HYDERABAD INSTITTE OF TECHNOLOGY AND MANAGEMENT

DEPARTMENT OF EMERGING TECHNOLOGY

Operating Systems

LAB MANUAL

Subject Code	: 22PC3CS03
Regulation	: R22/HITAM
Academic Year	: 2023-2024
Year & Sem	: II - I

Prepared by
Periasamy S

DEPARTMENT OF COMPUTERSCIENCE AND ENGINEERING

PROGRAM EDUCATIONAL OBJECTIVES (PEOS):

A graduate of the Computer Science and Engineering Program should:

PEO1	**Program Educational Objective1: (PEO1)** The Graduates will provide solutions to difficult and challenging issues in their professionby applying computer science and engineering theory and principles.
PEO2	**Program Educational Objective2 :(PEO2)** **The** Graduates have successful careers in computer science and engineering fields or will beable to successfully pursue advanced degrees.
PEO3	**Program Educational Objective3: (PEO3)** The Graduates will communicate effectively, work collaboratively and exhibit high levels of Professionalism, moral and ethical responsibility.
PEO4	**Program Educational Objective4 :(PEO4)** **The** Graduates will develop the ability to understand and analyse Engineering issues in abroader perspective with ethical responsibility towards sustainable development.

PROGRAM OUTCOMES (POS):

PO1	**Engineeringknowledge**:Applytheknowledgeofmathematics,science,engineering Fundamentals andanengineeringspecializationtothesolutionofcomplexengineeringproblems.
PO2	**Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusionsusing first principles of mathematics, natural sciences, and engineering sciences.
PO3	**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.
PO4	**Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis ofthe information to provide valid conclusions.
PO5	**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.
PO6	**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 tothe professional engineering practice.
PO7	**Environment and sustainability**: Understand the impact of the professional engineering Solutions in societal and environmental contexts, and demonstrate the knowledge of, and needfor sustainable development.
PO8	**Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
PO9	**Individual and team work**: Function effectively as an individual, and as a member or leaderIn diverse teams, and in multi-disciplinary settings.
PO10	**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 receiveclear instructions.
PO11	**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.
PO12	**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.

PROGRAM SPECIFIC OUTCOMES(PSOS):

PSO1	**Problem Solving Skills** – Graduate will be able to apply computational techniques and software principles to solve complex engineering problems pertaining to software engineering.
PSO2	**Professional Skills** – Graduate will be able to think critically, communicate effectively, and collaborate in teams through participation in co and extra-curricular activities.
PSO3	**Successful Career** – Graduates will possess a solid foundation in computer science and engineering that will enable them to grow in their profession and pursue lifelong learningthrough post-graduation and professional development.

HR22

B. Tech II Year–III Sem	L	T	P	C
Subject Code: 22PC3CS17	0	0	2	1

Operating Systems Lab

Prerequisites:

A course on "Programming for Problem Solving".
A course on "Computer Organization and Architecture".

Course Objectives:

1. To provide an understanding of the design aspects of operating system concepts through Simulation.
2. Introduce basic Unix commands, system call interface for process management, inter process communication and I/O in Unix.

Course Outcomes:

1. Simulate and implement operating system concepts such as scheduling, deadlock management, file management and memory management.
2. Able to implement C programs using Unix system calls

List of Experiments

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

   a) FCFS b) SJF Round Robin d) priority

2. Write a C program to simulate Bankers Algorithm for Deadlock Avoidance and Prevention.
3. Write a C program to implement the Producer – Consumer problem using semaphores.
4. Write a C program to simulate the concept of Dining-philosophers problem.
5. Write C programs to simulate the following memory management techniques

   a) Paging b) Segmentation

6. Write C programs to illustrate the following IPC mechanisms

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

7. Write a C program to simulate the following contiguous memory allocation Techniques

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

8. Simulate all File Organization Techniques

a) Single level directory b) Two level directory

9. Write a C program to simulate the following contiguous memory allocation Techniques

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

10. Implementation of the following Page Replacement Algorithms

a) FIFO b) LRU c) LFU

TEXT BOOKS:

1. Operating System Principles , Abraham Silberchatz, Peter B. Galvin, Greg Gagne, 8th Edition, Wiley Student Edition
2. Operating Systems – Internals and Design Principles, W. Stallings, 6th Edition, Pearson.

REFERENCE BOOKS:

1. Modern Operating Systems, Andrew S Tanenbaum, 3rd Edition, PHI
2. Operating Systems A concept-based Approach, 2nd Edition, D.M.Dhamdhere, TMH.
3. Principles of Operating Systems, B.L.Stuart, Cengage learning, India Edition.

