**JAWAHARLAL COLLEGE OF ENGINEERING AND TECHNOLOGY**

**(Approved by AICTE, Affiliated to APJ Abdul Kalam Technological University, Kerala)**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**(NBA Accredited)**

# *COURSE MATERIAL*

# *CSL 204 OPERATING SYSTEM LAB*

## VISION OF THE INSTITUTION

Emerge as a centre of excellence for professional education to produce high quality engineers and entrepreneurs for the development of the region and the Nation.

## MISSION OF THE INSTITUTION

- To become an ultimate destination for acquiring latest and advanced knowledge in the multidisciplinary domains.
- To provide high quality education in engineering and technology through innovative teaching-learning practices, research and consultancy, embedded with professional ethics.
- To promote intellectual curiosity and thirst for acquiring knowledge through outcome-based education.
- To have partnership with industry and reputed institutions to enhance the employability skills of the students and pedagogical pursuits.
- To leverage technologies to solve the real-life societal problems through community services.

## ABOUT THE DEPARTMENT

- Established in: 2008

- Courses offered: B.Tech in Computer Science and Engineering
- Affiliated to the A P J Abdul Kalam Technological University.

## DEPARTMENT VISION

To produce competent professionals with research and innovative skills, by providing them with the most conducive environment for quality academic and research oriented undergraduate education along with moral values committed to build a vibrant nation.

## DEPARTMENT MISSION

- Provide a learning environment to develop creativity and problem-solving skills in a professional manner.
- Expose to latest technologies and tools used in the field of computer science.
- Provide a platform to explore the industries to understand the work culture and expectation of an organization.
- Enhance Industry Institute Interaction program to develop the entrepreneurship skills.
- Develop research interest among students which will impart a better life for the society and the nation.

## PROGRAMME EDUCATIONAL OBJECTIVES

Graduates will be able to

- Provide high-quality knowledge in computer science and engineering required for a computer professional to identify and solve problems in various application domains.
- Persist with the ability in innovative ideas in computer support systems and transmit the knowledge and skills for research and advanced learning.
- Manifest the motivational capabilities, and turn on a social and economic commitment to community services.

## PROGRAM OUTCOMES (POS)

**Engineering Graduates will be able to:**

1. **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

2. **Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

3. **Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

4. **Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

5. **Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

6. **The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

7. **Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

8. **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

9. **Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

10. **Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

11. **Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

12. **Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

**COURSE OUTCOMES**

| SlNo | DESCRIPTION | Blooms'Taxonomy Level |
|---|---|---|
| C216.1 | Illustrate the use of systems calls and Implement Process Creation and Inter Process Communication in Operating Systems. | LEVEL2 |
| C216.2 | Apply First Come First Served, Shortest Job First, Round Robin and Priority based CPU Scheduling Algorithms. | LEVEL3 |
| C216.3 | Illustrate the performance of Memory allocation methods and Page Replacement Algorithms. | LEVEL2 |
| C216.4 | Apply modules for Deadlock Detection and Deadlock Avoidance in Operating Systems. | LEVEL3 |
| C216.5 | Apply modules for Storage Management and Disk Scheduling in Operating Systems | LEVEL3 |

**PROGRAM SPECIFIC OUTCOMES (PSO)**

The students will be able to

- Use fundamental knowledge of mathematics to solve problems using suitable analysis methods, data structure and algorithms.
- Interpret the basic concepts and methods of computer systems and technical specifications to provide accurate solutions.
- Apply theoretical and practical proficiency with a wide area of programming knowledge, design new ideas and innovations towards research.

**CO PO PSO MAPPING**

**Note: H-Highly correlated=3, M-Medium correlated=2,L-Less correlated=1**

| Subject Code | PO1 | PO2 | PO3 | PO 4 | PO5 | PO6 | PO7 | PO8 | PO9 | PO10 | PO11 | PO12 | PSO1 | PSO2 | PSO3 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| C216.1 | 3 | 3 | 3 | 2 | 3 | | | 2 | 3 | 3 | 2 | 3 | 3 | 3 | 2 |
| C216.2 | 3 | 3 | 3 | 2 | 3 | | | 2 | 3 | 3 | 2 | 3 | 3 | 3 | 2 |
| C216.3 | 3 | 3 | 3 | 2 | 3 | | | 2 | 3 | 3 | 2 | 3 | 3 | 3 | 2 |
| C216.4 | 3 | 3 | 3 | 2 | 3 | | | 2 | 3 | 3 | 2 | 3 | 3 | 3 | 2 |
| C216.5 | 3 | 3 | 3 | 2 | 3 | | | 2 | 3 | 3 | 2 | 3 | 3 | 3 | 2 |
| C216 | 3 | 3 | 3 | 2 | 3 | - | - | 2 | 3 | 3 | 2 | 3 | 3 | 3 | 2 |

# LAB PROGRAMS

## TABLEOFCONTENTS


| EXP.NO | NAMEOFTHEEXPERIMENT | PAGE.NO |
|---|---|---|
| 1 | CPUSCHEDULINGALGORITHMS | |
| | A)FIRSTCOMEFIRSTSERVE(FCFS) | 1-3 |
| | B)SHORTESTJOBFIRST(SJF) | 4-6 |
| | C)ROUND ROBIN | 7-9 |
| | D)PRIORITY | 10-12 |
| 2 | PRODUCER-CONSUMER PROBLEM USINGSEMAPHORES | 13-14 |
| 3 | DINING-PHILOSOPHERSPROBLEM | 15-18 |
| 4 | MEMORYMANAGEMENTTECHNIQUES | |
| | A)MULTI PROGRAMMING WITH FIXED NUMBEROFTASKS(MFT) | 19-21 |
| | B)MULTI PROGRAMMING WITH VARIABLENUMBEROFTASKS(MVT) | 22-24 |
| 5 | CONTIGUOUSMEMORYALLOCATION | |
| | A)WORSTFIT | 25-26 |
| | B)BESTFIT | 27-28 |
| | C)FIRSTFIT | 28-29 |
| | PAGEREPLACEMENTALGORITHMS | |

| | | |
|---|---|---|
| 6 | A)FIRSTINFIRSTOUT(FIFO) | 30-32 |
| | B)LEASTRECENTLYUSED(LRU) | 33-35 |
| | C)OPTIMAL | 36-39 |
| 7 | FILEORGANIZATIONTECHNIQUES | |
| | A)SINGLELEVELDIRECTORY | 40-42 |
| | B)TWO LEVELDIRECTORY | 43-46 |
| 8 | FILEALLOCATIONSTRATEGIES | |
| | A)SEQUENTIAL | 47-49 |
| | B)INDEXED | 50-52 |
| | C)LINKED | 53-55 |
| 9 | DEADLOCKAVOIDANCE | 56-59 |
| 10 | DEADLOCKPREVENTION | 60-62 |
| 11 | DISKSCHEDULINGALGORITHMS | |
| | A)FCFS | 63-64 |
| | B)SCAN | 65-66 |
| | C)C-SCAN | 67-69 |

**EXPERIMENTNO.1**

## CPUSCHEDULINGALGORITHMS

### A). FIRSTCOMEFIRSTSERVE:

**AIM:**TowriteacprogramtosimulatetheCPUschedulingalgorithmFirstComeFirstServe (FCFS)

**DESCRIPTION:**

To calculate the average waiting time using the FCFS algorithm first the waitingtime of the first process is kept zero and the waiting time of the second process is theburst time of the first process and the waiting time of the third process is the sum of theburst times of the first and the second process and so on. After calculating all the waitingtimes the average waiting time is calculated as the average of all the waiting times. FCFSmainlysaysfirstcomefirstservethealgorithmwhichcamefirstwillbeservedfirst.

**ALGORITHM:**

Step1: Start theprocess
Step2: Accept thenumberofprocessesinthe readyQueue
Step 3: For each process in the ready Q, assign the process name and the burst time
Step4: Set the waiting of the first process as _0'and its burst time as its turnaround time
Step5:foreachprocessintheReadyQcalculate
a). Waitingtime(n)=waitingtime(n-1)+Bursttime(n-1)b).Turnaroundtime (n)= waitingtime(n)+Bursttime(n)
Step6: Calculate

a) Averagewaiting time=Totalwaiting Time/ Numberofprocess

b) AverageTurnaround

time=TotalTurnaroundTime/NumberofprocessStep7:Stoptheprocess

**SOURCECODE:**

```
#include<stdio.h>
#include<conio.h>
main()
{
intbt[20],wt[20],tat[20],i,n;floa
 twtavg,tatavg;
clrscr();
printf("\nEnter the number of processes --
                    ");scanf("%d",&n);
for(i=0;i<n;i++)
```

```
{
printf("\nEnterBurstTimeforProcess%d--
",i);scanf("%d",&bt[i]);
}
wt[0]=wtavg=0;tat[0] =
tatavg                =
bt[0];for(i=1;i<n;i++)
{
wt[i]=wt[i-1]+bt[i-1];
tat[i]=tat[i-
1]+bt[i];wtavg=wtavg+w
t[i];tatavg=tatavg+ tat[i];
}
printf("\tPROCESS \tBURSTTIME\tWAITINGTIME\tTURNAROUNDTIME\n");
for(i=0;i<n;i++)
        printf("\n\tP%d\t\t%d\t\t%d\t\t%d",i,bt[i],wt[i],tat[i]);printf("\nAverage
        Waiting Time--%f",wtavg/n);
printf("\nAverageTurnaroundTime--
%f",tatavg/n);getch();
}
```

***INPUT***

| | |
|---|---|
| Enterthenumberofprocesses-- | 3 |
| EnterBurstTimeforProcess0-- | 24 |
| EnterBurstTimeforProcess1 -- | 3 |
| EnterBurstTimeforProcess2-- | 3 |

***OUTPUT***

| PROCESS | BURSTTIME | WAITINGTIME | TURNAROUND TIME |
|---|---|---|---|
| P0 | 24 | 0 | 24 |
| P1 | 3 | 24 | 27 |
| P2 | 3 | 27 | 30 |

AverageWaitingTime-- 17.000000

AverageTurnaroundTime-- 27.000000

**B). SHORTESTJOBFIRST:**

**AIM:**TowriteaprogramtostimulatetheCPUschedulingalgorithmShortestjobfirst(Non-Preemption)

**DESCRIPTION:**

To calculate the average waiting time in the shortest job first algorithm the sorting ofthe process based on their burst time in ascending order then calculate the waiting time ofeach process as the sum of the bursting times of all the process previous or before to thatprocess.

