

GOKARAJU RANGARAJU INSTITUTE OF ENGINEERING AND TECHNOLOGY

OPERATING SYSTEMS LAB

Course Code: GR22A2079

L/T/P/C: 0/0/3/1.5

II Year II Semester

Course Objectives: The Objectives of this course is to provide the student to:

- Learn different types of CPU scheduling algorithms
- Demonstrate the usage of semaphores for solving synchronization problem
- Understand Banker's algorithm used for deadlock avoidance
- Implement paging techniques and page replacement policies, memory allocation techniques in memory management.
- Implement disk scheduling techniques and file allocation strategies.

Course Outcomes: At the end of the course, the student will be able to

- Evaluate the performance of different types of CPU scheduling algorithms
- Implement producer-consumer problem, reader-writers problem, Dining philosophers problem using semaphore
- Simulate Banker's algorithm for deadlock avoidance
- Implement paging techniques and page replacement policies, memory allocation techniques in memory management
- Implement disk scheduling techniques and file allocation strategies

GOKARAJU RANGARAJU INSTITUTE OF ENGINEERING AND TECHNOLOGY

OPERATING SYSTEMS LAB

Course Code: GR22A2079

L/T/P/C: 0/0/3/1.5

II Year II Semester

Syllabus

Task 1:

Practice the following commands in UNIX environment

- a) cp b) rm c) mv d) chmod e) ps f) kill

Task 2:

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

Task 3:

Simulate the following CPU scheduling algorithms

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

Task 4:

Simulate the Producer-Consumer Problem

Task 5:

Simulate the Readers-Writers Problem using Semaphores.

Task 6:

Simulate the Dining Philosophers Problem using Semaphores.

Task 7:

Simulate Bankers Algorithm for Dead Lock Avoidance.

Task 8:

Simulate First Fit and Best Fit algorithms for Memory Management.

Task 9:

Simulate Paging Technique of memory management.

Task 10:

Simulate all page replacement algorithms

- a) FIFO b) LRU

Task 11:

Simulate the following Disk Scheduling Algorithms

- a)FCFS b)SSTF c)SCAN d)C-SCAN e)LOOK f)CLOOK

Task 12:

Simulate all file allocation strategies

- a) Sequential b) Indexed c) Linked

Text /Reference Books:

1. Operating System Concepts- Abraham Silberchatz , Peter B. Galvin, Greg Gagne 7th Edition, John Wiley.
2. Operating Systems– Internal and Design Principles Stallings, Fifth Edition–2005, Pearson education/PHI

INDEX

S.No	LIST OF PROGRAMS	Page no
1	Practice the following commands in UNIX environment a)cp b) rm c) mv d) chmod e) ps f) kill	5
2	Write a program that makes a copy of a file using standard I/O and system calls	11
3	Simulate the following Scheduling algorithms a)Round Robin b)SJF c)FCFS d)Priority	12
4	Simulate the Producer Consumer problem	21
5	Simulate the Readers – writers problem using semaphores	24
6	Simulate the Dining Philosophers problem using semaphores	26
7	Simulate Bankers Algorithm for Deadlock Avoidance	29
8	Simulate First Fit and Best Fit algorithms for Memory Management	34
9	Simulate paging technique of memory management.	38
10	Simulate All page replacement Algorithms a)FIFO b)LRU	39
11	Simulate following Disk Scheduling algorithms a)FCFS b)SSTF c)SCAN d)C-SCAN e)LOOK f)C-LOOK	45
12	Simulate file allocation strategies a)Sequential b)Indexed c)Linked	54

TASK 1

Task: Practice the following commands in UNIX environment

- a) cp b) rm c) mv d) chmod e) ps f) kill

\$cp: used for copying files.

Syntax: \$cp [options] source_file destination-file

Example: \$cp f1 f2

OUTPUT:

```
$cat f1
```

This is GRIET

```
$cat f2
```

This is GRIET

It will copy the contents of f1 to f2

Options:

- a)-f: Force copy by removing the destination file if needed.

Syntax: \$cp -f source_file destination-file

Example: \$cp -f f1 f2

OUTPUT: \$cat f1

This is CSE

```
$cat f2
```

This is GRIET

- b)-i: Ask the confirmation to overwrite.

Syntax: \$cp -i source_file destination-file

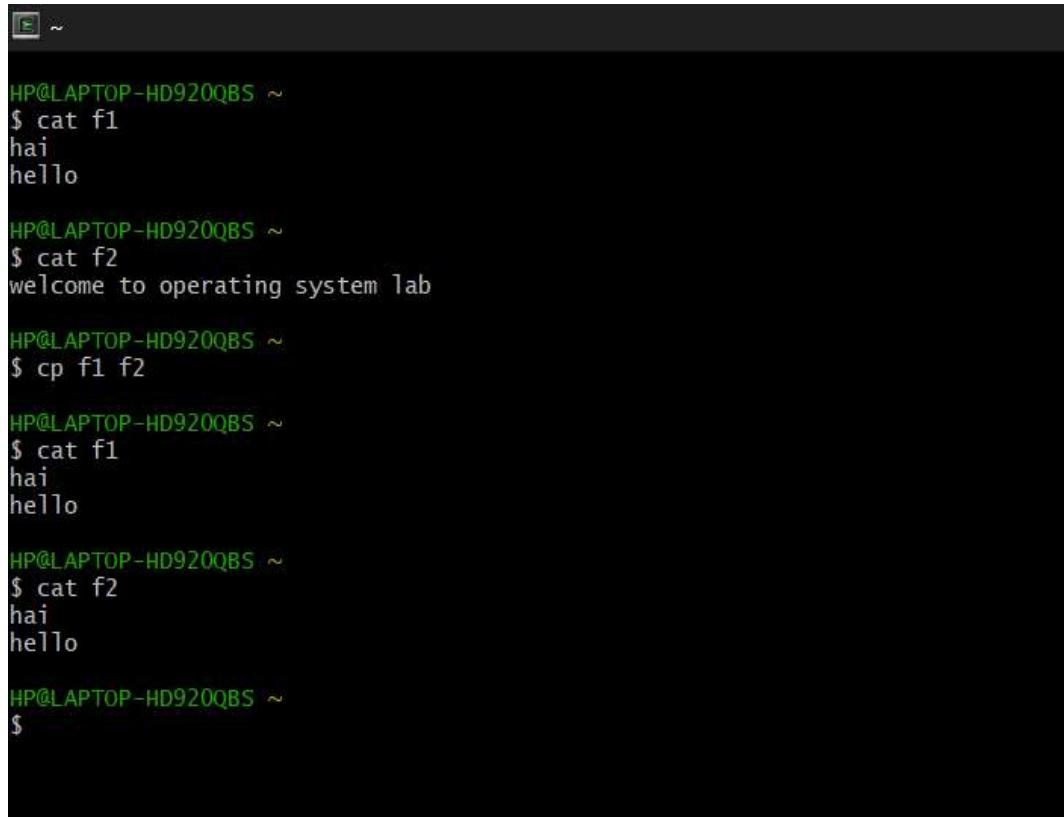
Example: \$cp -i f1 f2

- c)-b: It creates backup files before overriding.