CO-PO &PSO Mapping:

Course Name - Course Outcomes / Program Outcomes	1	2	3	4	5	6	7	8	9	10	11	12	PSO1	PSO2	PSO3
CO1	M	L													
CO2	H	H			M										
CO3	M	H													
CO4	M	L			L										

EXPERIMENT-1

WriteaCprogramsimulatethefollowingCPUschedulingalgorithms:
a)FCFS b)SJF c)RoundRobin d)Priority

a)FCFS

DESCRIPTION

Assumealltheprocessesarriveat thesametime.

FCFSCPUSCHEDULINGALGORITHM

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

CPUSCHEDULING

Maximum CPU utilization obtained with multiprogramming CPU–

I/O Burst Cycle – Process executionconsists ofa *cycle* of CPU

execution and I/O wait

CPUburstdistribution

a) First-Come,First-Served(FCFS)Scheduling

Process	BurstTime
P1	24
P2	3
P3	3

Supposethattheprocessesarrive intheorder:P1,P2,P3 The

Gantt Chart for the schedule is:

Waitingtime for P1=0;P2=24;P3=27

Average waiting time: (0 + 24 + 27)/3 = 17

ALGORITHM

1. Start
2. Declarethearraysize
3. Readthenumberofprocessestobeinserted
4. ReadtheBurst timesofprocesses
5. calculatethewaitingtimeofeachprocess
   wt[i+1]=bt[i]+wt[i]
6. calculatetheturnaroundtimeofeachprocess
   tt[i+1]=tt[i]+bt[i+1]
7. Calculatetheaveragewaitingtimeandaverageturnaroundtime.
8. Displaythe values
9. Stop

PROGRAM:

```
#include<stdio.h>voi
d main()
{
inti,j,bt[10],n,wt[10],tt[10],w1=0,t1=0;
float aw,at;
printf("enterno.ofprocesses:\n");
scanf("%d",&n);
printf("entertheburstttimeofprocesses:");
for(i=0;i<n;i++)
scanf("%d",&bt[i]);
for(i=0;i<n;i++)
{
wt[0]=0;
tt[0]=bt[0];
wt[i+1]=bt[i]+wt[i];
tt[i+1]=tt[i]+bt[i+1];
w1=w1+wt[i];
t1=t1+tt[i];
}
aw=w1/n;
at=t1/n;
printf("\nbt\twt\ttt\n");
for(i=0;i<n;i++)
printf("%d\t%d\t%d\n",bt[i],wt[i],tt[i]);
printf("aw=%f\n,at=%f\n",aw,at);
}
```

INPUT

Enternoofprocesses 3
enterbursttime
12
8
20

EXPECTEDOUTPUT

btwttt

12012
81220
20 20 40
aw=10.666670
at=24.00000

b) SJF

SJFCPUSCHEDULINGALGORITHM

ForSJFschedulingalgorithm,readthenumberofprocesses/jobsinthesystem,theirCPUbursttimes.
Arrange all the jobs in order withrespect to their burst times. There may be two jobs in queue with thesameexecutiontime,andthenFCFSapproachistobeperformed.Eachprocesswillbeexecuted accordingtothelengthofitsbursttime.Thencalculatethewaitingtimeandturnaroundtimeofeach of the processes accordingly.

HARDWAREREQUIREMENTS: IntelbasedDesktopPc
RAMof512MB

SOFTWAREREQUIREMENTS:
TurboC/BorlandC.

THEORY:

ExampleofNonPreemptiveSJF

Process	ArrivalTime	BurstTime
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	3.0	4

P1	P3	P2	P4
0 7	8	12	16

ExampleofPreemptiveSJF

Process	ArrivalTime	BurstTime
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	3.0	4

P1	P2	P3	P2	P4	P1

Averagewaitingtime=(9+1+0+2)/4=3 ALGORITHM

1. Start
2. Declarethearraysize
3. Readthenumberofprocessestobeinserted
4. ReadtheBurst timesofprocesses
5. sorttheBursttimes inascendingorderandprocesswithshortestbursttimeis firstexecuted.
6. calculatethewaitingtimeofeachprocess
   wt[i+1]=bt[i]+wt[i]
7. calculatetheturnaroundtimeofeachprocess
   tt[i+1]=tt[i]+bt[i+1]
8. Calculate the average waiting timeandaverageturnaroundtime.
9. Display the values
10. Stop

PROGRAM:

```
#include<stdio.h>voi
d main()
{
inti,j,bt[10],t,n,wt[10],tt[10],w1=0,t1=0;
float aw,at;
printf("enterno.ofprocesses:\n");
scanf("%d",&n);
printf("entertheburstttimeofprocesses:");
for(i=0;i<n;i++)
scanf("%d",&bt[i]);
for(i=0;i<n;i++)
{
for(j=i;j<n;j++)
if(bt[i]>bt[j])
{
t=bt[i];
bt[i]=bt[j];
bt[j]=t;
}
}
for(i=0;i<n;i++)
printf("%d",bt[i]);
for(i=0;i<n;i++)
{
wt[0]=0;
tt[0]=bt[0];
```

```
wt[i+1]=bt[i]+wt[i];
tt[i+1]=tt[i]+bt[i+1];
w1=w1+wt[i];
t1=t1+tt[i];
}
aw=w1/n;
at=t1/n;
printf("\nbt\twt\ttt\n");
for(i=0;i<n;i++)
printf("%d\t%d\t%d\n",bt[i],wt[i],tt[i]);
printf("aw=%f\n,at=%f\n",aw,at);
}
```