**ALGORITHM:**

Step1: Start theprocess
Step2: Acceptthenumberofprocessesinthe readyQueue
Step3:ForeachprocessintheadyQ,assigntheprocessidandaccepttheCPUbursttime
Step4:StarttheReadyQaccordingtheshortestBursttimebysortingaccordingtolowesttohighestburs ttime.
Step5:Setthewaitingtimeofthefirstprocessas_0‘anditsturnaroundtimeasitsbursttime.
Step6:Sort theprocessesnamesbasedontheirBurttimeStep7:Foreachpr ocessinthereadyqueue,calculate
a) Waitingtime(n)=waitingtime(n-1)+Burst time(n-1)
b) Turnaroundtime(n)=waitingtime(n)+Bursttime(n)
Step8: Calculate
c) Averagewaiting time=Totalwaiting Time/ Number ofprocess
d) AverageTurnaround time=TotalTurnaroundTime/Number ofprocessStep9:Stoptheprocess

**SOURCE CODE :**

```
#include<stdio.h>
#include<conio.h>
main()
{
intp[20],bt[20],wt[20],tat[20],i,k,n,temp;floatwtavg,tatavg;
clrscr();
printf("\nEnter the number of processes --
");scanf("%d",&n);
for(i=0;i<n;i++)
{
p[i]=i;
printf("Enter Burst Time for Process %d -- ",
i);scanf("%d",&bt[i]);

}
```

```
for(i=0;i<n;i++)for(
k=i+1;k<n;k++)if(b
t[i]>bt[k])
{
temp=bt[i];
bt[i]=bt[k];
bt[k]=temp;

temp=p[i];
p[i]=p[k];p
[k]=temp;
}
wt[0]=wtavg=0;
tat[0] =tatavg =bt[0];for(i=1;i<n;i++)
{
wt[i]=wt[i-1]+bt[i-1];
 tat[i]=tat[i-
1]+bt[i];wtavg  =  wtavg
+  wt[i];tatavg  =tatavg+
tat[i];
}
printf("\n\tPROCESS\tBURSTTIME\tWAITINGTIME\tTURNAROUNDTIME\n");
for(i=0;i<n;i++)
        printf("\n\t P%d \t\t %d \t\t %d \t\t %d", p[i], bt[i], wt[i],
        tat[i]);printf("\nAverageWaitingTime--%f",wtavg/n);
printf("\nAverageTurnaroundTime --%f", tatavg/n);getch();}
```

***INPUT***

| | |
|---|---|
| Enterthenumberofprocesses-- | 4 |
| EnterBurstTimeforProcess0-- | 6 |
| EnterBurstTimeforProcess1-- | 8 |
| EnterBurstTimeforProcess2-- | 7 |
| EnterBurstTimeforProcess3-- | 3 |

***OUTPUT***

| PROCESS | BURST TIME | WAITING TIME | TURNAROUNDTIME |
|---|---|---|---|
| P3 | 3 | 0 | 3 |
| P0 | 6 | 3 | 9 |
| P2 | 7 | 9 | 16 |
| P1 | 8 | 16 | 24 |
| AverageWaitingTime -- | | 7.000000 | |
| AverageTurnaroundTime-- | | 13.000000 | |

## C). ROUNDROBIN:

**AIM:**TosimulatetheCPUschedulingalgorithmround-robin.

**DESCRIPTION:**

To aim is to calculate the average waiting time. There will be atimeslice, eachprocessshould be executed within that time-slice and if not it will go tothe waitingstate so first check whether the burst time is less than the time-slice.If it is less than itassign the waiting time to the sum of the total times. If it is greater than the burst-timethen subtract the time slot from the actual burst time and increment it by time-slot andthe loopcontinues untilalltheprocesses arecompleted.

**ALGORITHM:**

Step1: Start theprocess

Step2:AcceptthenumberofprocessesinthereadyQueueandtimequantum(or)timeslice

Step3:ForeachprocessinthereadyQ,assigntheprocessidandacceptheCPUbursttime

Step4:Calculatetheno.oftimeslicesforeachprocesswhereNo.oftimesliceforproces s(n)=bursttime process(n)/timeslice

Step5:Iftheburstimeislessthanthetime slice thenthe no.oftimeslices=1.

Step6:Consider thereadyqueueisacircular Q, calculate

a) Waitingtimeforprocess(n)=waitingtimeofprocess(n-1)+bursttimeofprocess(n-1)+thetimedifferenceingettingtheCPUfromprocess(n-1)

b) Turnaroundtimeforprocess(n)=waitingtimeofprocess(n)+bursttimeofprocess(n)+ thetime differenceingettingCPUfromprocess(n).

Step7: Calculate

c) Averagewaiting time=Totalwaiting Time/ Number ofprocess

d) Average Turnaround time = Total Turnaround Time / Number ofprocess

Step8:Stoptheprocess

**SOURCECODE**

```
#include<stdio.h>
main()
{
int
        i,j,n,bu[10],wa[10],tat[10],t,ct[10],max;
floatawt=0,att=0,temp=0;
clrscr();
printf("Enterthenoofprocesses--
");scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("\nEnterBurstTimeforprocess%d--
",i+1);scanf("%d",&bu[i]);
ct[i]=bu[i];
}
printf("\nEnterthesizeoftimeslice--
");scanf("%d",&t);
max=bu[0];for(i=1;i<n;i
++)if(max<bu[i])max=b
u[i];for(j=0;j<(max/t)+1;
j++)for(i=0;i<n;i++)if(b
u[i]!=0)
 if(bu[i]<=t)
                    {ta
t[i]=temp+bu[i];te
mp=temp+bu[i];b
u[i]=0;
 }
 else{bu[i]=b
u[i]-
t;temp=temp+
t;
 }
for(i=0;i<n;i++){
wa[i]=tat[i]-
ct[i];att+=tat[i];awt
+=wa[i];}
printf("\nTheAverageTurnaroundtimeis--
%f",att/n);printf("\nTheAverageWaitingtimeis--%f",awt/n);
printf("\n\tPROCESS\tBURSTTIME\tWAITINGTIME\tTURNAROUNDTIME\n");
for(i=0;i<n;i++)
printf("\t%d\t%d\t\t%d\t\t%d\n",i+1,ct[i],wa[i],tat[i]);getch();}
```

**INPUT:**

Enterthe noofprocesses–3
Enter Burst Time for process 1 –
24EnterBurstTimeforprocess2--
3EnterBurstTimeforprocess3–
3Enterthesize oftimeslice–3

**OUTPUT:**

| PROCESS | BURSTTIME | WAITINGTIME | TURNAROUNDTIME |
|---|---|---|---|
| 1 | 24 | 6 | 30 |
| 2 | 3 | 4 | 7 |
| 3 | 3 | 7 | 10 |

TheAverageTurnaroundtimeis–
15.666667TheAverageWaitingtimeis5.666667

## D). PRIORITY:

**AIM:**TowriteacprogramtosimulatetheCPUschedulingpriorityalgorithm.

**DESCRIPTION:**

To calculate the average waiting time in the priority algorithm, sort the bursttimes according to their priorities and then calculate the average waiting time of theprocesses.The waiting time of each process isobtained by summing up the burst timesofalltheprevious processes.

**ALGORITHM:**

Step1: Start theprocess
Step2: Accept thenumberofprocessesinthe readyQueue
Step3:ForeachprocessinthereadyQ,assigntheprocessidandacceptheCPUbursttime
Step4: Sort thereadyqueueaccording to theprioritynumber.
Step5:Setthewaitingofthefirstprocessas_0‘anditsbursttimeas itsturnaroundtimeStep6:Arrange theprocessesbasedonprocesspriority
Step7:ForeachprocessintheReadyQcalculateStep8:
foreachprocessinthe ReadyQcalculate

a) Waitingtime(n)=waitingtime(n-1)+Bursttime(n-1)
b) Turnaroundtime(n)=waitingtime(n)+Bursttime(n)

Step9: Calculate

c) Averagewaiting time=Totalwaiting Time/ Number ofprocess
d) AverageTurnaroundtime=TotalTurnaroundTime/NumberofprocessPrinttheresultsinanorder.