Syntax: \$cp -b source_file destination-file

Example:\$cp -b f1 f2

OUTPUT:



A terminal window titled 'HP@LAPTOP-HD920QBS ~' showing the execution of the cp command. It displays the contents of file f1, copies it to f2, and then shows the contents of both f1 and f2.

```
HP@LAPTOP-HD920QBS ~
$ cat f1
hai
hello

HP@LAPTOP-HD920QBS ~
$ cat f2
welcome to operating system lab

HP@LAPTOP-HD920QBS ~
$ cp f1 f2

HP@LAPTOP-HD920QBS ~
$ cat f1
hai
hello

HP@LAPTOP-HD920QBS ~
$ cat f2
hai
hello

HP@LAPTOP-HD920QBS ~
$
```

\$rm: Used to remove files (or) directories

Syntax: \$rm [options] filename

Example:\$rm f1

OUTPUT:

f1 is deleted

Options:

a)-f: ignores non existing files, never prompt

Syntax: \$rm -f filename

Example: \$rm -f myfile.txt

OUTPUT: Removes file myfile.txt

b)-r: Removes all files in directory and directory itself

Syntax: \$rm -r filename

Example: \$rm -r mydirectory

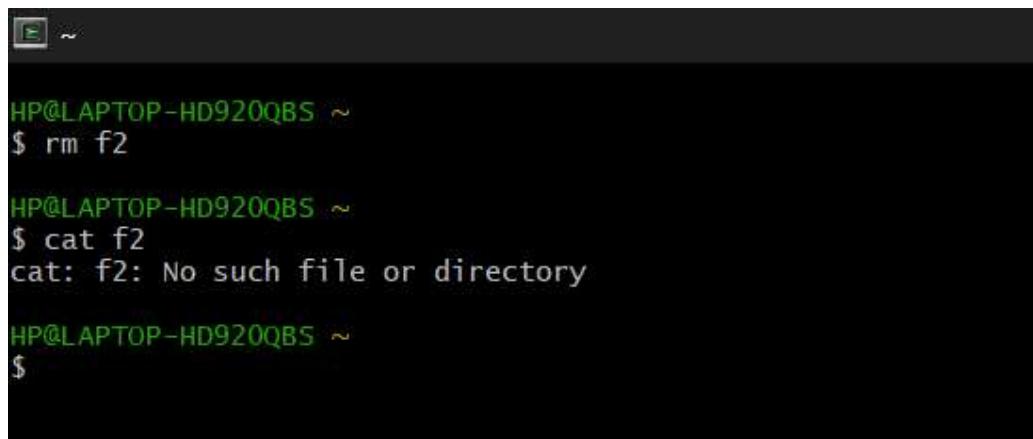
OUTPUT: Removes directory mydirectory and all files in it.

c)-i: prompts before every removal.

Syntax: \$rm -i filename

Example: \$rm -i bak.c

OUTPUT:



A terminal window titled 'E ~' showing the following command execution:

```
HP@LAPTOP-HD920QBS ~
$ rm f2

HP@LAPTOP-HD920QBS ~
$ cat f2
cat: f2: No such file or directory

HP@LAPTOP-HD920QBS ~
$
```

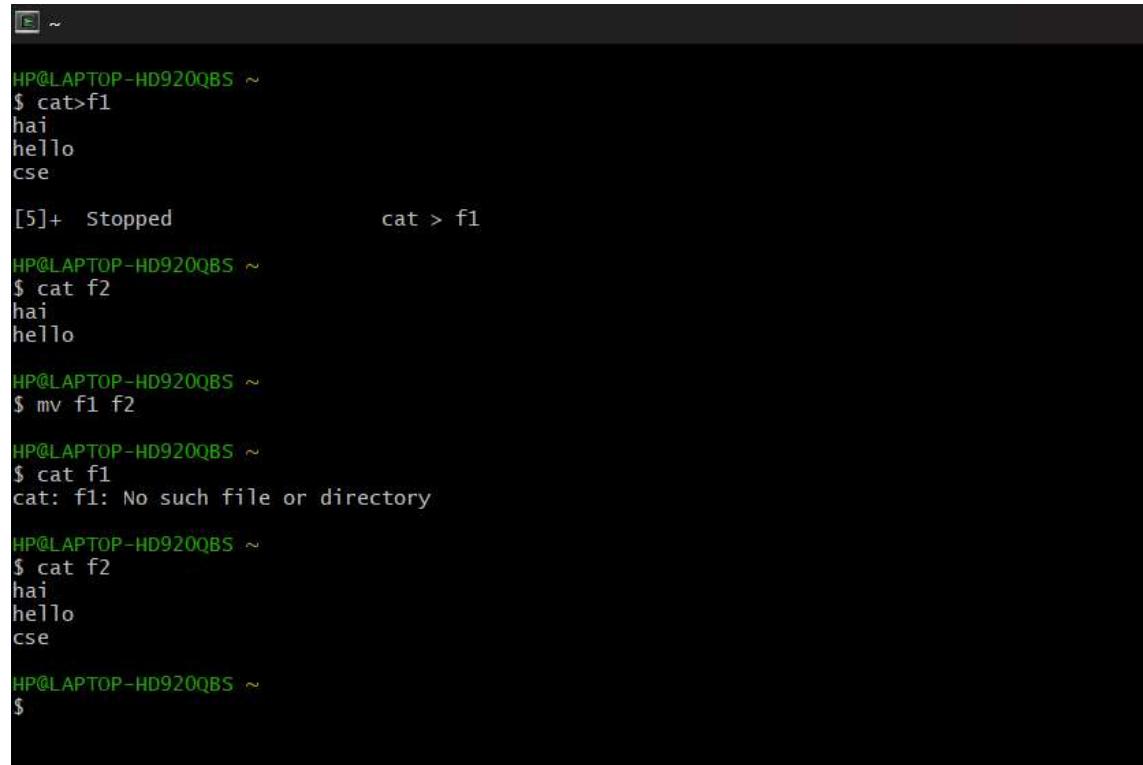
\$mv: mv stands for move. mv is used to move one or more files or directories from one place to another in file system like UNIX. It has two distinct functions:

- (i) It rename a file or folder.
- (ii) It moves group of files to different directory.

No additional space is consumed on a disk during renaming. This command normally works silently means no prompt for confirmation.

Syntax:

mv [Option] source destination



```
HP@LAPTOP-HD920QBS ~
$ cat>f1
hai
hello
cse
[5]+  Stopped                  cat > f1

HP@LAPTOP-HD920QBS ~
$ cat f2
hai
hello

HP@LAPTOP-HD920QBS ~
$ mv f1 f2

HP@LAPTOP-HD920QBS ~
$ cat f1
cat: f1: No such file or directory

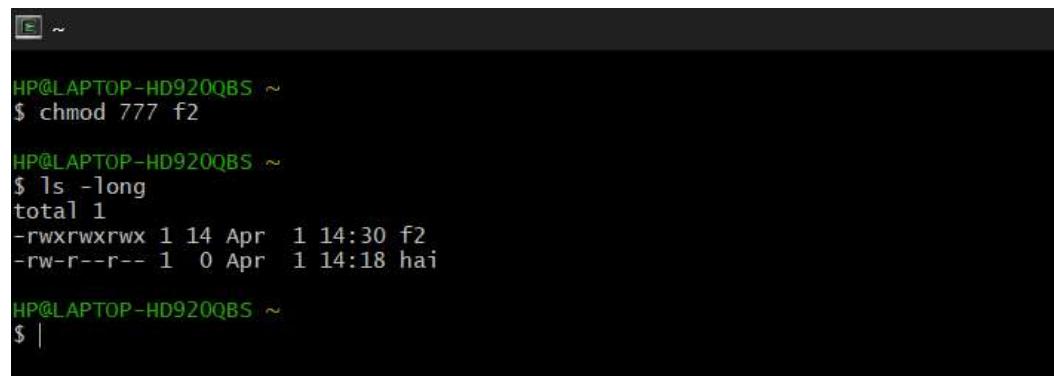
HP@LAPTOP-HD920QBS ~
$ cat f2
hai
hello
cse

HP@LAPTOP-HD920QBS ~
$
```

\$chmod: To change directory permissions in Linux, use the following:

1. chmod +rwx filename to add permissions.
2. chmod -rwx directoryname to remove permissions.
3. chmod +x filename to allow executable permissions.
4. chmod -wx filename to take out write and executable permissions.

OUTPUT:



```
HP@LAPTOP-HD920QBS ~
$ chmod 777 f2

HP@LAPTOP-HD920QBS ~
$ ls -long
total 1
-rwxrwxrwx 1 14 Apr  1 14:30 f2
-rw-r--r-- 1  0 Apr  1 14:18 hai

HP@LAPTOP-HD920QBS ~
$ |
```

\$ps(Process Status):

This command is used to display the attributes of a process.