INPUT:

```
enternoofprocesses 3
enterbursttime
12
8
20
```

OUTPUT:

```
bt  wt  tt
12820
808
202040
aw=9.33
at=22.64
```

c) RoundRobin

DESCRIPTION

Assumealltheprocessesarriveat thesametime.

ROUNDROBINCPUSCHEDULING ALGORITHM

For round robin scheduling algorithm, read the number of processes/jobs in the system, their CPU burst times, and the size ofthetime slice. Time slicesare assigned to eachprocess inequalportions and in circular order, handling all processes execution. This allows every process to get an equal chance. Calculate the waiting time and turnaround time of each of the processes accordingly.

HARDWAREREQUIREMENTS: IntelbasedDesktopPcRAMof512MB

SOFTWAREREQUIREMENTS:TurboC/BorlandC.

THEORY:

RoundRobin:

ExampleofRRwith timequantum=3

Process	Bursttime
aaa	4
Bbb	3
Ccc	2
Ddd	5
Eee	1

ALGORITHM

1. Start
2. Declarethearraysize
3. Readthenumberofprocessestobeinserted
4. Readthe burst timesofthe processes
5. ReadtheTimeQuantum
6. ifthe bursttimeofaprocess isgreaterthantimeQuantumthensubtracttimequantumformthe burst time

   Else

   Assigntheburst timetotime quantum.
7. calculatetheaveragewaitingtimeand turnaround timeoftheprocesses.
8. Displaythe values
9. Stop

PROGRAM:

```
#include<stdio.h>voi
d main()
{
int st[10],bt[10],wt[10],tat[10],n,tq;
inti,count=0,swt=0,stat=0,temp,sq=0;
float awt=0.0,atat=0.0;
printf("Enternumberofprocesses:");
scanf("%d",&n);
printf("Enterbursttimeforsequences:");
for(i=0;i<n;i++)
{
scanf("%d",&bt[i]);
st[i]=bt[i];
}
printf("Entertimequantum:");
scanf("%d",&tq);
while(1)
{
for(i=0,count=0;i<n;i++)
{
temp=tq;
if(st[i]==0)
{
count++;
continue;
}
if(st[i]>tq)
st[i]=st[i]-tq;
else
if(st[i]>=0)
{
temp=st[i];
st[i]=0;
}
sq=sq+temp;
tat[i]=sq;
}
if(n==count)
break;
}
for(i=0;i<n;i++)
{
wt[i]=tat[i]-bt[i];
swt=swt+wt[i];
```

```
stat=stat+tat[i];
}
awt=(float)swt/n;
atat=(float)stat/n;
printf("Process_noBursttimeWaittimeTurnaroundtime");
for(i=0;i<n;i++)
printf("\n%d\t %d\t%d\t%d",i+1,bt[i],wt[i],tat[i]);
printf("\nAvgwaittimeis%fAvgturnaroundtimeis %f",awt,atat);
}
```

Input:

```
Enternoofjobs 4
Enterbursttime
5
12
8
20
```

Output:

```
Btwttt
5 0 5
12513
81325
20 25 45
aw=10.75000
at=22.000000
```

d) Priority

HARDWAREREQUIREMENTS: Intelbased DesktopPc
RAMof512MB

SOFTWAREREQUIREMENTS:
TurboC/BorlandC.

THEORY:

InPriorityScheduling, eachprocess isgivenapriority, andhigherprioritymethodsareexecuted first, while equal priorities are executed First Come First Servedor Round Robin.

Thereareseveralwaysthatprioritiescanbe assigned:

- Internalprioritiesareassignedbytechnicalquantitiessuchasmemoryusage,and file/IO operations.
- Externalprioritiesareassignedbypolitics,commerce,oruserpreference,suchas importanceandamount beingpaid forprocessaccess(the latterusuallybeing for mainframes).