Step10:Stop

**SOURCECODE:**

```
#include<stdio.h>
main()
{
intp[20],bt[20],pri[20],wt[20],tat[20],i,k,n,temp;floatwtavg,tatavg;
clrscr();
printf("Enterthenumberofprocesses---
");scanf("%d",&n);
for(i=0;i<n;i++){
p[i]= i;
printf("EntertheBurst Time&PriorityofProcess%d---",i);scanf("%d
%d",&bt[i],&pri[i]);
}
for(i=0;i<n;i++)for(
k=i+1;k<n;k++)if(p
ri[i]
>pri[k]){temp=p[i];
p[i]=p[k];p[k]=tem
p;temp=bt[i];bt[i]=
bt[k];bt[k]=temp;te
mp=pri[i];pri[i]=pri
[k];pri[k]=temp;
}
wtavg=wt[0]=0;tatavg
=          tat[0]          =
bt[0];for(i=1;i<n;i++)
{
wt[i]=wt[i-1] +bt[i-1];
tat[i] =tat[i-1]+bt[i];

wtavg=wtavg+wt[i];tata
 vg =tatavg+ tat[i];
}
printf("\nPROCESS\t\tPRIORITY\tBURSTTIME\tWAITINGTIME\tTURNAROUNDTIME"
);
for(i=0;i<n;i++)
printf("\n%d      \t\t      %d      \t\t      %d      \t\t      %d      \t\t      %d
",p[i],pri[i],bt[i],wt[i],tat[i]);printf("\nAverage  Waiting  Time  is  ---
%f",wtavg/n); printf("\nAverageTurnaroundTimeis---%f",tatavg/n);
getch();}
```

***INPUT***

Enterthe numberofprocesses--5

Enterthe BurstTime &PriorityofProcess 0---10 3

Enterthe BurstTime &PriorityofProcess 1---1 1

Enterthe BurstTime &PriorityofProcess 2---2 4

Enterthe BurstTime &PriorityofProcess 3---1 5

Enterthe BurstTime &PriorityofProcess 4---5 2

***OUTPUT***

| PROCESS | PRIORITY | BURSTTIME | WAITIN G TIME0 | TURNAROU ND TIME1 |
|---|---|---|---|---|
| 1 | 1 | 1 | | |
| 4 | 2 | 5 | 1 | 6 |
| 0 | 3 | 10 | 6 | 16 |
| 2 | 4 | 2 | 16 | 18 |
| 3 | 5 | 1 | 18 | 19 |

AverageWaitingTimeis--- 8.200000

AverageTurnaroundTimeis---------- 12.000000

**VIVAQUESTIONS**

1) Definethefollowing
   a) Turnaroundtime b)Waitingtime c) Bursttime d)Arrivaltime
2) Whatismeantbyprocessscheduling?
3) Whatarethe variousstatesofprocess?
4) Whatisthedifferencebetweenpreemptiveandnon-preemptivescheduling
5) Whatis meantbytimeslice?
6) Whatisroundrobinscheduling?

**EXPERIMENT.NO2**

**AIM**:To WriteaCprogramtosimulateproducer-consumer problemusingsemaphores.

**DESCRIPTION**

Producer consumer problem is a synchronization problem. There is a fixed size buffer where theproducer produces items and that is consumed by a consumer process. One solution to the producer-consumerproblemusessharedmemory.Toallowproducerandconsumerprocessestorunconcurrently, there must be available a buffer of items that can be filled by the producer and emptiedby the consumer. This buffer will reside in a region of memory that is shared by the producer andconsumer processes. The producer and consumer must be synchronized, so that the consumer doesnottrytoconsumeanitemthathasnotyetbeenproduced.

**PROGRAM**

```
#include<stdio.>
voidmain()
{
        intbuffer[10],bufsize,in,out,produce,consume,choice=
        0;in=0;
        out=0;
        bufsize=10;
        while(choice!=3)
        {
                printf("\n1.  Produce  \t  2.  Consume  \t3.
                        Exit");printf("\nEnteryourchoice:");
                scanf("%d",&choice);
                switch(choice){
                        case1:if((in+1)%bufsize==out)
                                        printf("\nBufferisFull");
                                else
                                {
                                        printf("\nEnterthevalue:");s
                                        canf("%d",&produce);buffe
                                        r[in]= produce;
                                        in=(in+1)%bufsize;
                                }
                                break;;;
                        case2:  if(in==out)
                                        printf("\nBufferisEmpty");
                                else
                                {
                                consume=buffer[out];
                                printf("\nTheconsumedvalue
                                is%d",consume);out=(out+1)%bufsize;
                                }
                                break;
}       }               }
```

***OUTPUT***

```
1.Produce      2.Consume      3.
               ExitEnteryourchoice:2
BufferisEmpty
1.Produce      2.Consume      3.
               ExitEnteryourchoice:1
Enterthevalue:100
1.Produce      2.Consume      3.
               ExitEnteryourchoice:2
Theconsumedvalueis100
1.Produce      2.Consume      3.
               ExitEnteryourchoice:3
```

**EXPERIMENT.NO3**

**AIM:**ToWriteaCprogramtosimulatetheconceptofDining-Philosophersproblem.

**DESCRIPTION**

The dining-philosophers problem is considered a classic synchronization problem becauseitis an exampleof a large class of concurrency-control problems. It is a simple representation of the need to allocate severalresources among several processes in a deadlock-free and starvation-free manner. Consider five philosopherswho spend their lives thinking and eating. The philosophers share a circular table surrounded by five chairs,each belonging to one philosopher. In the center of the table is a bowl of rice, and the table is laid with fivesingle chopsticks. When a philosopher thinks, she does not interact with her colleagues. From time to time, aphilosopher gets hungry and tries to pick up the two chopsticks that are closest to her (the chopsticks that arebetween her and her left and right neighbors). A philosopher may pick up only one chopstick at a time.Obviously, she cam1ot pick up a chopstick thatis already in the hand of a neighbor. When a hungryphilosopher has both her chopsticks at the same time, she eats without releasing her chopsticks. When she isfinished eating, she puts down both of her chopsticks and starts thinking again. The dining-philosophersproblem may lead to a deadlock situation and hence some rules have to be framed to avoid the occurrence ofdeadlock.

**PROGRAM**

```
int tph,philname[20],status[20],howhung,hu[20],cho;main()
{
        inti;clrscr();
        printf("\n\nDININGPHILOSOPHERPROBLEM");
        printf("\nEnterthetotalno.ofphilosophers:");sca
        nf("%d",&tph);
        for(i=0;i<tph;i++)
        {
                philname[i]=(i+1);status[i]=1;
        }
        printf("Howmanyarehungry:
         ");scanf("%d",&howhung);if(ho
         whung==tph)
        {
    printf("\n All are hungry..\nDead lock stage will
    occur");printf(\n"Exiting\n");
    else{for(i=0;i<howhung;
    i++){
    printf("Enterphilosopher%dposition:",(i+1));scanf("%d",&hu[i
    ]);
    status[hu[i]]=2;
    }
```

```
    do
            {
                printf("1.Onecaneatatatime\t2.Twocaneatatatime
                \t3.Exit\nEnter your
                choice:");scanf("%d",&cho);s
                witch(cho)
                {
                case1:    one();
                            break;
            case2:    two();
                    break;c
            ase3:exit(0);
                    default:printf("\nInvalidoption..");
                }
            }while(1);
        }
}
one()
{
        intpos=0,x,i;
        printf("\nAllow one philosopher to eat at any
        time\n");for(i=0;i<howhung;i++,pos++)
        {
    printf("\nP %d is granted to eat",
    philname[hu[pos]]);for(x=pos;x<howhung;x++)
    printf("\nP%diswaiting",philname[hu[x]]);

        }
}
two()
{

        inti,j,s=0,t,r,x;
        printf("\nAllowtwophilosopherstoeatatsametime\n");for(i=0
        ;i<howhung;i++)
        {
            for(j=i+1;j<howhung;j++)
            {
                if(abs(hu[i]-hu[j])>=1&&abs(hu[i]-hu[j])!=4)
                {
                    printf("\n\ncombination%d\n",(s+1));t=
                    hu[i];
                    r=hu[j];s++;
                    printf("\nP%dandP%daregrantedtoeat",philname[hu[i]],philn
                        ame[hu[j]]);
```

```
                    for(x=0;x<howhung;x++)
                    {
                        if((hu[x]!=t)&&(hu[x]!=r))
                        printf("\nP%diswaiting",philname[hu[x]]);
                    }
                }
            }
        }
    }
```

***INPUT***

```
DININGPHILOSOPHERPROBLEM
Enterthetotalno.ofphilosophers:5How
manyarehungry:3
Enterphilosopher1position:2
Enterphilosopher2position:4
Enterphilosopher3position:5
```

***OUTPUT***

```
1.Onecaneatatatime      2.Two
caneatatatime           3.ExitEnter your choice:1

AllowonephilosophertoeatatanytimeP3is
 grantedtoeat
P       3       is
waitingP  5  is
waitingP0isw
aiting
P5isgrantedtoeatP
5iswaiting
P0iswaiting
P0isgrantedtoeatP
0iswaiting
```

1.Onecaneatatatime
2.Twocaneatatatime3.ExitEnte
ryourchoice: 2

Allow
twophilosopherstoeatatsametimecombinatio
n1
P3andP5aregrantedtoeatP0is
waiting

combination2
P3andP0aregrantedtoeatP5is
waiting

combination3
P5andP0aregrantedtoeatP3is
waiting

1.Onecaneatatatime 2.Twocaneatat atime
3.ExitEnter your choice:3

**EXPERIMENT.NO4MEMO**
**RYMANAGEMENT**

## A).MEMORYMANAGEMENTWITHFIXEDPARTITIONINGTECHNIQUE(MFT)

**AIM:**To implement andsimulatetheMFTalgorithm.

**DESCRIPTION:**

In this the memory is divided in two parts and process is fit into it. The process which is bestsuited will be placed in the particular memory where it suits. In MFT, the memory is partitionedinto fixed size partitions and each job is assigned to a partition. The memory assigned to apartition does not change. In MVT, each job gets just the amount of memory it needs. That is, thepartitioning of memory is dynamic and changes as jobs enter and leave the system. MVT is amore ``efficient'' user of resources. MFT suffers with the problem of internal fragmentation andMVTsuffers withexternalfragmentation.

