",sum);

return 0;
}
```

**Output:**

```
student@NNRG310:~/oslab$  cc dsstf.c
student@NNRG310:~/oslab$ ./a.out
enter size of queue 8
enter position of head      53
enter elements of disk queue:     98 183 37 122 14 124 65 67

movement of total cylinders 236
student@NNRG310:~/oslab$
```