ALGORITHM

1. Start
2. Declarethearraysize
3. Readthenumberofprocessestobeinserted
4. ReadthePrioritiesofprocesses
5. sorttheprioritiesandBursttimesinascending order
5. calculatethewaitingtimeofeachprocess
   wt[i+1]=bt[i]+wt[i]
6. calculatetheturnaroundtimeofeachprocess
   tt[i+1]=tt[i]+bt[i+1]
7. Calculatetheaveragewaitingtimeandaverageturnaroundtime.
8. Displaythe values
9. Stop

PROGRAM:

```
#include<stdio.h>voi
d main()
 {
  int i,j,pno[10],prior[10],bt[10],n,wt[10],tt[10],w1=0,t1=0,s;
  float aw,at;
  printf("enterthenumberofprocesses:");
  scanf("%d",&n);
  for(i=0;i<n;i++)
  {
  printf("The process %d:\n",i+1);
   printf("Enterthebur…
```

```
        at=t1/n;
        }
        printf("\njob\t bt\twt \ttat\t prior\n");
          for(i=0;i<n;i++)
           printf("%d\t%d\t%d\t%d\t%d\n",pno[i],bt[i],wt[i],tt[i],prior[i]);
           printf("aw=%f \t at=%f \n",aw,at);

  }
```

Input:

```
Enternoofjobs 4
Enterbursttime
10
2
4
7
Enterpriorityvalues 4
2
1
3
```

Output:

```
Btprioritywttt 4
1 0 4
2246
73613
10 4 13 23
aw=5.750000
at=12.500000
```

VIVA QUESTIONS:

1. RoundRobinscheduling isusedin

(A)Diskscheduling.(B)CPUscheduling

(C) I/Oscheduling. (D)Multitasking

2. Whatarethedis-advantagesofRRScheduling Algoritm?

3. WhataretheadvantagesofRRScheduling Algoritm?

4. Super computers typicallyemploy_____.

1RealtimeOperatingsystem2MultiprocessorsOS 3

desktop OS 4 None of the above

5. Anoptimalschedulingalgorithmintermsofminimizingtheaveragewaitingtimeofagivenset of processes is _____.

1 FCFS scheduling algorithm 2Roundrobinschedulingalgorithm 3

Shortest job - first scheduling algorithm4 None of the above

6. TheoptimumCPU scheduling algorithmis

(A)FIFO (B)SJFwithpreemption.(C)SJFwithoutpreemption.(D)RoundRobin.

7. Intermsofaveragewait timetheoptimumschedulingalgorithmis

(A)FCFS(B)SJF (C)Priority(D)RR

8. Whatarethedis-advantagesofSJFSchedulingAlgorithm?

9. WhataretheadvantagesofSJFSchedulingAlgorithm?

10. DefineCPUSchedulingalgorithm?

11. WhatisFirst-Come-First-Served(FCFS)Scheduling?

12. WhyCPUschedulingisrequired?

13. Whichtechniquewasintroducedbecauseasingle jobcould not keepboththeCPUandtheI/O devices busy?

1)Time-sharing2)SPOOLing3)Preemptivescheduling4) Multiprogramming

14. CPUperformance is measuredthrough_____.

1) Throughput2) MHz3) Flaps4) Noneoftheabove

15. Whichofthefollowingisacriteriontoevaluateaschedulingalgorithm?

1 CPUUtilization:KeepCPUutilizationas highaspossible.

2 Throughput:numberofprocessescompletedperunittime.

3 WaitingTime:Amountoftimespentreadytorunbutnotrunning.

4 Alloftheabove

16. PriorityCPUschedulingwould mostlikelybeused ina_____os.
17. CPU allocated process to _____priority.
18. Calculateavgwaitingtime=
19. MaximumCPU utilizationobtained with_____
20. Using _____algorithmsfindthemin&maxwaiting time.`]\oiui

EXPERIMENT-2

WriteaCprogramtosimulateBankersAlgorithmforDeadlockAvoidance andPrevention

OBJECTIVE

WriteaCprogramto simulateBankersAlgorithmfor Deadlock Avoidance

DESCRIPTION

Inamultiprogrammingenvironment,severalprocessesmaycompeteforafinitenumberofresources. A process requests resources; if the resources are not available at that time, the process enters a waitingstate.Sometimes,awaitingprocessisneveragainabletochangestate,becausetheresources it has requested are 4held by other waiting processes. This situation is called a deadlock. Deadlock avoidance isoneofthetechniques forhandlingdeadlocks. Thisapproachrequiresthattheoperating systembegiveninadvanceadditionalinformationconcerningwhichresourcesaprocesswillrequest anduseduringitslifetime.Withthisadditionalknowledge, itcandecideforeachrequestwhetheror nottheprocessshouldwait.Todecidewhetherthecurrentrequestcanbesatisfiedormustbedelayed, the systemmust consider the resources currentlyavailable, the resources currentlyallocatedto each process, and the future requests and releases of each process.

Banker's algorithm is a deadlock avoidance algorithm that is applicable to a system with multiple instances of each resource type.

NAMEOFEXPERIMENT:SimulateBanker'sAlgorithmforDeadlockAvoidance.