**ALGORITHM:**

Step1: Start the
process.Step2:Declar
evariables.
Step3:Entertotalmemorysizems.Step
4:Allocate memoryforos.
Ms=ms-os
Step5:ReadthenopartitiontobedividednPartitionsize=ms/n.Step6:Rea
dtheprocessnoandprocess size.
Step 7: If process sizeisless than partitionsize allotalseblocketheprocess.
Whileallocatingupdatememorywastage-externalfragmentation.

```
if(pn[i]==pn[j])f=1;
if(f==0){if(ps[i]<=siz)
{
extft=extft+size-
ps[i];avail[i]=1;count++;
}
}
```

Step8: Printtheresults

**SOURCECODE:**

```
#include<stdio.h>
#include<conio.h>
main()
{
int     ms,     bs,     nob,
        ef,n,mp[10],tif=0; inti,p=0;
clrscr();
printf("Enter the total memory available (in Bytes) --
");scanf("%d",&ms);
printf("Enter the block size (in Bytes) --
");scanf("%d",&bs);
nob=ms/bs;ef=m
s-nob*bs;
printf("\nEnterthenumberofprocesses--
");scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("Entermemoryrequiredforprocess%d(inBytes)--
",i+1);scanf("%d",&mp[i]);
}
printf("\nNo.          of          Blocks          available          in          memory--
%d",nob);printf("\n\nPROCESS\tMEMORYREQUIRED\tALLOCATED\tINTERNALFRAGM
ENTATION");
for(i=0;i<n&&p<nob;i++)
{
printf("\n%d\t\t%d",i+1,mp[i]);i
f(mp[i]>bs)
printf("\t\tNO\t\t---
");else
{
printf("\t\tYES\t%d",bs-
 mp[i]);tif=tif+bs-mp[i];
p++;
}
}
if(i<n)
printf("\nMemory is Full, Remaining Processes cannot be
accomodated");printf("\n\nTotalInternalFragmentationis%d",tif);
 printf("\nTotalExternalFragmentationis%d",ef);ge
tch();
}
```

***INPUT***

Enterthetotalmemoryavailable(inBytes)-- 1000

Enterthe block size (inBytes)-- 300

Enterthenumberofprocesses–5

Enter memoryrequiredfor process1(inBytes)-- 275

Enter memoryrequiredfor process2(inBytes)-- 400

Enter memoryrequiredfor process3(inBytes)-- 290

Enter memoryrequiredfor process4(inBytes)-- 293

Enter memoryrequiredfor process5(inBytes)-- 100

No.ofBlocks availableinmemory-- 3

***OUTPUT***

| PROCESS | MEMORYREQUIRED | ALLOCATED | INTERNAL FRAGMENTATION |
|---|---|---|---|
| 1 | 275 | YES | 25 |
| 2 | 400 | NO | ----- |
| 3 | 290 | YES | 10 |
| 4 | 293 | YES | 7 |

Memory is Full, Remaining Processes cannot be accommodated

TotalInternalFragmentationis 42

TotalExternalFragmentationis100

**B)MEMORYVARIABLEPARTIONING TYPE(MVT)**

**AIM:**TowriteaprogramtosimulatetheMVTalgorithm

**ALGORITHM:**

Step1: start the process.Step2:Declare variables.
Step3:Entertotalmemorysizems.Step 4:Allocate memoryforos.
Ms=ms-os
Step5:ReadthenopartitiontobedividednPartitionsize=ms/n.Step6:Readtheprocessnoandprocess size.
Step7:Ifprocesssizeislessthanpartitionsizeallotalseblocketheprocess.Whileallocatingupdate memorywastage-externalfragmentation.

```
if(pn[i]==pn[j])
                    f=1;
if(f==0){if(ps[i]<=size)
{
extft=extft+size-
ps[i];avail[i]=1;count++;
}
}
```

Step8:PrinttheresultsStep 9: Stoptheprocess.

**SOURCECODE:**

```
#include<stdio.h>#in
clude<conio.h>main(
)
 {
 int
                    ms,mp[10],i,
 temp,n=0; char ch =
 'y';clrscr();
 printf("\nEnterthetotalmemoryavailable(inBytes)--
 ");scanf("%d",&ms);
 temp=ms;for(i=0;ch=='y';i++,n+
 +)
 {
 printf("\nEntermemoryrequiredforprocess%d(inBytes)--
 ",i+1);scanf("%d",&mp[i]);
 if(mp[i]<=temp)
 {
 printf("\nMemory
 isallocatedforProcess%d",i+1);temp=temp-mp[i];
 }
 else
 {
 printf("\nMemoryisFull");break;
 }
 printf("\nDo you want to continue(y/n) --
  ");scanf("%c",&ch);
 }
 printf("\n\nTotalMemoryAvailable--
 %d",ms);printf("\n\n\tPROCESS\t\tMEMORYALLOCATED")
 ;for(i=0;i<n;i++)
 printf("\n\t%d\t\t%d",i+1,mp[i]);
 printf("\n\nTotal      Memory      Allocated      is      %d",ms-
 temp);printf("\nTotalExternalFragmentationis %d",temp);
 getch();
 }
```

**OUTPUT:**

Enterthetotalmemoryavailable(inBytes)–1000
Entermemoryrequiredforprocess1(inBytes)–400Memoryis
allocatedforProcess 1
Doyouwanttocontinue(y/n)-- y
Entermemoryrequiredforprocess2(inBytes)--275Memoryis
allocatedforProcess 2
Doyouwanttocontinue(y/n)–y
Entermemoryrequiredforprocess3(inBytes)–550

MemoryisFull

TotalMemoryAvailable–1000

| PROCESS | MEMORYALLOCATED |
|---|---|
| 1 | 400 |
| 2 | 275 |

Total Memory Allocated is
675TotalExternalFragmentationis32
5

VIVAQUESTIONS

1) WhatisMFT?
2) WhatisMVT?
3) WhatisthedifferencebetweenMVTandMFT?
4) Whatismeantbyfragmentation?
5) Givethedifferencebetweeninternalandexternalfragmentation

**EXPERIMENT.NO5**

**MEMORYALLOCATIONTECHNIQUES**

**AIM:**ToWriteaCprogramtosimulatethefollowingcontiguousmemoryallocationtechniques

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

**DESCRIPTION**

Oneofthesimplestmethodsformemoryallocationistodividememoryintoseveralfixed-sizedpartitions. Each partition may contain exactly one process. In this multiple-partition method, when apartition is free, a process is selected from the input queue and is loaded into the free partition. When theprocess terminates, the partition becomes availablefor another process. The operatingsystem keeps atable indicating which parts of memory are available and which are occupied. Finally, when a processarrives and needs memory, a memory section large enough for this process is provided. When it is time toload or swap a processinto main memory, andif there is more than one free block of memory ofsufficient size, then the operating system must decide which free block to allocate. Best-fit strategychooses the block that is closestin size to the request. First-fitchooses the firstavailableblock thatislarge enough.Worst-fitchooses thelargestavailableblock.

**PROGRAM**

***WORST-FIT***

```
#include<stdio.h>
#include<conio.h>
#define max
25voidmain()
{
        intfrag[max],b[max],f[max],i,j,nb,nf
        ,temp;staticintbf[max],ff[max];clrsc
        r();
        printf("\n\tMemory   Management   Scheme   -   First
                                Fit");printf("\nEnterthenumberofblocks:"
        );
        scanf("%d",&nb);
        printf("Enterthenumberoffiles:");scanf("
        %d",&nf);
        printf("\nEnter   the   size   of   the   blocks:-
        \n");for(i=1;i<=nb;i++)
        {
                printf("Block%d:",i);
                scanf("%d",&b[i]);
        }
        printf("Enter   the   size   of   the   files   :-
        \n");for(i=1;i<=nf;i++)
        {
                printf("File%d:",i);
                scanf("%d",&f[i]);
```

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

}
```

***INPUT***

Enterthenumberofblocks:3En
terthenumberoffiles: 2

Enterthe sizeofthe blocks:-
Block1:5
Block2: 2
Block3: 7

Enterthesizeofthefiles:-File
1:1
File2:4

***OUTPUT***

| FileNo | FileSize | BlockNo | BlockSize | Fragment |
|---|---|---|---|---|
| 1 | 1 | 1 | 5 | 4 |
| 2 | 4 | 3 | 7 | 3 |

***BEST-FIT***

```
 #include<stdio.h>
#include<conio.h>
#define max
25voidmain()
{
        intfrag[max],b[max],f[max],i,j,nb,nf,temp,lowest=10000;sta
        ticintbf[max],ff[max];
        clrscr();
       printf("\nEnterthenumberofblocks:");scanf("%d",&nb)
        ;
        printf("Enterthenumberoffiles:");scanf("
        %d",&nf);
        printf("\nEnter the size of the blocks:-
        \n");for(i=1;i<=nb;i++)
     printf("Block%d:",i);
     scanf("%d",&b[i]);
       printf("Enter   the   size   of   the   files   :-
       \n");for(i=1;i<=nf;i++)
       {
               printf("File%d:",i);
               scanf("%d",&f[i]);
       }
        for(i=1;i<=nf;i++)
       {
                for(j=1;j<=nb;j++)
               {
                       if(bf[j]!=1)
                       {
                               temp=b[j]-
                               f[i];if(temp>=
                               0)
                                       if(lowest>temp)
                                       {
                                       ff[i]=j;lowest=
                                        temp;
                                        }
                     }}
                frag[i]=lowest;bf[ff[i]]=1;lowest=10000;
       }
        printf("\nFile    No\tFile    Size    \tBlock
                        No\tBlockSize\tFragment");for(i=1;i
        <=nf&&ff[i]!=0;i++)

               printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);ge
        tch();
}
```

***INPUT***

Enterthenumberofblocks:3
Enterthenumberoffiles: 2

Enterthesizeoftheblocks:-
Block1:5
Block2: 2
Block3: 7

Enterthesizeofthefiles:-File
1:1
File2:4

***OUTPUT***

| FileNo | FileSize | Block No | BlockSize | Fragment |
|---|---|---|---|---|
| 1 | 1 | 2 | 2 | 1 |
| 2 | 4 | 1 | 5 | 1 |

***FIRST-FIT***

```
#include<stdio.h>
#include<conio.h>
#define max
25voidmain()
{
        intfrag[max],b[max],f[max],i,j,nb,nf,temp,highe
        st=0;staticintbf[max],ff[max];
        clrscr();
        printf("\n\tMemoryManagementScheme-
        WorstFit");printf("\nEnterthenumberofblocks:");
         scanf("%d",&nb);
        printf("Enterthenumberoffiles:");scanf("
        %d",&nf);
        printf("\nEnter the size of the blocks:-
         \n");for(i=1;i<=nb;i++)
        {
                printf("Block%d:",i);
                scanf("%d",&b[i]);
        }
        printf("Enter the size of the files :-
        \n");for(i=1;i<=nf;i++)
        {
                printf("File%d:",i);
                scanf("%d",&f[i]);
        }
```