Syntax: \$ps

Example: \$ps

OUTPUT:	PID	TTY	TIME	CMD
	644	01	10:30:00	bash
	643	02	10:31:00	ps

Options:

-f: detailed listing which shows parent of every process,use(-f)->(full) option.

Example: \$ps -f

OUTPUT:

UID	PID	PPID	C	STIME	TTY	TIME	CMD
Sumid	291	1	0	10:24:36	console	0:00	-bash

-u:it displays processes of a user.

Example: \$ps -u sumit

OUTPUT:	PID	TTY	TIME	CMD
	378	?	00:05	xsun
	403	?	00:00	xsession

-a: displaying all user processes.

Example: \$ps -a

OUTPUT:	PID	TTY	TIME	CMD
	662	pts/01	00:00:00	ksh
	705	pts/02	00:00:00	sh

OUTPUT:

```
[~] ~  
HP@LAPTOP-HD920QBS ~  
$ ps  
   PID  PPID  PGID   WINPID   TTY      UID   STIME COMMAND  
 1271  1270  1271    11412  pty0  197609 14:17:09 /usr/bin/bash  
S 1282  1271  1282    13476  pty0  197609 14:18:35 /usr/bin/cat  
S 1284  1271  1284    19648  pty0  197609 14:18:59 /usr/bin/cat  
S 1310  1271  1310    15704  pty0  197609 14:30:40 /usr/bin/cat  
 1270     1  1270    13428 ?    197609 14:17:09 /usr/bin/mintty  
S 1303  1271  1303    13672  pty0  197609 14:29:27 /usr/bin/cat  
 1321  1271  1321    18068  pty0  197609 14:34:55 /usr/bin/ps  
S 1286  1271  1286    14148  pty0  197609 14:19:45 /usr/bin/cat  
HP@LAPTOP-HD920QBS ~  
$
```

\$kill: This command is used to kill the process i.e; stop or terminate a process.(by administrator)

Syntax: \$kill <pid>

Example: \$kill 644

OUTPUT: The process gets terminated.

```
[~] ~  
HP@LAPTOP-HD920QBS ~  
$ kill  
kill: usage: kill [-s sigspec | -n signum | -sigspec] pid | jobspec ... or kill -l [sigspec]  
HP@LAPTOP-HD920QBS ~  
$ |
```

TASK 2

Task: Write a program that makes a copy of a file using standard I/O and system calls

Program:

```
#include<stdio.h>
#include<unistd.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<fcntl.h>
void main()
{
    int fd1,fd2,n=1;
    char *buf;
    fd1=open("f1",O_RDONLY);
    fd2=open("f2",O_WRONLY|O_CREAT,S_IWUSR|S_IRUSR);
    if((fd1== -1) || (fd2== -1))
        printf("error");
    else
    {
        while(n>0)
        {
            n=read(fd1,&buf,1);
            write(fd2,&buf,1);
        }
    }
}
```

OUTPUT:

```
C:\Program Files\CodeBlocks\sec.exe
Process returned 1 (0x1) execution time : 0.543 s
Press any key to continue.
```

f1
f2

03-04-2023 10:36
03-04-2023 10:36

Text Document
Text Document

1 KB
1 KB

TASK 3

Task 3(a): To Simulate the Round Robin CPU Scheduling algorithm.

```
#include<stdio.h>

int main()

{
    int i, limit, total = 0, x, counter = 0, time_quantum;

    int wait_time = 0, turnaround_time = 0, arrival_time[10], burst_time[10], temp[10];

    float average_wait_time, average_turnaround_time;

    printf("\nEnter Total Number of Processes:\t");

    scanf("%d", &limit);

    x = limit;

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

    {
        printf("\nEnter Details of Process[%d]\n", i + 1);

        printf("Arrival Time:\t");

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

        printf("Burst Time:\t");

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

        temp[i] = burst_time[i];
    }

    printf("\nEnter Time Quantum:\t");

    scanf("%d", &time_quantum);

    printf("\nProcess ID\t\tBurst Time\t Turnaround Time\t Waiting Time\n");

    for(total = 0, i = 0; x != 0;)

    {
        if(temp[i] <= time_quantum && temp[i] > 0)
```

```

total = total + temp[i];

temp[i] = 0;

counter = 1;

}

else if(temp[i] > 0)

{

temp[i] = temp[i] - time_quantum;

total = total + time_quantum;

}

if(temp[i] == 0 && counter == 1)

{

x--;

printf("\nProcess[%d]\t%d\t%d\t%d\t%d", i + 1, burst_time[i], total -
arrival_time[i], total - arrival_time[i] - burst_time[i]);

wait_time = wait_time + total - arrival_time[i] - burst_time[i];

turnaround_time = turnaround_time + total - arrival_time[i];

counter = 0;

}

if(i == limit - 1)

{

i = 0;

}

else if(arrival_time[i + 1] <= total)

{

i++;

}

else

```

```

{
    i = 0;
}

}

average_wait_time = wait_time * 1.0 / limit;

average_turnaround_time = turnaround_time * 1.0 / limit;

printf("\n\nAverage Waiting Time:\t%f", average_wait_time);

printf("\nAvg Turnaround Time:\t%f\n", average_turnaround_time);

return 0;
}

```

OUTPUT:

```

C:\Users\HP\Desktop\rundrobin.exe

Enter Total Number of Processes:      4

Enter Details of Process[1]
Arrival Time:  0
Burst Time:   21

Enter Details of Process[2]
Arrival Time:  0
Burst Time:   3

Enter Details of Process[3]
Arrival Time:  0
Burst Time:   6

Enter Details of Process[4]
Arrival Time:  0
Burst Time:   2

Enter Time Quantum:     5

Process ID      Burst Time      Turnaround Time      Waiting Time
Process[2]        3              8                  5
Process[4]        2              15                 13
Process[3]        6              21                 15
Process[1]       21             32                 11

Average Waiting Time:  11.000000
Avg Turnaround Time: 19.000000

```

b) Aim: To Simulate the Shortest Job First(SJF) CPU Scheduling algorithm.

Program:

```
#include<stdio.h>

structsa
{
    char pro[10];
    int bt,wt,tat;
}p[10],temp[10];

void main()
{
    int i,j,n,temp1=0;
    float awt=0,atat=0;
    printf("\n enter number of processes");
    scanf("%d",&n);
    printf("\n enter the name of process and burst time:");
    for(i=0;i<n;i++)
    {
        scanf("%s %d",p[i].pro,&p[i].bt);
    }
    for(i=0;i<n;i++)
    {
        for(j=i+1;j<n;j++)
        {
            if(p[i].bt>p[j].bt)
            {
                temp[i]=p[i];
                p[i]=p[j];
                p[j]=temp[i];
            }
        }
    }
}
```

```

p[i]=p[j];

p[j]=temp[i];

}

}

}

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

{

p[i].wt=temp1;

p[i].tat=p[i].bt+p[i].wt;

temp1=p[i].bt+temp1;

}

```

OUTPUT:

```

C:\Users\HP\Desktop\sjf.exe

enter number of processes4

enter the name of process and burst time:p 1 6
p 2 8
p 3 7
Process      bt      wt      tat
  6        0        0        0
  8        0        0        0
  p        1        0        1
  p        2        1        3
Average Waiting time:0.250000
Process returned 31 (0x1F)  execution time : 28.814 s
Press any key to continue.

```

c) Aim: To Simulate the First Come First Served CPU Scheduling algorithm.