AIM: Simulate Banker's Algorithm for Deadlock Avoidance to find whether the systemis insafe state or not.

HARDWAREREQUIREMENTS: Intelbased DesktopPc

RAM of 512 MB

SOFTWAREREQUIREMENTS:TurboC/BorlandC. **THEORY:**

DEADLOCKAVOIDANCE

Toimplementdeadlockavoidance&PreventionbyusingBanker'sAlgorithm.

Banker'sAlgorithm:

When a new process enters a system, it must declare the maximum number of instances of each resourcetype it needed. This number mayexceed the totalnumber ofresources in the system. When the user request a set ofresources, the system must determine whether the allocation ofeach resources will leave the system in safe state. If it will the resources are allocation; otherwise the process must wait until some other process release the resources.

Datastructures

- n-Numberofprocess,m-numberofresourcetypes.
- Available:Available[j]=k, k–instanceofresourcetypeRjisavailable.
- Max:Ifmax[i,j]=k, Pimayrequestatmostkinstancesresource Rj.
- Allocation:IfAllocation[i,j]=k,Piallocatedto kinstancesofresource Rj
- Need:IfNeed[I,j]=k,Pimay needkmoreinstancesofresourcetypeRj,

Need[I,j]=Max[I,j]-Allocation[I,j];

Safety Algorithm

1. WorkandFinishbethevectoroflengthmand nrespectively, Work=Availableand Finish[i] =False.
2. Findanisuchthatboth
   - Finish[i]=False
   - Need<=Work

   IfnosuchIexistsgotostep4.
3. work=work+Allocation,Finish[i]=True;
4. ifFinish[1]=TrueforallI,thenthesystemisinsafestate.

Resourcerequestalgorithm

Let Request iberequest vector fortheprocessPi, Ifrequest i=[j]=k, thenprocessPiwantsk instances of resource type Rj.

1. ifRequest<=NeedIgotostep2.Otherwiseraise anerrorcondition.
2. ifRequest<=Availablego to step3.OtherwisePimustsincetheresourcesareavailable.
3. Havethesystempretendtohaveallocatedtherequestedresourcestoprocess Pibymodifying the state as follows;
   Available=Available-Request I;
   AllocationI=Allocation+RequestI;
   Need i=Need i-Request I;

Iftheresultingresourceallocationstateissafe, thetransactioniscompletedandprocessPiisallocated its resources. However if the state is unsafe, the Pi must wait for Request i and the old resource-allocation state is restored.

ALGORITHM:

1. Starttheprogram.
2. Getthevaluesofresourcesandprocesses.
3. Gettheavailvalue.
4. Afterallocationfindtheneed value.
5. Checkwhetheritspossibletoallocate.
6. Ifitispossiblethenthesystemis insafe state.
7. Elsesystemisnot insafetystate.
8. Ifthenewrequestcomesthencheck thatthesystemisinsafety.
9. ornotifweallowtherequest.
10. stoptheprogram.

PROGRAM:

```
#include<stdio.h>stru
ct da {
intmax[10],al[10],need[10],before[10],after[10];
}p[10];
voidmain(){
inti,j,k,l,r,n,tot[10],av[10],cn=0,cz=0,temp=0,c=0;
printf("\n Enter the no of processes:");
scanf("%d",&n);
printf("\nEnterthenoofresources:");
scanf("%d",&r);
for(i=0;i<n;i++) {
printf("process%d\n",i+1);
for(j=0;j<r;j++) {
printf("maximumvalueforresource%d:",j+1);
scanf("%d",&p[i].max[j]);
}
for(j=0;j<r;j++){
printf("allocatedfromresource%d:",j+1);
scanf("%d",&p[i].al[j]);
p[i].need[j]=p[i].max[j]-p[i].al[j];
}
}
for(i=0;i<r;i++){
printf("Entertotalvalueofresource%d:",i+1);
scanf("%d",&tot[i]);
}
for(i=0;i<r;i++) {
for(j=0;j<n;j++)
temp=temp+p[j].al[i];
av[i]=tot[i]-temp;
temp=0;
}
printf("\n\tmaxallocatedneededtotalavail"); for(i=0;i<n;i++)
{
printf("\n P%d \t",i+1);
for(j=0;j<r;j++)
printf("%d",p[i].max[j]);
printf("\t");
for(j=0;j<r;j++)
printf("%d",p[i].al[j]);
printf("\t");
for(j=0;j<r;j++)
printf("%d",p[i].need[j]);
printf("\t");
for(j=0;j<r;j++)
{
```

```
if(i==0)
printf("%d",tot[j]);
}
printf("");
for(j=0;j<r;j++) {
if(i==0)
printf("%d",av[j]);
}
}
printf("\n\n\tAVAILBEFORE\tAVAILAFTER"); for(l=0;l<n;l++)
{
for(i=0;i<n;i++)
{
for(j=0;j<r;j++)
{
if(p[i].need[j]>av[j])
cn++;
if(p[i].max[j]==0)
cz++;
}
if(cn==0&&cz!=r)
{
for(j=0;j<r;j++)
{
p[i].before[j]=av[j]-p[i].need[j];
p[i].after[j]=p[i].before[j]+p[i].max[j];
av[j]=p[i].after[j];
p[i].max[j]=0;
}
printf("\n p%d \t",i+1);
for(j=0;j<r;j++)
printf("%d",p[i].before[j]);
printf("\t");for(j=0;j<r;j++)
printf("%d",p[i].after[j]);
cn=0;
cz=0;
c++;
break;
}
else {
cn=0;cz=0;
}
}
}
if(c==n)
```

```
printf("\ntheabovesequenceisasafesequence"); else
printf("\ndeadlockoccured");
}
```