```
        for(i=1;i<=nf;i++)
        {
                for(j=1;j<=nb;j++)
                {
                        if(bf[j]!=1)//ifbf[j]isnotallocated
                        {
                                temp=b[j]-
                                f[i];if(temp>=
                                0)
                                        if(highest<temp)
                                        {
                        }
                }
                 frag[i]=highest;bf[ff[i]]=1;highest=0;
        }
        ff[i]=j;highest=temp;
    }
        printf("\nFile_no:\tFile_size:\tBlock_no:\tBlock_size:\tFragement");
        for(i=1;i<=nf;i++)
                printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);ge
        tch();
}
```

***INPUT***

Enterthenumberofblocks:3E
nterthe numberoffiles: 2

Enterthesizeoftheblocks:-
Block1:5
Block2: 2
Block3: 7

Enterthesizeofthefiles:-
File 1:1
File2:4

***OUTPUT***

| FileNo | FileSize | Block No | BlockSize | Fragment |
|---|---|---|---|---|
| 1 | 1 | 3 | 7 | 6 |
| 2 | 4 | 1 | 5 | 1 |

**EXPERIMENTNO.6**

**PAGEREPLACEMENTALGORITHMS**

**AIM:**Toimplement FIFOpagereplacement technique.
**a)FIFO b)LRU c) OPTIMAL**

**DESCRIPTION:**

Page replacement algorithms are an important part of virtual memory management and it helps the OS todecide which memory page can be moved out making space for the currently needed page. However, theultimateobjective ofall page replacementalgorithmsistoreducethenumberofpagefaults.
FIFO-This is the simplest page replacement algorithm. In this algorithm, the operating system keeps trackof all pages in the memory in a queue, the oldest page is in the front of the queue. When a page needs to bereplacedpageinthefrontofthequeueisselectedforremoval.

LRU-Inthisalgorithmpagewill bereplacedwhichis leastrecentlyused

OPTIMAL- In this algorithm, pages are replaced which would not be used for the longest duration of timein the future. This algorithm will give us less page faultwhen compared to other page replacementalgorithms.

**ALGORITHM:**

1. Startthe process
2. Read numberofpagesn
3. Readnumberofpagesno
4. Readpagenumbersintoanarraya[i]
5. Initializeavail[i]=0.tocheckpagehit
6. Replacethepagewithcircularqueue,whilere-placingcheckpageavailabilityintheframePlace avail[i]=1ifpageis placedintheframe Countpage faults
7. Print theresults.
8. Stoptheprocess.

**A) FIRSTINFIRSTOUT SOURCECODE :**

```
#include<stdio.h>#include<
conio.h> int
fr[3];voidmain()
{
voiddisplay();
inti,j,page[12]={2,3,2,1,5,2,4,5,3,2,5,2};
intflag1=0,flag2=0,pf=0,frsize=3,top
=0;clrscr();
for(i=0;i<3;i++)
{
fr[i]=-1;
}
for(j=0;j<12;j++)
{
flag1=0;flag2=0;for(i=0;i<12;i++)
{
if(fr[i]==page[j])
{
flag1=1;flag2=1;break;
}
}
if(flag1==0)
{
for(i=0;i<frsize;i++)
{
if(fr[i]==-1)
{
fr[i]=page[j];flag2=1;break;
}
}
}
if(flag2==0)
{
fr[top]=page[j];
top++;
pf++;if(top>=f
rsize)top=0;
}
display();
}
```

```
printf("Numberofpagefaults:%d",pf+frsize);get
 ch();
}
voiddisplay()
{
int     i;
        printf("\n");for(i
=0;i<3;i++)printf("%d
\t",fr[i]);
}
```

**OUTPUT:**

```
2-1-1
23-1
23-1
231
531
521
524
524
324
324
354
352

Numberofpagefaults:9
```

**B) LEASTRECENTLYUSED**

**AIM:**ToimplementLRUpagereplacementtechnique.

**ALGORITHM:**

1. Starttheprocess
2. Declarethesize
3. Get thenumber ofpagesto beinserted
4. Getthevalue
5. Declarecounterandstack
6. Selecttheleastrecentlyusedpagebycountervalue
7. Stackthemaccordingtheselection.
8. Displaythevalues
9. Stoptheprocess

**SOURCECODE:**

```
#include<stdio.h>
#include<conio.h>
intfr[3];
voidmain()
{
voiddisplay();
intp[12]={2,3,2,1,5,2,4,5,3,2,5,2},i,j,fs[3];
intindex,k,l,flag1=0,flag2=0,pf=0,frsize=3;
 clrscr();
for(i=0;i<3;i++)
{
fr[i]=-1;
}
for(j=0;j<12;j++)
{
flag1=0,flag2=0;
for(i=0;i<3;i++)
{
if(fr[i]==p[j])
{
flag1=1;flag2=
1;break;
}
}
if(flag1==0)
```

```
{
for(i=0;i<3;i++)
{
if(fr[i]==-1)
{
fr[i]=p[j];
                    flag2=1;
break;
}
}
}
if(flag2==0)
{
for(i=0;i<3;i++)
fs[i]=0;
for(k=j-1,l=1;l<=frsize-1;l++,k--)
{
for(i=0;i<3;i++)
{
if(fr[i]==p[k])fs[i]=1;
}}
for(i=0;i<3;i++)
{
if(fs[i]==0)
index=i;
}
fr[index]=p[j];
pf++;
}
display();
}
printf("\nnoofpagefaults:%d",pf+frsize);get
ch();
}
voiddisplay()
{
inti;printf("\n");for(i=0;i<3;i++)
printf("\t%d",fr[i]);
}
```

**OUTPUT:**

2-1-1
23-1
23-1
231
251
251
254
254
354
352
352
352

Noofpage faults:7

**C) OPTIMAL**

**AIM:**Toimplementoptimalpagereplacementtechnique.

**ALGORTHIM:**

**1.** Start Program
**2.** ReadNumberOfPagesAndFrames3.R
eadEachPageValue
**4.** SearchForPageInTheFrames
**5.**IfNotAvailableAllocateFreeFrame
**6.** IfNoFramesIsFreeRepalceThePageWithThePageThatIsLeastlyUsed7.PrintPageN
umberOfPageFaults
8.Stopprocess.

**SOURCECODE:**

```
/* Program to simulate optimal page replacement
   */#include<stdio.h>
#include<conio.h>
intfr[3],n,m;void
display();v
oidmain()
{
 inti,j,page[20],fs[10];i
 nt
 max,found=0,lg[3],index,k,l,flag1=0,flag2=0,pf=0;
 floatpr;
 clrscr();
 printf("Enter length of the reference string:
                 ");scanf("%d",&n);
 printf("Enter the reference string:
                 ");for(i=0;i<n;i++)
 scanf("%d",&page[i]);
 printf("Enter no of frames:
                 ");scanf("%d",&m);
 for(i=0;i<m;i++)
 fr[i]=-1;pf=m;
```

```
for(j=0;j<n;j++)
 {
 flag1=0;
                        flag2=0;
 for(i=0;i<m;i++)
 {
 if(fr[i]==page[j])
 {
 flag1=1;flag2=1;
 break;
 }
 }
 if(flag1==0)
 {
 for(i=0;i<m;i++)
 {
 if(fr[i]==-1)
 {
fr[i]=page[j];flag2=1;brea
k;
 }
 }
 }
 if(flag2==0)
 {
 for(i=0;i<m;i++)
 lg[i]=0;for(i=0;i
 <m;i++)
 {
 for(k=j+1;k<=n;k++)
 {
 if(fr[i]==page[k])
 {
 lg[i]=k-
 j;break;
 }
 }
 }
 found=0;for(i=0;
 i<m;i++)
 {
 if(lg[i]==0)
 {
index=i;f
ound=1;
```

```
 break;
 }
 }
if(found==0)
 {
 max=lg[0];
                    index=0;
 for(i=0;i<m;i++)
 {
 if(max<lg[i])
{
 max=lg[i];
 index=i;
}
 }
 }
 fr[index]=page[j];
 pf++;
}
display();
}
printf("Number of page faults : %d\n",
                    pf);pr=(float)pf/n*100;
printf("Pagefaultrate=%f\n",pr);getch();
}
voiddisplay()
{
 int   i;
       for(i=0;i<m;i++)pri
 ntf("%d\t",fr[i]);printf("\
 n");
}
```

**OUTPUT:**

Enter lengthofthereferencestring:12
Enterthereferencestring:123412512345Enternooff
rames:3
1-1-1
12-1
123
124
124
124
125
125
125
325
425
425
Number ofpagefaults:7Pagefaultrate=58.333332

**VIVAQUESTIONS**

1) Whatismeantbypagefault?
2) Whatismeantbypaging?
3) Whatispagehitandpagefaultrate?
4) Listthevariouspagereplacementalgorithm
5) Whichone isthebestreplacementalgorithm?

**EXPERIMENTNO. 7**

**FILEORGANIZATIONTECHNIQUES**

## A) SINGLELEVELDIRECTORY:

**AIM:**ProgramtosimulateSingleleveldirectoryfileorganizationtechnique.

**DESCRIPTION:**

The directory structure is the organization of files into a hierarchy of folders. In a single-leveldirectory system, all the files are placed in one directory. There is a root directory which has allfiles. It has a simple architecture and there are no sub directories. Advantage of single leveldirectorysystemis thatitis easytofindafileinthe directory.

**SOURCECODE :**

```
#include<stdio.h>
struct
{
char
        dname[10],fname[10][10];
intfcnt;
}dir;

voidmain()
{
int    i,ch;
       charf[30];
clrscr();dir.fcnt
= 0;
printf("\nEnter   name   of   directory   --
                    ");scanf("%s",dir.dname);
while(1)
{
printf("\n\n1.CreateFile\t2.DeleteFile\t3.SearchFile \n
4. Display Files\t5. Exit\nEnter your choice --
");scanf("%d",&ch);
switch(ch)
{
case1:printf("\nEnterthenameofthefile--
");scanf("%s",dir.fname[dir.fcnt]);
dir.fcnt++;break;
case 2: printf("\nEnter the name of the file --
");scanf("%s",f);
for(i=0;i<dir.fcnt;i++)
{
if(strcmp(f,dir.fname[i])==0)
{
printf("File%s isdeleted",f);strcpy(dir.fname[i],dir.fname[dir.fcnt-1]);break;
```

}