Program:

```
#include<stdio.h>
main()
{
int p[10];
int tat[10],wt[10],i,n,pt[10],bt[10];
float avg=0,tot=0;
printf("enter no of processes:");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("enter process%d number:\n",i+1);
scanf("%d",&p[i]);
printf("enter process time");
scanf("%d",&pt[i]);
}
wt[0]=0;
for(i=1;i<n;i++)
{
wt[i]=pt[i-1]+wt[i-1];
tot=tot+wt[i];
}
avg=(float)tot/n;
for(i=0;i<n;i++)
tat[i]=pt[i]+wt[i];
printf("p_number\t P_time\t w_time\t turn around time\n");
for(i=0;i<n;i++)
printf("%d\t%d\t%d\t%d\n",p[i],pt[i],wt[i],tat[i]);
printf("total waiting time=%f\n avg waiting time=%f",tot,avg);
}
```

OUTPUT:

The screenshot shows a Windows Notepad window titled "lab manual os - Notepad". The window contains C code for calculating waiting times for processes. Below the code, the program's execution is shown, including user input and the resulting process table and statistics.

```
for(i=0;i<n;i++)
printf("%d\t%d\t%d\t%d\n",p[i],pt[i],wt[i],tat[i]);
printf("total waiting time=%f\n avg waiting time=%f",tot,ave
}

OUTPUT

enter no of processes:3
enter process1 number:
1
enter process time24
enter process2 number:
2
enter process time3
enter process3 number:
3
enter process time3
p_number      P_time      w_time      turn around time
1            24          0        24
2            3           24        27
3            3           27        30
total waiting time=51.000000
avg waiting time=17.000000
```

Ln 149, Col 1 | 100% | Windows (CRLF) | UTF-8

d)Aim: To Simulate the Priority CPU Scheduling algorithm.

Program:

```
#include<stdio.h>

struct sq
{
    char pro[10];
    int bt,wt,prior,tat;
};

P[10],temp;

main()
{
    int i,j,n,temp1=0;
    float awt=0,atat=0;
    printf("Enter no. of processes\n");
    scanf("%d",&n);
    printf("enter name, burst time, priority\n");
    for(i=0;i<n;i++)
    {
        scanf("%s%d%d",P[i].pro,&P[i].bt,&P[i].prior);
    }
    for(i=0;i<n;i++)
    {
        for(j=i+1;j<n;j++)
        {
            if(P[i].prior>P[j].prior)
            {
                temp=P[i];
                P[i]=P[j];
                P[j]=temp;
            }
        }
    }
    for(i=0;i<n;i++)
```

```

{
P[i].wt=temp1;
P[i].tat=P[i].wt+P[i].bt;
temp1+=P[i].bt;
}
for(i=0;i<n;i++)
{
awt+=P[i].wt;
atat+=P[i].tat;
}
printf("process\b\tbt\b\twt\b\ttat\b\n");
awt/=n;
atat/=n;
for(i=0;i<n;i++)
{
printf("%s\b\t%d\b\t%d\b\t%d\b\n",P[i].pro,P[i].bt,P[i].wt,P[i].tat);
}
printf("awt=%f\b\n,atat=%f\b\n",awt,atat);
}

```

OUTPUT:

```

C:\Users\HP\Desktop\proi.exe
Enter no. of processes
3
enter name, burst time, priority
a 10
b 5
c 90
^Z

process  bt      wt      tat
        a    10     0     10 4199705
        b     5    10     15 4199705
        c   90    15   105 4199705awt:8.333333
atat:43.333332
Process returned 0 (0x0)  execution time : 43.706 s
Press any key to continue.

```

TASK 4

Task: C Program to implement Producer-Consumer problem.

Program:

```
#include<stdio.h>

#include<stdlib.h>

int mutex=1,full=0,empty=3,x=0;

void main()

{

    int n;

    void producer();

    void consumer();

    int wait(int);

    int signal(int);

    printf("\n 1.Producer 2.Consumer 3.Exit:");

    while(1)

    {

        printf("\n Enter your choice:");

        scanf("%d",&n);

        switch(n)

        {

            case 1:if((mutex==1)&&(empty!=0))

                producer();

                else

                    printf("Buffer is full");

                    break;

            case 2:if((mutex==1)&&(full!=0))

                consumer();

        }

    }

}
```

```

        else
            printf("Buffer is empty");
            break;
        case 3:exit(0);
            break;
    }
}

}

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

int signal(int s)
{
    return (++s);
}

void producer()
{
    empty=wait(empty);
    mutex=wait(mutex);

    x++;
    printf("Producer produces item %d",x);
    mutex=signal(mutex);
    full=signal(full);
}

```

```

void consumer()
{
    full=wait(full);

    mutex=wait(mutex);

    printf("Consumer consumes item %d",x);

    x--;

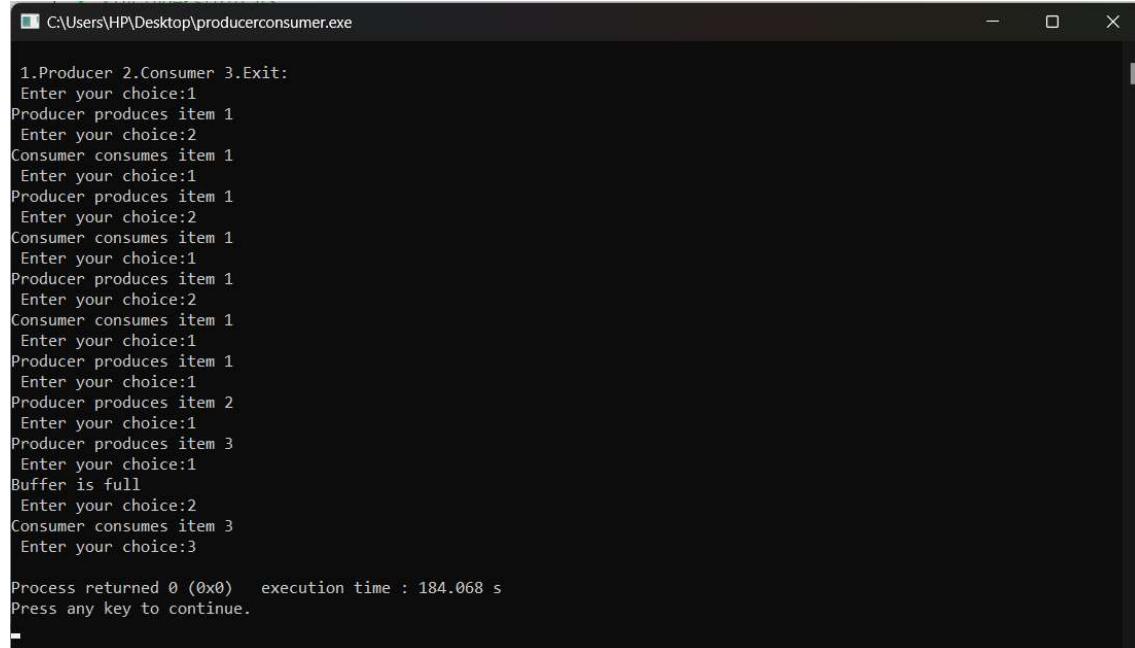
    mutex=signal(mutex);

    empty=signal(empty);

}

```

OUTPUT:



```

C:\Users\HP\Desktop\producerconsumer.exe

1.Producer 2.Consumer 3.Exit:
Enter your choice:1
Producer produces item 1
Enter your choice:2
Consumer consumes item 1
Enter your choice:1
Producer produces item 1
Enter your choice:2
Consumer consumes item 1
Enter your choice:1
Producer produces item 1
Enter your choice:2
Consumer consumes item 1
Enter your choice:1
Producer produces item 1
Enter your choice:2
Producer produces item 2
Enter your choice:1
Producer produces item 3
Enter your choice:1
Buffer is full
Enter your choice:2
Consumer consumes item 3
Enter your choice:3

Process returned 0 (0x0)  execution time : 184.068 s
Press any key to continue.

```

TASK 5

Task: Program to implement READERS-WRITERS concept.