OUTPUT:

```
//TESTCASE 1:

 ENTERTHENO.OFPROCESSES:4

 ENTERTHENO.OFRESOURCES:3
PROCESS 1
MAXIMUMVALUEFORRESOURCE1:3
MAXIMUMVALUEFORRESOURCE2:2
MAXIMUMVALUEFORRESOURCE3:2
ALLOCATED FROM RESOURCE 1:1
ALLOCATED FROM RESOURCE 2:0
ALLOCATED FROM RESOURCE 3:0
PROCESS 2
MAXIMUMVALUEFORRESOURCE1:6
MAXIMUMVALUEFORRESOURCE2:1
MAXIMUMVALUEFORRESOURCE3:3
ALLOCATED FROM RESOURCE 1:5
ALLOCATED FROM RESOURCE 2:1
ALLOCATED FROM RESOURCE 3:1
PROCESS 3
MAXIMUMVALUEFORRESOURCE1:3
MAXIMUMVALUEFORRESOURCE2:1
MAXIMUMVALUEFORRESOURCE3:4
ALLOCATED FROM RESOURCE 1:2
ALLOCATED FROM RESOURCE 2:1
ALLOCATED FROM RESOURCE 3:1


PROCESS4
MAXIMUM VALUE FOR RESOURCE 1:4
MAXIMUM VALUE FOR RESOURCE 2:2
MAXIMUM VALUE FOR RESOURCE 3:2
ALLOCATED FROM RESOURCE 1:0
ALLOCATED FROM RESOURCE 2:0
ALLOCATED FROM RESOURCE 3:2
ENTERTOTALVALUEOFRESOURCE1:9
ENTERTOTALVALUEOFRESOURCE2:3
```

ENTERTOTALVALUEOFRESOURCE3:6

RESOURCES	ALLOCATED		NEEDED	TOTAL	AVAIL
P1	322	100	222	936	112
P2	613	511	102		
P3	314	211	103		
P4	422	002	420		

AVAILBEFORE		AVAILAFTER
P2	010	623
P1	401	723
P3	620	934
P4	514	936

THEABOVESEQUENCEISASAFE SEQUENCE

VIVA QUESTIONS:

1. Differentiatedeadlockavoidanceandfragmentation
2. Tellmetherealtimeexamplewherethisdeadlockoccurs?
3. Howdowecalculatetheneedforprocess?
4. Whatisthenameofthealgorithmtoavoid deadlock?
5. Banker'salgorithmforresourceallocationdealswith

   (A)Deadlock prevention. (B)Deadlock avoidance.

   (C)Deadlock recovery. (D)Mutual exclusion

6. Each request requires that the system consider the _____todecidewhetherthecurrent request can be satisfied or must wait to avoid a future possible deadlock.
7. Given a priori information about the _____numberofresourcesofeachtypethat maybe requestedfor eachprocess, it ispossibleto construct analgorithmthat ensuresthatthesystemwill never enter a deadlock state.
8. A deadlock avoidance algorithm dynamically examines the _____toensurethatacircular wait condition can never exist.

9. DefineSafeState.

10. Asystemisina safe stateonlyifthere exists a_____.

11. IsAllunsafestatesaredeadlocks?

12. Asystemhas12magnetictapedrivesand3processes:P0,P1,andP2. ProcessP0requires10 tape drives, P1 requires 4 and P2 requires 9 tape drives.