```
        }
        if(i==dir.fcnt)
        printf("File%snotfound",f);
                                        else
                                            dir.fcnt--
                                            ;break;
        case3:                              printf("\nEnterthe nameofthefile--");
                                            scanf("%s",f);for(i=0;
                                            i<dir.fcnt;i++)
                                            {
                                            if(strcmp(f,dir.fname[i])==0)
                                            {
                                            printf("File%sisfound",f);bre
                                            ak;
                                            }
                                            }
                                            if(i==dir.fcnt)
                                            printf("File%snotfound",f);b
                                            reak;
        case4:                              if(dir.fcnt==0)
                                            printf("\nDirectoryEmpty");
                                            else
                                            {
                                            printf("\nThe Files are --
                                            ");for(i=0;i<dir.fcnt;i++)pri
                                            ntf("\t%s",dir.fname[i]);
                                            }
                                            break;
        default:exit(0);
        }
}
getch();}
```

**OUTPUT:**

Entername ofdirectory--CSE
1. CreateFile2. DeleteFile3. SearchFile
4. DisplayFiles5.ExitEnteryourchoice–1

Enterthe name ofthe file--A
1. CreateFile2. DeleteFile3. SearchFile
4. DisplayFiles5.ExitEnteryourchoice–1

Enterthe name ofthe file--B
1. CreateFile2. DeleteFile3. SearchFile
4. DisplayFiles5.ExitEnteryour choice–1

Enterthe name ofthe file--C
1. CreateFile2. DeleteFile3. SearchFile
4. DisplayFiles5.ExitEnteryourchoice–4

TheFilesare--ABC
1. CreateFile2. DeleteFile3. SearchFile
4. DisplayFiles5.ExitEnteryour choice–3

Enterthenameofthefile–
ABCFileABCnotfound
1. CreateFile2. DeleteFile3. SearchFile
4. DisplayFiles5. ExitEnteryourchoice–2

Enterthenameofthefile–BFile
Bisdeleted
1. CreateFile2.DeleteFile3.SearchFile
4. DisplayFiles5.ExitEnteryourchoice–5

## B) TWOLEVELDIRECTORY

**AIM:**Programtosimulatetwolevelfileorganizationtechnique

**Description:**
In the two-level directory system, each user has own user file directory (UFD). The systemmaintainsamasterblock thathasoneentry foreach user.Thismasterblockcontainstheaddresses of the directory of the users. When a userjob starts or a user logs in, the system'smaster file directory (MFD) is searched. When a user refers to a particular file, only his own UFDissearched.

**SOURCECODE :**

```
#include<stdio.h>
struct
{
        char
                dname[10],fname[10][10];
        intfcnt;
}dir[10];

voidmain()
{
        int    i,ch,dcnt,k;
               charf[30], d[30];
        clrscr();dcnt=0;
        while(1)
        {
                printf("\n\n1.CreateDirectory\t2.CreateFile\t3.DeleteFile");printf("\n4
                .SearchFile\t\t5.Display\t6.Exit\tEnteryour choice--
                ");scanf("%d",&ch);
                switch(ch)
                {
                        case 1: printf("\nEnter name of directory --
                                ");scanf("%s",
                                                dir[dcnt].dname);
                                dir[dcnt].fcnt=0;
                                dcnt++;
                                printf("Directorycreated");break;
                        case2:printf("\nEnternameofthedirectory--
                                ");scanf("%s",d);
                                for(i=0;i<dcnt;i++)
                                        if(strcmp(d,dir[i].dname)==0)
                                        {
                          printf("Entername      of      the      file          --
                                                ");scanf("%s",dir[i].fname[dir[i].fcnt])
                          ;
```

```
                    dir[i].fcnt++;printf("F
                     ilecreated");
                        }
                if(i==dcnt)
                    printf("Directory%snotfound",d);b
                    reak;
        case3:printf("\nEnternameofthedirectory--
                ");scanf("%s",d);
                for(i=0;i<dcnt;i++)
                for(i=0;i<dcnt;i++)
                {
                if(strcmp(d,dir[i].dname)==0)
                {
                        printf("Enternameofthefile--
                        ");scanf("%s",f);for(k=0;k<dir[i].fcnt;
                        k++)
                        {
                        if(strcmp(f,dir[i].fname[k])==0)
                        {
                        printf("File%s
                        isdeleted",f);dir[i].fcnt--;
                        strcpy(dir[i].fname[k],dir[i].fname[dir[i].fcnt]);got
                        ojmp;
                        }
                }

                printf("File%snotfound",f);gotojmp;
        }
        }
        printf("Directory%snotfound",d);j
        mp:break;
case4:printf("\nEnternameofthedirectory--
        ");scanf("%s",d);
        for(i=0;i<dcnt;i++)
        {
                if(strcmp(d,dir[i].dname)==0)
                {
                        printf("Enterthenameofthefile--
                        ");scanf("%s",f);for(k=0;k<dir[i].fcnt;
                        k++)
                        {
                                if(strcmp(f,dir[i].fname[k])==0)
                                {
                                printf("File%s is found",f);gotojmp1;
                                }
                        }
        printf("File%snotfound",f);gotojmp1;
}
}
```

```
                    printf("Directory%snotfound",d);jmp1:break;case
                5:if(dcnt==0)
              printf("\nNoDirectory's");
                    else
                    {
                          printf("\nDirectory\tFiles");
                          for(i=0;i<dcnt;i++)
                          {
                             printf("\n%s\t\t",dir[i].dname);for(k=0;k<dir[i].fcnt;k++
                             )printf("\t%s",dir[i].fname[k]);

                          }
                    }
                    break;

            default:exit(0);
        }
    }
getch();
}
```

**OUTPUT**

1.CreateDirectory2.CreateFile3.DeleteFile
4.SearchFile5.Display6.ExitEnt
eryourchoice--1
Enternameofdirectory--DIR1Directorycreated

1.CreateDirectory2.CreateFile3.DeleteFile
4. Search File 5. Display 6. Exit Enter your choice --
1Enternameof directory--DIR2Directorycreated
1.CreateDirectory2.CreateFile3.DeleteFile
4. Search File 5. Display 6. Exit Enter your choice --
2Enternameofthedirectory–DIR1
Enter name of the file --
A1Filecreated
1.CreateDirectory2.CreateFile3.DeleteFile
4.SearchFile5.Display6.ExitEnt
eryourchoice--2
Enternameofthedirectory–DIR1

Enternameofthefile--
A2Filecreated
1.CreateDirectory2.CreateFile3.DeleteFile
4.SearchFile5.Display6.Ex
itEnteryourchoice–6

**VIVAQUESTIONS**

1. Definedirectory?
2. Listthedifferent typesofdirectorystructures?
3. Whatistheadvantageofhierarchicaldirectorystructure?
4. Whichofthe directorystructuresis efficient?Why?
5. Whatisacyclicgraphdirectory?

**EXPERIMENT.NO.8**

**FILEALLOCATIONSTRATEGIES**

**A) SEQUENTIAL:**

**AIM:**TowriteaCprogramforimplementingsequentialfileallocation method

**DESCRIPTION:**

The most common form of file structure is the sequential file in this type of file,a fixed format is used for records. All records (of the system) have the same length,consisting of the same number of fixed length fields in a particular order becausethelength and position of each field are known, only the values of fields need to be stored, thefieldnameandlengthforeachfieldareattributes ofthefilestructure.

**ALGORITHM:**

Step1:Starttheprogram.
Step2:Getthenumberoffiles.
Step3: Getthememoryrequirement ofeach file.
Step 4: Allocate the required locations to each in sequential order
a).Randomlyselectalocation fromavailablelocation s1=random(100);
a) Checkwhethertherequiredlocationsarefreefromtheselectedlocation.

```
 if(b[s1].flag==0){
for
                    (j=s1;j<s1+p[i];j++){
if((b[j].flag)==0)count++;
      }
if(count==p[i])break;
}
```

b) Allocateandsetflag=1totheallocatedlocations.

```
for(s=s1;s<(s1+p[i]);s++)
{
k[i][j]=s;j=j+1;b[s].bno=s;b[
s].flag=1;
}
```

Step5:Printtheresultsfileno,length,Blocksallocated.Step6:Stopth
eprogram

**SOURCECODE :**

```
#include<stdio.h>
main()
{
intf[50],i,st,j,len,c,k;clr
scr();for(i=0;i<50;i++)
f[i]=0;
X:
printf("\nEnterthestartingblock&lengthoffile");scanf("%d%d",&st,
&len);
for(j=st;j<(st+len);j++)
if(f[j]==0)
{
f[j]=1
;
printf("\n%d->%d",j,f[j]);
}
else
{
printf("Blockalreadyallocated");bre
ak;
}
if(j==(st+len))
printf("\nthefileisallocatedtodisk");
printf("\nifuwanttoentermorefiles?(y-1/n-0)");scanf("%d",&c);
if(c==1)
gotoX;el
seexit();
getch();
}
```

**OUTPUT:**

Enterthestartingblock&lengthoffile4104->1
5->1
6->1
7->1
8->1
9->1
10->1
11->1
12->1
13->1
Thefileisallocatedtodisk.

**B) INDEXED:**

**AIM:**Toimplementallocationmethodusingchained method

**DESCRIPTION:**

In the chained method file allocation table contains a field which points to startingblock of memory. From it for each bloc a pointer is kept to next successive block. Hence,there is noexternalfragmentation.

**ALGORITHM:**

Step1:Starttheprogram.
Step2: Getthenumber offiles.
Step3: Getthememoryrequirement ofeach file.
Step4:Allocatetherequiredlocationsbyselectingalocationrandomlyq=random(100);

a) Checkwhethertheselectedlocationis free.
b) Ifthelocationisfreeallocateandsetflag=1totheallocatedlocations.