Program:

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

sem_t mutex,wrt;
int data = 0,rcount = 0;

void *reader(void *arg)
{
    int f;
    f = ((int)arg);
    sem_wait(&mutex);
    rcount = rcount + 1;
    if(rcount==1)
        sem_wait(&wrt);
    sem_post(&mutex);
    printf("Data read by the reader%d is %d\n",f,data);
    // sleep(1);
    sem_wait(&mutex);
    rcount = rcount - 1;
    if(rcount==0)
        sem_post(&wrt);
    sem_post(&mutex);
}

void *writer(void *arg)
{
    int f;
    f = ((int) arg);
    sem_wait(&wrt);
```

```

data++;
printf("Data written by the writer%d is %d\n",f,data);
sleep(1);
sem_post(&wrt);
}

main()
{
int i,b;
pthread_t rtid[5],wtid[5];
sem_init(&mutex,0,1);
sem_init(&wrt,0,1);
for(i=0;i<=2;i++)
{
pthread_create(&wtid[i],NULL,writer,(void *)i);
pthread_create(&rtid[i],NULL,reader,(void *)i);
}
for(i=0;i<=2;i++)
{
pthread_join(wtid[i],NULL);
pthread_join(rtid[i],NULL);
}
}

```

OUTPUT:

```

C:\Users\HP\Desktop\rdwr.exe
Data written by the writer0 is 1
Data read by the reader0 is 1
Data read by the reader1 is 1
Data read by the reader2 is 1
Data written by the writer1 is 2
Data written by the writer2 is 3
Process returned 0 (0x0)   execution time : 3.501 s
Press any key to continue.

```

TASK 6:

Task: Program to implement Dining Philosopher problem using semaphores.

Program:

```
#include<stdio.h>
#include<pthread.h>
#include<stdlib.h>
#include<semaphore.h>
#define N 5
#define thinking 0
#define hungry 1
#define eating 2
#define left (ph_num+4)%N
#define right (ph_num+1)%N
sem_t mutex;
sem_t s[N];
void *philosopher(void *num);
void take_fork(int);
void put_fork(int);
void teet(int);
int state[N]={thinking,thinking,thinking,thinking,thinking};
int phil_num[N]={0,1,2,3,4};
int main()
{
    int i;
    pthread_t thread_id[N];
    sem_init(&mutex,0,1);
    for(i=0;i<N;i++)
        sem_init(&s[i],0,0);
    for(i=0;i<N;i++)
    {
        pthread_create(&thread_id[i],NULL,philosopher,&phil_num[i]);
    }
}
```

```

printf("philosopher %d is thinking \n",i+1);
}
for(i=0;i<N;i++)
pthread_join(thread_id[i],NULL);
}
void *philosopher(void *num)
{
while(1)
{
int *i=num;
sleep(1);
take_fork(*i);
sleep(1);
put_fork(*i);
}
}
void take_fork(int ph_num)
{
sem_wait(&mutex);
state[ph_num]=hungry;
printf("Philosopher %d is hungry\n",ph_num+1);
teet(ph_num);
sem_post(&mutex);
sem_wait(&s[ph_num]);
sleep(1);
}
void teet(int ph_num)
{
static count=0;
if(state[ph_num]==hungry&& state[left]!=eating && state[right]!=eating)
{
state[ph_num]=eating;
printf("Philosopher %d takes fork %d and %d\n",ph_num+1,left+1,ph_num+2);
printf("Philosopher %d is eatng\n",ph_num+1);
}
}

```

```

sem_post(&s[ph_num]);
count++;
}
if(count==5)
exit(1);
}
void put_fork(int ph_num)
{
sem_wait(&mutex);
state[ph_num]=thinking;
printf("Philosopher %d putting fork %d and %d down \n",ph_num+1,left+1,ph_num+1);
printf("Philosopher %d is thinking\n",ph_num+1);
teet(left);
teet(right);
sem_post(&mutex);
}

```

OUTPUT:

```

"C:\Program Files\CodeBlocks\dp.exe"
philosopher 3 is thinking
philosopher 4 is thinking
philosopher 5 is thinking
Philosopher 2 is hungry
Philosopher 2 takes fork 1 and 3
Philosopher 2 is eatng
Philosopher 1 is hungry
Philosopher 4 is hungry
Philosopher 4 takes fork 3 and 5
Philosopher 4 is eatng
Philosopher 3 is hungry
Philosopher 5 is hungry
Philosopher 2 putting fork 1 and 2 down
Philosopher 2 is thinking
Philosopher 1 takes fork 5 and 2
Philosopher 1 is eatng
Philosopher 4 putting fork 3 and 4 down
Philosopher 4 is thinking
Philosopher 3 takes fork 2 and 4
Philosopher 3 is eatng
Philosopher 2 is hungry
Philosopher 4 is hungry
Philosopher 1 putting fork 5 and 1 down
Philosopher 1 is thinking
Philosopher 5 takes fork 4 and 6
Philosopher 5 is eatng

Process returned 1 (0x1)  execution time : 5.774 s
Press any key to continue.

```

TASK 7

Task : Simulate Bankers Algorithm for Deadlock Avoidance

Program:

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int Max[10][10], need[10][10], alloc[10][10], avail[10], completed[10], safeSequence[10];
    /*Max denotes max required resource
    alloc denotes already allocated resources for each process
    avail denotes available resource of each kind
    completed array indicates whether each process has met with its requirements and completed
    or not.
    Safe sequence is an array which holds order of execution that can result in completion of all
    process*/
    int p, r, i, j, process, count;
    count = 0;

    printf("Enter the no of processes : ");
    scanf("%d", &p);

    for(i = 0; i < p; i++)
        completed[i] = 0; /*initially no process is completed*/

    printf("\n\nEnter the no of resources : ");
    scanf("%d", &r);
    printf("\n\nEnter the Max Matrix for each process : ");
    for(i = 0; i < p; i++)
    {
        printf("\nFor process %d : ", i + 1);
        for(j = 0; j < r; j++)
            scanf("%d", &Max[i][j]);
```

```
}
```

```
printf("\n\nEnter the allocation for each process : ");
for(i = 0; i < p; i++)
{
    printf("\nFor process %d : ",i + 1);
    for(j = 0; j < r; j++)
        scanf("%d", &alloc[i][j]);
}

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

for(i = 0; i < p; i++)
    for(j = 0; j < r; j++)
        need[i][j] = Max[i][j] - alloc[i][j]; // process still need these many resources.

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

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

process = -1; //indicates process can not completed.
```

```

for(i = 0; i < p; i++)
{
    if(completed[i] == 0)//if not completed.
    {
        process = i ; //ith process not yet completed.
        for(j = 0; j < r; j++)
        {
            if(avail[j] < need[i][j])
            {
                process = -1; //excess required which is not possible
                break;
            }
        }
    }
    /*end if*/
}

if(process != -1)
break; /* that means there exists a process that can complete its requirement*/
/*for end*/

/* process holds i th process which is not yet completed*/
if(process != -1)
{
    printf("\nProcess %d runs to completion!", process );
    safeSequence[count] = process ; /*join it to safe sequence*/
    count++; //identifying number of completed processes
    for(j = 0; j < r; j++)
    {
        avail[j] += alloc[process][j]; /*return back the resources*/
        alloc[process][j] = 0;
        Max[process][j] = 0;
        completed[process] = 1;
    }
}
}

}while(count != p && process != -1); /*for all process*/

```

```

if(count == p)
{
    printf("\nThe system is in a safe state!!\n");
    printf("Safe Sequence : < ");
    for( i = 0; i < p; i++)
        printf("%d ", safeSequence[i]);
    printf(">\n");
}
else
printf("\nThe system is in an unsafe state!!");

}

```

OUTPUT:

```

Enter the no of processes : 5

Enter the no of resources : 3

Enter the Max Matrix for each process :
For process 1 : 7 5 3

For process 2 : 3 2 2

For process 3 : 9 0 2

For process 4 : 2 2 2

For process 5 : 4 3 3

Enter the allocation for each process :
For process 1 : 0 1 0

For process 2 : 2 0 0

For process 3 : 3 0 2

For process 4 : 2 1 1

For process 5 : 0 0 2

```

```

For process 4 : 2 1 1

For process 5 : 0 0 2

Enter the Available Resources : 3 3 2

Max matrix: Allocation matrix:
7 5 3      0 1 0
3 2 2      2 0 0
9 0 2      3 0 2
2 2 2      2 1 1
4 3 3      0 0 2

Process 1 runs to completion!
Max matrix: Allocation matrix:
7 5 3      0 1 0
0 0 0      0 0 0
9 0 2      3 0 2
2 2 2      2 1 1
4 3 3      0 0 2

Process 3 runs to completion!
Max matrix: Allocation matrix:
7 5 3      0 1 0
0 0 0      0 0 0
9 0 2      3 0 2
0 0 0      0 0 0
4 3 3      0 0 2

Process 0 runs to completion!
Max matrix: Allocation matrix:
0 0 0      0 0 0
0 0 0      0 0 0
9 0 2      3 0 2
0 0 0      0 0 0
4 3 3      0 0 2

Process 2 runs to completion!
Max matrix: Allocation matrix:
0 0 0      0 0 0
0 0 0      0 0 0
0 0 0      0 0 0
0 0 0      0 0 0
4 3 3      0 0 2

Process 4 runs to completion!
The system is in a safe state!
Safe Sequence : < 1 3 0 2 4 >

```

TASK 8

Task 8a: Simulate First fit algorithm for Memory Management.