Process
P0
P1
P2
Maximumneeds(process-wise:P0throughP2topto bottom) 10
4
9
Currentlyallocated(process-wise)
5
2
2
Whichofthefollowingsequenceisasafesequence?
a) P0,P1,P2
b) P1,P2,P0
c) P2,P0,P1
d) P1,P0,P2

13. Ifnocycle exists inthe resource allocationgraphthen_____

14. Theresourceallocationgraphisnotapplicabletoaresourceallocationsystem:with _____

15. TheBanker'salgorithmis_____thantheresourceallocationgraphalgorithm.

16. Listdata structures available in the Banker's algorithm.

Thedatastructuresavailable intheBanker'salgorithmare:
a) Available
b) Need
c) Allocation

17. WhatisthecontentofthematrixNeedis:

18. A systemwith 5 processes P0 through P4 and three resource types A, B, C has A with 10 instances,Bwith5 instances,andCwith7instances.Attimet0,the followingsnapshothasbeen taken :

Process
P0
P1
P2
P3
P4
Allocation(process-wise:P0throughP4topTObottom) ABC
010
200
302

211
002
MAX(process-wise:P0throughP4topTObottom) ABC
753
322
902
222
433
Available
ABC
332
Thesequence<P1,P3,P4,P2,P0>leadsthesystemto:
a) anunsafestate
b) asafestate
c) a protectedstate
d) adeadlock
Answer:
19. Thewait-for graphisadeadlockdetectionalgorithmthatisapplicablewhen:_____
20. What doesanedgefromprocessPitoPjinawait forgraphindicates?: Pi is waiting for Pj to release a resource that Pi needs.
21. Ifthewaitfor graphcontainsacycle:_____.
22. Ifdeadlocksoccur frequently, the detectionalgorithmmustbe invoked_____.
23. The disadvantage ofinvoking the detectionalgorithmfor everyrequestis_____
24. 'm'processesshare'n'resourcesofthesametype. The maximumneedofeachprocess doesn't exceed'n'andthesumofalltheir maximumneeds isalways lessthanm+n. Inthissetup, deadlock :
a) canneveroccur
b) mayoccur
c) hastooccur
d) none ofthementioned
20. Asystemhas3processessharing4resources.Ifeachprocessneedsa maximumof2unitsthen, deadlock :
a) canneveroccur
b) mayoccur
c) hastooccur
d) none ofthementioned

OBJECTIVE

WriteaCprogramto simulateBankersalgorithmforDeadlockPrevention

DESCRIPTION

Inamultiprogrammingenvironment,severalprocessesmaycompeteforafinitenumberofresources. A process requests resources; if the resources are not available at that time, the process enters a waitingstate.Sometimes,awaitingprocessisneveragainabletochangestate,becausetheresources it has requested are held by other waiting processes. This situation is called a deadlock. Deadlock avoidance isoneofthetechniques forhandlingdeadlocks. Thisapproachrequiresthattheoperating systembegiveninadvanceadditionalinformationconcerningwhichresourcesaprocesswillrequest anduseduringitslifetime.Withthisadditionalknowledge, itcandecideforeachrequestwhetheror nottheprocessshouldwait.Todecidewhetherthecurrentrequestcanbesatisfiedormustbedelayed, the systemmust consider the resources currentlyavailable, the resources currentlyallocatedto each process, and the future requests and releases of each process.

Banker's algorithm is a deadlock avoidance algorithm that is applicable to a system with multiple instances of each resource type.

NAMEOFEXPERIMENT:SimulateAlgorithmforDeadlockprevention. AIM: Simulate Algorithm for Deadlock prevention .

HARDWAREREQUIREMENTS: Intelbased DesktopPc
RAMof512MB

SOFTWAREREQUIREMENTS:TurboC/BorlandC.

THEORY:

DeadlockDefinition:

A set of processes is deadlocked if each process in the set is waiting for an event that only another process in the set can cause (including itself).Waiting for an event could be:

- waitingforaccesstoacriticalsection
- waitingforaresourceNotethatitisusuallyanon-preemptable (resource).
-

ConditionsforDeadlock :

- Mutualexclusion:resourcescannotbe shared.
- Holdandwait:processesrequestresourcesincrementally,andholdonto What they've got.
- Nopreemption:resourcescannotbeforcibly takenfromprocesses.
- Circularwait:circularchainofwaiting, inwhicheachprocessiswaiting fora resource held by the next process in the chain.

StrategiesfordealingwithDeadlock:

- ignoretheproblemaltogether
- detectionandrecovery
- avoidancebycarefulresource allocation
- preventionbystructurally negatingoneofthefournecessaryconditions.

DeadlockPrevention:

Difference from avoidance is that here, the system itself is built in such a way that there are no deadlocks. Make sure atleast one ofthe 4 deadlock conditions is never satisfied. This may however be even more conservative than deadlock avoidance strategy.