```
q=random(100);
{
if(b[q].flag==0)
b[q].flag=1;
b[q].fno=j;
r[i][j]=q;
```

Step5:Printtheresultsfileno,length,Blocksallocated
.
Step6:Stoptheprogram

**SOURCECODE :**

```
#include<stdio.h>
int
        f[50],i,k,j,inde[50],n,c,count=0,p;
main()
{
clrscr();for(i=0;i
<50;i++)f[i]=0;
x:printf("enterindexblock\t");sca
nf("%d",&p);
if(f[p]==0)
{f[p]=
1;
printf("enternooffilesonindex\t");scanf
("%d",&n);
}
else
{
printf("Blockalreadyallocated\n");got
ox;
}
for(i=0;i<n;i++)scanf("%d
",&inde[i]);for(i=0;i<n;i+
+)if(f[inde[i]]==1)
{
printf("Blockalreadyallocated");goto
x;
}
for(j=0;j<n;j++)
f[inde[j]]=1;
printf("\nallocated");prin
tf("\n                    file
indexed");for(k=0;k<n;k
++)
printf("\n%d->%d:%d",p,inde[k],f[inde[k]]);
printf("Enter1toentermorefilesand0toexit\t");scanf("%
d",&c);
if(c==1)
goto
          x;
elseexit(
);
getch();
}
```

**OUTPUT:**enterindexblock9Enter
no of files on index 3 123
AllocatedFil
eindexed9-
>1:1
9->2;1
9->3:1enter 1toenter morefilesand0toexit

**C) LINKED:**

**AIM:**To implementlinked fileallocationtechnique.

**DESCRIPTION:**

In the chained method file allocation table contains a field which points to startingblock of memory. From it for each bloc a pointer is kept to next successive block. Hence,there is noexternalfragmentation

**ALGORTHIM:**

Step1:Starttheprogram.Step2
:Getthenumberoffiles.
Step3: Getthememoryrequirement ofeach file.
Step4:Allocatetherequiredlocationsbyselectingalocation
randomlyq=random(100);

a) Checkwhethertheselectedlocationis free.
b) Ifthelocationisfreeallocateandsetflag=1totheallocatedlocations.
Whileallocatingnextlocationaddresstoattachittopreviouslocation

```
for(i=0;i<n;i++)
{
for(j=0;j<s[i];j++)
{
q=random(100);
                    if(b[q].flag==0)
b[q].flag=1;
b[q].fno=j;
r[i][j]=q;
        if(j>0)
        {
}
}
p=r[i][j-1];b[p].next=q;}
```

Step5:Printtheresultsfileno,length,Blocksallocated
.
Step6:Stoptheprogram

**SOURCECODE :**

```
#include<stdio.h>
main()
{
intf[50],p,i,j,k,a,st,len,n,c;
clrscr();
for(i=0;i<50;i++)f[i]=0;
printf("Enterhowmanyblocksthatarealreadyalloca
ted");scanf("%d",&p);
printf("\nEntertheblocksno.sthatarealreadyallocated");for(
i=0;i<p;i++)
{
scanf("%d",&a);
f[a]=1;
}
X:
printf("Enter the starting index block
&length");
scanf("%d%d",&st,&len);
k=len;for(j=st;j<(k+st);j++)
{
if(f[j]==0)
{f[j]=1;
printf("\n%d->%d",j,f[j]);
}
else
{
printf("\n%d-
>fileisalreadyallocated",j);
k++;
}
}
printf("\nIfuwanttoenteronemor
efile?(yes-1/no-0)");
scanf("%d",&c);
if(c==1)
goto
X;
elseex
it();
getch( );}
```

**OUTPUT:**

Enterhowmanyblocksthatarealreadyallocated3Entertheblocksno.sthatarealread
yallocated47Enterthestartingindexblock& length379
3->1
4->1fileisalreadyallocated5-
>1
6->1
7->1fileisalreadyallocated8-
>1
9->1file is already
allocated10->1
11->1
12->1

**VIVAQUESTIONS**

1) Listthevarioustypesoffiles
2) Whatarethevariousfileallocationstrategies?
3) Whatislinkedallocation?
4) Whataretheadvantagesoflinkedallocation?
5) Whatarethedisadvantagesofsequentialallocationmethods?

**EXPERIMENT.NO9**

**DEADLOCKAVOIDANCE**

**AIM:To**SimulatebankersalgorithmforDeadLockAvoidance(Banker‘sAlgorithm)

**DESCRIPTION:**

Deadlockis a situation where in two or more competing actions are waitingf or the otherto finish, and thus neither ever does. When a new process enters a system, it must declare themaximum number of instances of each resource type it needed. This number may exceed thetotal number of resources in the system. When the user request a set of resources, the systemmust determine whether the allocation of each resources will leave the system in safe state. Ifit will the resources are allocation; otherwise the process must wait until some other processrelease theresources.

Datastructures

- n-Number ofprocess,m-number ofresourcetypes.
- Available:Available[j]=k,k– instance of resource type Rj is available.Max:
- Ifmax[i,j]=k,Pimayrequestatmost kinstancesresource Rj.
- Allocation:If Allocation [i,j]=k,Pi allocated tok instances of resource Rj Need:IfNeed[I, j]=k, Pi may need k more instances of resource type Rj, Need[I, j]=Max[I, j]-Allocation[I,j];

***SafetyAlgorithm***

1. Work and Finish be the vector of length m and n respectively, Work=Available andFinish[i]=False.
2. FindanisuchthatbothFinish[i] =False
   Need<=WorkIfnosuchIexistsgo to step4.
3. work=work+Allocation,Finish[i]=True;
4. if Finish[1]=True for all I, then the system is in safe state.Resource requestalgorithm

Let Request i be request vector for the process Pi, If request i=[j]=k, then process Piwantskinstances ofresourcetype Rj.

1. ifRequest<=Need Igoto step2.Otherwiseraiseanerrorcondition.
2. ifRequest<=Availablegotostep3.OtherwisePimustsincetheresourcesareavailable.
3. HavethesystempretendtohaveallocatedtherequestedresourcestoprocessPibymodifyingthestateasfollows;
   Available=Available-RequestI;Allocation I=Allocation +Request I;Needi=Needi-RequestI;

If the resulting resource allocation state is safe, the transaction is completed and process Pi isallocated its resources. However if the state is unsafe, the Pi must wait for Request i and theoldresource-allocationstateisrestored.

**ALGORITHM:**

1. Starttheprogram.
2. Getthevaluesofresourcesandprocesses.
3. Gettheavailvalue.
4. Afterallocationfindtheneedvalue.
5. Checkwhetheritspossibletoallocate.
6. Ifitispossiblethenthesystem isinsafestate.
7. Elsesystemisnotinsafetystate.
8. Ifthenewrequestcomesthencheckthatthesystemisinsafety.
9. ornotifweallowtherequest.
10. stoptheprogram.
11. *end*

**SOURCECODE :**

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
voidmain()
{
intalloc[10][10],max[10][10];in
tavail[10],work[10],total[10];int
i,j,k,n,need[10][10];
intm;
int     count=0,c=0;
char
            finish[10];
clrscr();
printf("Enter    the    no.    of    processes    and
                 resources:");scanf("%d%d",&n,&m);
for(i=0;i<=n;i++)
finish[i]='n';
printf("Enter      the      claim
                  matrix:\n");for(i=0;i<n;i++)
for(j=0;j<m;j++)scanf("%d",&m
ax[i][j]);
printf("Enter      the      allocation
                  matrix:\n");for(i=0;i<n;i++)
for(j=0;j<m;j++)scanf("%
d",&alloc[i][j]);printf("Re
source
vector:");for(i=0;i<m;i++)
scanf("%d",&total[i]);for(i
=0;i<m;i++)
avail[i]=0;          for(i=0;i<n;i++)
```

```
for(j=0;j<m;j++)avail[j]+=allo
c[i][j];for(i=0;i<m;i++)work[i]
=avail[i];for(j=0;j<m;j++)wor
k[j]=total[j]-
work[j];for(i=0;i<n;i++)for(j=
0;j<m;j++)need[i][j]=max[i][j]
-alloc[i][j];A:
for(i=0;i<n;i++)
{
 c=0;
for(j=0;j<m;j++)if((need[i][j]<=work[j])&&(f
inish[i]=='n'))c++;
if(c==m)
{
printf("AlltheresourcescanbeallocatedtoProcess%d",i+1);printf
("\n\nAvailable resources are:");
for(k=0;k<m;k++)
{
work[k]+=alloc[i][k];
printf("%4d",work[k]);
}
printf("\n");
finish[i]='y';
printf("\nProcess%dexecuted?:%c\n",i+1,finish[i]);coun
t++;
}
}
if(count!=n)
gotoA;
 else
printf("\nSystemisinsafemode");printf("
\n The given state is safe state");getch();
}
```

**OUTPUT**

Enter the no. of processes and resources: 4
3Entertheclaimmatrix:
322
613
314
422
Entertheallocationmatrix:
100
612
211
002
Resource vector:936
Allthe resourcescanbe
allocatedtoProcess2Available resources
are:623
Process2 executed?:y
AlltheresourcescanbeallocatedtoProcess3Availableresourcesare:834
Process3 executed?:y
AlltheresourcescanbeallocatedtoProcess4Availableresourcesare:836
Process4 executed?:y
Allthe resourcescanbe
allocatedtoProcess1Available resources
are:936
Process1executed?:ySyste
misinsafemodeThegivensta
teissafestate

**VIVAQUESTIONS**

1) Whatismeantbydeadlock?
2) Whatissafestateinbanker'salgorithms?
3) Whatisbanker'salgorithm?
4) Whatarethenecessaryconditionswheredeadlock occurs?
5) Whataretheprinciplesandgoalsofprotection?