Program:

```
#include<stdio.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];
printf("Memory management Scheme-first fit");
printf("\nenter number of blocks:");
scanf("%d",&nb);
printf("\n enter the number of files:");
scanf("%d",&nf);
printf("\n enter size of blocks:");
for(i=1;i<=nb;i++)
{   printf("\nblock %d:",i);
scanf("%d",&b[i]);
}
printf("\n enter size of files:");
for(i=1;i<=nf;i++)
{
printf("\nfile %d:",i);
scanf("%d",&f[i]);
}
for(i=1;i<=nf;i++)
{
for(j=1;j<=nb;j++)
{
if(bf[j]!=1)
{
temp=b[j]-f[i];
if(temp>=0)
```

```

    {
ff[i]=j;
break;
}
}

frag[i]=temp;
bf[ff[i]]=1;
}

printf("\n file no\tfile size\tblock no\tblocksize\tfragment");
for(i=1;i<=nf;i++)
printf("\n %d \t %d \t %d \t %d \t %d",i,f[i],ff[i],b[ff[i]],frag[i]);
}

```

OUTPUT:

```

C:\Users\HP\Documents\sa.exe
Memory management Scheme-first fit
enter number of blocks:5

enter the number of files:3

enter size of blocks:
block 1:10
block 2:20
block 3:30
block 4:40
block 5:50

enter size of files:
file 1:2
file 2:5
file 3:4

      file no      file size      block no      blocksize      fragment
      1          2            1          10            8
      2          5            2          20            15
      3          4            3          30            26
Process returned 3 (0x3)  execution time : 34.679 s
Press any key to continue.

```

Task 8b:Simulate Best fit algorithm for Memory Management.

Program:

```
#include<stdio.h>
#define MAX 25
void main()
{
int frag[MAX],b[MAX],f[MAX],i,j,nb,nf,temp,lowest=10000;
static int bf[MAX],ff[MAX];
printf("\nEnter the number of blocks");
scanf("%d",&nb);
printf("\nEnter the number of files");
scanf("%d",&nf);
printf("\nEnter the size of the blocks");
for(i=1;i<=nb;i++)
{
printf("\nBlock %d",i);
scanf("%d",&b[i]);
}
printf("\nEnter the size of files");
for(i=1;i<=nf;i++)
{
printf("\nFile %d",i);
scanf("%d",&f[i]);
}
for(i=1;i<=nf;i++)
{
for(j=1;j<=nb;j++)
{
if(bf[j]!=1)
{
temp=b[j]-f[i];
if(temp>=0)
{

```

```

        ff[i]=j;
        lowest=temp;
    }
}

frag[i]=lowest;
bf[ff[i]]=1;
lowest=10000;
}

printf("\nFile No \t File Size \t Block No \t Block Size \t fragment");
for(i=1;i<=nf&&ff[i]!=0;i++)
printf("\n %d \t %d \t %d \t %d \t %d",i,f[i],ff[i],b[ff[i]],frag[i]);
}

```

OUTPUT:

```

Enter the number of files4
Enter the size of the blocks
Block 110
Block 220
Block 330
Block 440
Block 550
Enter the size of files
File 14
File 25
File 36
File 47
      File No      File Size      Block No      Block Size      fragment
      1          4            5            50           46
      2          5            4            40           35
      3          6            3            30           24
      4          7            2            20           13
Process returned 4 (0x4)   execution time : 40.701 s
Press any key to continue.

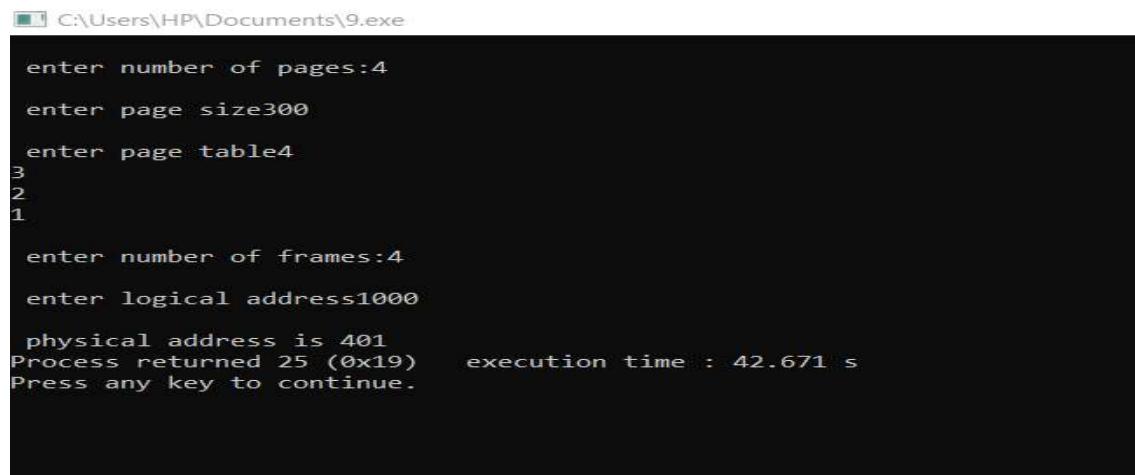
```

TASK 9

Task : To implement paging technique of memory management.

```
#include<stdio.h>
void main()
{
int np,ps,pt[20],nf,la,pn,index,pa,i,j;
printf("\n enter number of pages:");
scanf("%d",&np);
printf("\n enter page size");
scanf("%d",&ps);
printf("\n enter page table");
for(i=0;i<np;i++)
scanf("%d",&pt[i]);
printf("\n enter number of frames:");
scanf("%d",&nf);
printf("\n enter logical address");
scanf("%d",&la);
pn=la/ps;
index=la%ps;
pa=(pt[pn]+ps)+index;
printf("\n physical address is %d",pa);
}
```

OUTPUT:



```
C:\Users\HP\Documents\9.exe
enter number of pages:4
enter page size300
enter page table4
3
2
1
enter number of frames:4
enter logical address1000
physical address is 401
Process returned 25 (0x19)    execution time : 42.671 s
Press any key to continue.
```

TASK 10

Task 10a: Simulate first in first out Page Replacement Algorithm.

Program:

```
#include<stdio.h>
int main()
{
    int i,j,n,a[50],frame[10],no,k,avail,count=0;
    printf("\n ENTER THE NUMBER OF PAGES:\n");
    scanf("%d",&n);
    printf("\n ENTER THE PAGE NUMBER :\n");
    for(i=1;i<=n;i++)
        scanf("%d",&a[i]);
    printf("\n ENTER THE NUMBER OF FRAMES :");
    scanf("%d",&no);
    for(i=0;i<no;i++)
        frame[i]=-1;
    j=0;
    printf("\tref string\t page frames\n");
    for(i=1;i<=n;i++)
    {
        printf("%d\t",a[i]);
        avail=0;
        for(k=0;k<no;k++)
            if(frame[k]==a[i])
                avail=1;
        if(avail==0)
        {
            frame[j]=a[i];
            j=(j+1)%no;
            count++;
            for(k=0;k<no;k++)
                printf("%d\t",frame[k]);
        }
        printf("\n");
    }
    printf("Page Fault Is %d",count);
    return 0;
}
```

OUTPUT:

```
ENTER THE NUMBER OF PAGES:  
3  
  
ENTER THE PAGE NUMBER :  
4  
8  
2  
  
ENTER THE NUMBER OF FRAMES :3  
ref string      page frames  
4          4      -1      -1  
8          4      8       -1  
2          4      8       2  
Page Fault Is 3  
Process returned 0 (0x0)   execution time : 24.724 s  
Press any key to continue.
```

Task 10b: To simulate least recently used page replacement algorithm.

```
#include<stdio.h>

int main()

{

int frames[10], temp[10], pages[10];

int total_pages, m, n, position, k, l, total_frames;

int a = 0, b = 0, page_fault = 0;

printf("\nEnter Total Number of Frames:\t");

scanf("%d", &total_frames);

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

{

frames[m] = -1;

}

printf("Enter Total Number of Pages:\t");

scanf("%d", &total_pages);

printf("Enter Values for Reference String:\n");

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

{

printf("Value No.[%d]:\t", m + 1);

scanf("%d", &pages[m]);

}

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

{

a = 0, b = 0;

for(m = 0; m < total_frames; m++)
```

```
{  
if(frames[m] == pages[n])  
{  
a = 1;  
b = 1;  
break;  
}  
}  
  
if(a == 0)  
{  
for(m = 0; m < total_frames; m++)  
{  
if(frames[m] == -1)  
{  
frames[m] = pages[n];  
b = 1;  
break;  
}  
}  
}  
  
if(b == 0)  
{  
for(m = 0; m < total_frames; m++)  
{  
temp[m] = 0;  
}
```

```

}

for(k = n - 1, l = 1; l <= total_frames - 1; l++, k--)

{
    for(m = 0; m < total_frames; m++)

    {
        if(frames[m] == pages[k])

        {
            temp[m] = 1;
        }
    }
}

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

{
    if(temp[m] == 0)

    position = m;

    }

frames[position] = pages[n];

page_fault++;

}

printf("\n");

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

{
    printf("%d\t", frames[m]);
}

}

```

```
printf("\nTotal Number of Page Faults:\t%d\n", page_fault);

return 0;

}
```

OUTPUT :

```
Enter Total Number of Frames: 3
Enter Total Number of Pages: 5
Enter Values for Reference String:
Value No.[1]: 8
Value No.[2]: 75
Value No.[3]: 2
Value No.[4]: 36
Value No.[5]: 4

8      -1      -1
8      75      -1
8      75      2
36     75      2
36     4       2
Total Number of Page Faults: 2

Process returned 0 (0x0)  execution time : 25.979 s
Press any key to continue.
```

TASK 11

Task :Simulate the following Disc Scheduling Algorithms

Program:

```
#include<stdio.h>

int absolute(int a,int b)

{int c;

c=a-b;

if(c<0)

return -c;

else

return c;

}

int main()

{int choice,m,n,x,start,i,j,pos,min,a[15],count;

count=0;

printf("\nEnter the number of cylinders :");

scanf("%d",&m);

printf("\nEnter the number of requests :");

scanf("%d",&n);

printf("\nEnter current position :");

scanf("%d",&start);

printf("\nEnter the request queue :");

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

{scanf("%d",&a[i]);
```

```

if(a[i]>=m)
{printf("\ninvalid input");
scanf("%d",&a[i]);
}

}

do

{printf("\n\nDISK SCHEDULING ALGORITHMS\n1. FCFS\n2. SSTF\n3. SCAN\n4. C-
SCAN\n5. LOOK\n6. C-LOOK");
printf("\nEnter choice :");

scanf("%d",&choice);

count=0;

x=start;

switch(choice)

{case 1:printf("\nFCFS :\n");

printf("Scheduling services the request in the order that follows:\n%d\t",start);

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

{x-=a[i];

if(x<0)

x=-x;

count+=x;

x=a[i];

printf("%d\t",x);

}

printf("\nTotal Head Movement :%d Cylinders",count);

break;
}

```

```

case 2:printf("\nSSTF :\n");
printf("Scheduling services the request in the order that follows:\n%d\t",start);
for(i=0;i<n;i++)
{
min=absolute(a[i],x);
pos=i;
for(j=i;j<n;j++)
if(min>absolute(x,a[j]))
{
pos=j;
min=absolute(x,a[j]);
}
count+=absolute(x,a[pos]);
x=a[pos];
a[pos]=a[i];
a[i]=x;
printf("%d\t",x);
}
printf("\nTotal Head Movement: %d Cylinders",count);
break;
case 3:printf("\nSCAN :\n");
printf("Scheduling services the request in the order that follows:\n");
count=0;
pos=0;
for(i=0;i<n;i++)
for(j=0;j<n-i-1;j++)
if(a[j]>a[j+1])

```

```

{x=a[j];
a[j]=a[j+1];
a[j+1]=x;
}

for(i=0;i<n;i++)
if(a[i]<start)
pos++;
for(i=0;i<pos;i++)
for(j=0;j<pos-i-1;j++)
if(a[j]<a[j+1])
{x=a[j];
a[j]=a[j+1];
a[j+1]=x;
}

x=start;
printf("%d\t",x);
for(i=0;i<pos;i++)
{count+=absolute(a[i],x);
x=a[i];
printf("%d\t",x);
}
count+=absolute(x,0);
x=0;
printf("%d\t",x);
for(i=pos;i<n;i++)

```

```

{count+=absolute(a[i],x);

x=a[i];

printf("%d\t",x);

}

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

printf("%d\t",a[i]);*/

printf("\nTotal Head Movement: %d Cylinders",count);

break;

case 4:printf("\nC-SCAN :\n");

printf("Scheduling Services the request in the order that follows:\n%d\t",start);

count=0;

pos=0;

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

for(j=0;j<n-i-1;j++)

if(a[j]>a[j+1])

{x=a[j];

a[j]=a[j+1];

a[j+1]=x;

}

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

if(a[i]<start)

pos++;

x=start;

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

{count+=absolute(x,a[i]);

```

```

x=a[i];
printf("%d\t",x);
}
count+=absolute(m-1,x);

x=0;
printf("%d\t%d\t",m-1,0);
for(i=0;i<pos;i++)
{count+=absolute(x,a[i]);
x=a[i];
printf("%d\t",x);
}

/*for(i=0;i<n;i++)
printf("%d\t",a[i]);*/
printf("\nTotal Head movement: %d Cylinders",count);
break;

case 5:printf("\nLOOK :\n");
printf("\nScheduling services the request in the order as follows :\n%d\t",start);
count=0;
pos=0;
for(i=0;i<n;i++)
for(j=0;j<n-i-1;j++)
if(a[j]>a[j+1])
{x=a[j];
a[j]=a[j+1];
a[j+1]=x;
}
}

```

```

}

for(i=0;i<n;i++)
if(a[i]<start)
pos++;
for(i=0;i<pos;i++)
for(j=0;j<pos-i-1;j++)
if(a[j]<a[j+1])
{x=a[j];
a[j]=a[j+1];
a[j+1]=x;
}

x=start;
for(i=0;i<pos;i++)
{count+=absolute(a[i],x);
x=a[i];
printf("%d\t",x);
}

for(i=pos;i<n;i++)
{count+=absolute(a[i],x);
x=a[i];
printf("%d\t",x);
}

printf("\nTotal Head Movement: %d Cylinders",count);
break;
case 6:printf("\nC-LOOK :\n");

```

```

printf("Scheduling Services the request in the order that follows:\n%d\t",start);

count=0;

pos=0;

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

for(j=0;j<n-i-1;j++)

if(a[j]>a[j+1])

{x=a[j];

a[j]=a[j+1];

a[j+1]=x;

}

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

if(a[i]<start)

pos++;

x=start;

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

{count+=absolute(x,a[i]);

x=a[i];

printf("%d\t",x);

}

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

{count+=absolute(x,a[i]);

x=a[i];

printf("%d\t",x);

}

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