**EXPERIMENT.NO 10DEADLOCKPREVENTION**

**AIM:**Toimplementdeadlockpreventiontechnique

**Banker'sAlgorithm:**

When a new process enters a system, it must declare the maximum number ofinstances of each resource type it needed. This number may exceed the total number ofresources in the system. When the user request a set of resources, the system mustdeterminewhethertheallocationofeachresourceswill leavethesystemin safestate.Ifit will the resources are allocation; otherwise the process must wait until some otherprocessreleasetheresources.

**DESCRIPTION:**

Datastructures

- n-Number ofprocess,m-number ofresourcetypes.
- Available:Available[j]=k,k–instanceofresourcetypeRjisavailable.Max:Ifmax[i,j]=k,Pimayrequestatmostkinstancesresource Rj.
  Allocation:IfAllocation[i,j]=k,Pi allocatedtokinstancesofresourceRjNeed:IfNeed[I,j]=k,Pimayneedkmoreinstancesof resourcetype Rj,
  Need[I,j]=Max[I,j]-Allocation[I,j];

*SafetyAlgorithm*

Work and Finish be the vector of length m and n respectively, Work=Available andFinish[i]=False.

Findanisuchthat bothFinish[i]=FalseNeed<=Work

IfnosuchIexistsgotostep 4.

**5.** work=work+Allocation,Finish[i]=True;

ifFinish[1]=TrueforallI,thenthesystemisinsafestate

**ALGORITHM:**

1. Starttheprogram.
2. Getthevaluesofresourcesandprocesses.
3. Gettheavail value.
4. Afterallocationfindtheneedvalue.
5. Checkwhetheritspossibletoallocate.
6. Ifitispossiblethenthesystemisinsafestate.
7. Else systemisnotinsafetystate
8. Stop theprocess.

**SOURCECODE :**

```
#include<stdio.h>
#include<conio.h>
voidmain()
{
charjob[10][10];
inttime[10],avail,tem[10],temp[10];intsafe[10];i
nt          ind=1,i,j,q,n,t;
clrscr();
printf("Enternoofjobs:");sca
nf("%d",&n);for(i=0;i<n;i+
+)
{
printf("Enter    name    and    time:
                    ");scanf("%s%d",&job[
i],&time[i]);
}
printf("Enter     the     available
                    resources:");scanf("%d",&avail
);
for(i=0;i<n;i++)
{
temp[i]=time[i];
tem[i]=i;
}
for(i=0;i<n;i++)fo
r(j=i+1;j<n;j++)
{
if(temp[i]>temp[j])
{
```

```
t=temp[i];
```

```
temp[i]=temp[j];
temp[j]=t;t=tem[i];
tem[i]=tem[j];
tem[j]=t;
}
 }
for(i=0;i<n;i++)
{
q=tem[i];if(time[
q]<=avail)
{
safe[ind]=tem[i];avail=ava
il-
tem[q];printf("%s",job[safe
[ind]]);ind++;
}
else
{
printf("Nosafesequence\n");
}
}
printf("Safesequenceis:");f
or(i=1;i<ind;i++)
printf("%s
%d\n",job[safe[i]],time[safe[i]]);getch();
}
```

**OUTPUT:**

```
Enter noofjobs:4
Enter name and time: A
1Enter name and time: B
4Enter name and time: C
2Enternameandtime:D3
Enter the available resources:
20Safesequenceis:A1,C2,D3,B4.
```

**EXPERIMENT.NO11**

**AIM:**ToWriteaCprogramtosimulatediskschedulingalgorithms

a) FCFS b)SCAN c) C-SCAN

**DESCRIPTION**

One of the responsibilities of the operating system is to use the hardware efficiently. For thediskdrives,meetingthisresponsibilityentailshavingfastaccesstimeandlargediskbandwidth.

Both the access time and the bandwidth can be improved by managing the order inwhich disk I/O requests are serviced which is called as disk scheduling. The simplest form ofdisk scheduling is, of course, the first-come, first-served (FCFS) algorithm. This algorithm isintrinsically fair, but it generally does not provide the fastest service. In the SCAN algorithm,the disk arm starts at one end, and moves towards the other end, servicing requests as it reacheseach cylinder, until it gets to the other end of the disk. At the other end, the direction of headmovement is reversed, and servicing continues. The head continuously scans back and forthacross the disk. C-SCAN is a variant of SCAN designed to provide a more uniform wait time.Like SCAN, C-SCAN moves the head from one end of the disk to the other, servicing requestsalong the way. When the head reaches the other end, however, it immediately returns to thebeginningofthe diskwithoutservicinganyrequestsonthereturntrip

**PROGRAM**

**A) FCFSDISKSCHEDULINGALGORITHM**

```
#include<stdio.h>
main()
{
        intt[20],n,I,j,tohm[20],tot=0;floatavhm;clrsc
        r();
        printf("enter     the     no.of
                          tracks");scanf("%d",&
        n);
        printf("enter     the     tracks     to     be
                          traversed");for(i=2;i<n+2;i++)
                scanf("%d",&t*i+);
        for(i=1;i<n+1;i++)
        {
                tohm[i]=t[i+1]-
                t[i];if(tohm[i]<0)toh
                m[i]=tohm[i]*(-1);
        }
        for(i=1;i<n+1;i++)
                tot+=tohm[i];
        avhm=(float)tot/n;
        printf("Tracks     traversed\tDifference     between
                          tracks\n");for(i=1;i<n+1;i++)
                printf("%d\t\t\t%d\n",t*i+,tohm*i+);
                printf("\nAverage      header
                                     movements:%f",avhm);getch();
}
```

***INPUT***

Enterno.oftracks:9

Entertrackposition:55 58 60 70 18 90 150 160184

***OUTPUT***

| Trackstraversed | Difference betweentracks |
|---|---|
| 55 | 45 |
| 58 | 3 |
| 60 | 2 |
| 70 | 10 |
| 18 | 52 |
| 90 | 72 |
| 150 | 60 |
| 160 | 10 |
| 184 | 24 |

Averageheadermovements:30.888889

**B) SCANDISKSCHEDULINGALGORITHM**

```
#include<stdio.h>
main()
{
        intt[20],d[20], h, i, j, n,temp, k,
        atr[20],tot,p,sum=0;clrscr();
        printf("enterthenooftrackstobetraveresed");scanf("%d'
        ",&n);
        printf("enterthepositionofhead");scanf("%
        d",&h);
        t[0]=0;t[1]=h;
        printf("enter the
        tracks");for(i=2;i<n+2;i+
        +)
                scanf("%d",&t[i]);
        for(i=0;i<n+2;i++)
        {
      for(j=0;j<(n+2)-i-1;j++)
      {
      if(t[j]>t[j+1])
        {
      temp=t[j];t[
      j]=t[j+1];t[j
      +1]=temp;
      }}}
      for(i=0;i<n+2;i++)if(t
      [i]==h)
              j=i;k=i;
        p=0;
        while(t[j]!=0)
        {
                atr[p]=t[j];j--
                ;p++;
        }
        atr[p]=t[j];for(p=k+1;p<n+2;p++,
        k++)
                atr[p]=t[k+1];
        for(j=0;j<n+1;j++)
        {
                if(atr[j]>atr[j+1])
                        d[j]=atr[j]-atr[j+1];
                else
                        d[j]=atr[j+1]-atr[j];
                sum+=d[j];
        }
        printf("\nAverageheadermovements:%f",(float)sum/n);
        getch();}
```

***INPUT***

Enterno.oftracks:9

Entertrackposition:55 58 60 70 18 90 150 160184

***OUTPUT***

| Trackstraversed | Difference betweentracks |
|---|---|
| 150 | 50 |
| 160 | 10 |
| 184 | 24 |
| 90 | 94 |
| 70 | 20 |
| 60 | 10 |
| 58 | 2 |
| 55 | 3 |
| 18 | 37 |

Average headermovements:27.77

**C) C-SCANDISKSCHEDULINGALGORITHM**

```
#include<stdio.h>
main()
{
        intt[20],d[20], h, i, j, n,temp, k,
        atr[20],tot,p,sum=0;clrscr();
        printf("enterthenooftrackstobetraveresed");scanf("
        %d'",&n);
        printf("enterthepositionofhead");scanf("%
        d",&h);
        t[0]=0;t[1]=h;
        printf("enter        total
                             tracks");scanf("%d
        ",&tot);
        t[2]=tot-1;
        printf("enter       the
                            tracks");for(i=3;i
        <=n+2;i++)
                scanf("%d",&t[i]);
        for(i=0;i<=n+2;i++)
                for(j=0;j<=(n+2)-i-1;j++)
                        if(t[j]>t[j+1])
                        {
                                temp=t[j];t
                                [j]=t[j+1];t
                                [j+1]=temp
                        }
for(i=0;i<=n+2;i++)
if(t[i]==h);
        j=i;break;
        p=0;
        while(t[j]!=tot-1)
        {
                atr[p]=t[j];
                j++;
                p++;
        }
        atr[p]=t[j];
        p++;
        i=0;
        while(p!=(n+3)&&t[i]!=t[h])
        {
                atr[p]=t[i];i++;
                p++;
        }
```

```
        for(j=0;j<n+2;j++)
        {
                if(atr[j]>atr[j+1])
                        d[j]=atr[j]-atr[j+1];
                else
                        d[j]=atr[j+1]-atr[j];
                sum+=d[j];
        }
        printf("totalheadermovements%d",sum);p
         rintf("avgis%f",(float)sum/n);
        getch();
}
```

***INPUT***

Enterthetrackposition:55 58 60 70 18 90 150 160 184Enterstartingposition:100

***OUTPUT***

| Trackstraversed | Difference Betweentracks |
|---|---|
| 150 | 50 |
| 160 | 10 |
| 184 | 24 |
| 18 | 240 |
| 55 | 37 |
| 58 | 3 |
| 60 | 2 |
| 70 | 10 |
| 90 | 20 |

Averageseek time: 35.7777779