```

```

printf("%d\t",a[i]);*/
printf("\nTotal Head movement: %d Cylinders",count);
break;
}

printf("\nDo you want to continue(1 to continue) :");
scanf("%d",&choice);
}while(choice==1);
}

```

OUTPUT:

```

Enter the number of cylinders :200
Enter the number of requests :5
Enter current position :1
Enter the request queue :2
3
5
8
9

DISK SCHEDULING ALGORITHMS
1. FCFS
2. SSTF
3. SCAN
4. C-SCAN
5. LOOK
6. C-LOOK
Enter choice :1

FCFS :
Scheduling services the request in the order that follows:
1    2    3    5    8    9
Total Head Movement :8 Cylinders
Do you want to continue(1 to continue) :1

```

```

DISK SCHEDULING ALGORITHMS
1. FCFS
2. SSTF
3. SCAN
4. C-SCAN
5. LOOK
6. C-LOOK
Enter choice :

```

TASK 12

Task : To Simulate Sequential file allocation Strategy

Program:

```
#include<stdio.h>
#include<string.h>
void main()
{
    int st[20],b[20],b1[20],ch,i,j,n,blocks[20][20],sz[20];
    char F[20][20],S[20];
    printf("\n Enter no. of Files ::");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        printf("\n Enter file %d name ::",i+1);

        scanf("%s",&F[i]);
        printf("\n Enter file%d size(in kb):::",i+1);
        scanf("%d",&sz[i]);
        printf("\n Enter Starting block of %d:::",i+1);
        scanf("%d",&st[i]);
        printf("\n Enter blocksize of File%d(in bytes):::",i+1);
        scanf("%d",&b[i]);
    }
    for(i=0;i<n;i++)
        b1[i]=(sz[i]*1024)/b[i];
    for(i=0;i<n;i++)
    {
        for(j=0;j<b1[i];j++)
            blocks[i][j]=st[i]+j;
    }
    do
    {
```

```

printf("\nEnter the Filename ::");
scanf("%s",S);
for(i=0;i<n;i++)
{
    if(strcmp(S,F[i])==0)
    {
        printf("\nFname\tStart\tNblocks\tBlocks\n");
        printf("\n-----\n");
        printf("\n%s\t%d\t%d\t",F[i],st[i],b1[i]);
        for(j=0;j<b1[i];j++)
            printf("%d->",blocks[i][j]);
        }
    }
printf("\n-----\n");
printf("\nDo U want to continue ::(Y:n)");
scanf("%d",&ch);
if(ch!=1)
break;
}while(1);
}

```

OUTPUT:

C:\Users\HP\Documents\12a.exe

Enter no. of Files ::2

Enter file 1 name ::os1

Enter file1 size(in kb)::20

Enter Starting block of 1::2

Enter blocksize of File1(in bytes)::500

Enter file 2 name ::os2

Enter file2 size(in kb)::10

Enter Starting block of 2::3

Enter blocksize of File2(in bytes)::300

Enter the Filername ::os2

Fname Start Nblocks Blocks

os2 3 34 3->4->5->6->7->8->9->10->11->12->13->14->15->16->17->18->19->20->21->22->23->24->25->26->27->28->29->30->31->32->33->34->35->36->

Do U want to continue ::(Y:n)

Task 12b: To implement indexed file allocation method.

```
#include<stdio.h>
#include<string.h>
int n;
void main()
{
    int b[20],b1[20],i,j,blocks[20][20],sz[20];
    char F[20][20],S[20],ch;
    printf("\n Enter no. of Files ::");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        printf("\n Enter file %d name ::",i+1);
        scanf("%s",&F[i]);
        printf("\n Enter file%d size(in kb)::",i+1);
        scanf("%d",&sz[i]);
        printf("\n Enter blocksize of File%d(in bytes)::",i+1);
        scanf("%d",&b[i]);
    }
    for(i=0;i<n;i++)
    {
        b1[i]=(sz[i]*1024)/b[i];
        printf("\n\nEnter blocks for file%d",i+1);
        for(j=0;j<b1[i];j++)
        {
            printf("\n Enter the %dblock ::",j+1);
            scanf("%d",&blocks[i][j]);
        }
    }
    do
    {
        printf("\nEnter the Filename ::");
        scanf("%s",&S);
```

```

for(i=0;i<n;i++)
{
if(strcmp(F[i],S)==0)
{
printf("\nFname\tFsize\tBsize\tNblocks\tBlocks\n");
printf("\n-----\n");
printf("\n%s\t%d\t%d\t%d\t",F[i],sz[i],b[i],b1[i]);
for(j=0;j<b1[i];j++)
printf("%d->",blocks[i][j]);
}
printf("\n-----\n");
printf("\nDo U want to continue ::(Y:n)\n");

scanf("%d",&ch);
}while(ch!=0);
}

```

OUTPUT:

```

C:\Users\HP\Documents\12a.exe

Enter no. of Files ::2
Enter file 1 name ::os1
Enter file1 size(in kb)::1
Enter blocksize of File1(in bytes)::512
Enter file 2 name ::os2
Enter file2 size(in kb)::1
Enter blocksize of File2(in bytes)::512

Enter blocks for file1
Enter the 1block ::1000
Enter the 2block ::1001

Enter blocks for file2
Enter the 1block ::2000
Enter the 2block ::2001

Enter the Filenam e ::os1
Fname   Fsize   Bsize   Nblocks Blocks
-----
os1      1        512      2       1000->1001->
-----
Do U want to continue ::(Y:n)

```

Task 12c: To implement linked file allocation method.

```
#include<stdio.h>

#include<string.h>

int n;

void main()

{

int b[20],b1[20],i,j,blocks[20][20],sz[20];

char F[20][20],S[20],ch;

int sb[20],eb[20],x;

printf("\n Enter no. of Files ::");

scanf("%d",&n);

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

{

printf("\n Enter file %d name ::",i+1);

scanf("%s",&F[i]);

printf("\n Enter file%d size(in kb):::",i+1);

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

printf("\n Enter blocksize of File%d(in bytes):::",i+1);

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

}

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

{

b1[i]=(sz[i]*1024)/b[i];

printf("\n Enter Starting block of file%d:::",i+1);

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

```

printf("\n Enter Ending block of file%d::",i+1);

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

printf("\nEnter blocks for file%d::\n",i+1);

for(j=0;j<b1[i]-2;)

{

printf("\nEnter the %dblock ::",j+1);

scanf("%d",&x);

if(x>sb[i]&&x<eb[i])

{

blocks[i][j]=x;

j++;

}

else

printf("\n Invalid block::");

}

}

do

{

printf("\nEnter the Filename ::");

scanf("%s",&S);

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

{

if(strcmp(F[i],S)==0)

{

printf("\nFname\tFsize\tBsize\tNblocks\tBlocks\n");

```

```
printf("\n-----\n");

printf("\n%s\t%d\t%d\t%d\t",F[i],sz[i],b[i],b1[i]);

printf("%d->",sb[i]);

for(j=0;j<b1[i]-2;j++)

printf("%d->",blocks[i][j]);

printf("%d->",eb[i]);

}

}

printf("\n-----\n");

printf("\nDo U want to continue (Y:n):::");

scanf("%d",&ch);

}while(ch!=0);

}
```

OUTPUT:

C:\Users\HP\Documents\12a.exe

```
Enter no. of Files ::2
Enter file 1 name ::os1
Enter file1 size(in kb)::1
Enter blocksize of File1(in bytes)::512
Enter file 2 name ::os2
Enter file2 size(in kb)::1
Enter blocksize of File2(in bytes)::1024
Enter Starting block of file1::1100
Enter Ending block of file1::1600
Enter blocks for file1::
Enter Starting block of file2::2200
Enter Ending block of file2::2500
Enter blocks for file2::
Enter the Filename ::os1
Fname   Fsize   Bsize   Nblocks Blocks
-----
os1      1       512      2           1100->1600->
-----
Do U want to continue (Y:n)::
```