Algorithm:

1. Start
2. AttackingMutexcondition:nevergrantexclusiveaccess.butthismaynotbepossibleforseveral resources.
3. Attackingpreemption: notsomethingyouwanttodo.
4. Attacking holdandwait condition: makeaprocess holdatthe most 1resourceat atime.makeall the requests at the beginning. All or nothing policy. If you feel,retry. eg. 2-phase locking
5. Attackingcircularwait:Orderalltheresources.Makesurethattherequestsareissuedinthecorrect ordersothattherearenocyclespresent intheresourcegraph.Resourcesnumbered1...n.Resources canberequestedonlyinincreasingorder.ie.youcannotrequestaresourcewhosenoislessthanany you may be holding.
6. Stop

PROGRAM:

```
#include<stdio.h>
int max[10][10],alloc[10][10],need[10][10],avail[10],i,j,p,r,finish[10]={0},flag=0;
main()
{
printf("\n\nSIMULATIONOFDEADLOCKPREVENTION");
printf("Enter no. of processes, resources");
scanf("%d%d",&p,&r);printf("Enterallocationmatrix");
for(i=0;i<p;i++)
for(j=0;j<r;j++)
scanf("%d",&alloc[i][j]);
printf("entermaxmatrix");
for(i=0;i<p;i++)/*readingthemaximummatrixandavailalematrix*/ for(j=0;j<r;j++)
scanf("%d",&max[i][j]);
printf("enteravailablematrix");
for(i=0;i<r;i++)
scanf("%d",&avail[i]);
```

```
for(i=0;i<p;i++)
for(j=0;j<r;j++)
need[i][j]=max[i][j]-alloc[i][j];
fun(); /*calling function*/
if(flag==0)
{i
f(finish[i]!=1)
{
printf("\n\nFailing:Mutualexclusion");
for(j=0;j<r;j++)
{/*checkingformutualexclusion*/
if(avail[j]<need[i][j])
avail[j]=need[i][j];
}fun();
printf("\nByallocatingrequiredresourcesto process%ddeadlock isprevented",i); printf("\n\n lack
of preemption");
for(j=0;j<r;j++)
{
if(avail[j]<need[i][j])
avail[j]=need[i][j];
alloc[i][j]=0;
}

fun();
printf("\n\ndaedlockispreventedbyallocatingneededresources");
printf(" \n \n failing:Hold and Wait condition ");
for(j=0;j<r;j++)
{/*checkingholdandwaitcondition*/
if(avail[j]<need[i][j])
avail[j]=need[i][j];
}
fun();
printf("\nAVOIDINGANYONEOFTHECONDITION,UCANPREVENTDEADLOCK");
}
}
}
fun()
{
while(1)
{
for(flag=0,i=0;i<p;i++)
{
if(finish[i]==0)
{
for(j=0;j<r;j++)
{
if(need[i][j]<=avail[j])
```

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

Output:

```
SIMULATIONOFDEADLOCKPREVENTION
Enterno.ofprocesses,resources3,2
enter allocation matrix 2 4 5
                              345
Entermaxmatrix434
                          561
  Enteravailablematrix2
                              5
Failing : MutualExclusion
byallocatingrequiredresourcestoprocessdeadisprevented
Lackofnopreemptiondeadlockis prevented byallocating needed resources Failing
: Hold and Wait condition
```

VIVA QUESTIONS:

1. The Banker's algorithmis usedfor _____.
2. _____isthesituationinwhichaprocessiswaitingon anotherprocess,whichisalso waiting on another process ... which is waiting on the first process. None ofthe processes involved in this circular wait are making progress.
3. whatissafestate?
4. Whataretheconditionsthatcausedeadlock?
5. Howdowecalculatetheneedforprocess?
6. Thenumber ofresourcesrequestedbyaprocess:mustnotexceedthetotalnumberof _____
7. The request and release ofresources are_____
8. Multithreadedprogramsare:_____.
9. Foradeadlocktoarise,whichconditionsmust holdsimultaneously
   a) Mutualexclusion b) Nopreemption c)Holdandwait
10. For _____to prevailinthesystemat leastoneresourcemust beheld ina nonsharable mode.
11. ForaHoldandwaitconditiontoprevailwhatisrequire?
12. _____isasetofmethodsto ensurethatat leastoneofthenecessaryconditionscannot hold.
13. For _____resourceslikeaprinter,mutualexclusionmustexist.
14. Forsharableresources,mutualexclusionisnot required.
15. Toensurethattheholdandwaitconditionneveroccursinthesystem,itmustbeensured what?
    a) wheneveraresourceisrequestedbyaprocess,itisnotholdinganyother resources
    b) eachprocessmustrequestandbeallocatedallitsresourcesbeforeitbegins itsexecution
    c) aprocesscanrequestresourcesonlywhenithasnone
16. Thedisadvantageofaprocessbeingallocatedallitsresourcesbeforebeginningitsexecutionis _____
17. To ensure_____,ifaprocessisholdingsomeresourcesandrequestsanotherresource that cannot be immediatelyallocated to it then all resources currently being held are pre-empted.
18. A _____canbebrokenbyabortoneormoreprocessestobreak thecircularwait.
19. Whatarethetwowaysofabortingprocessesandeliminatingdeadlocks?:
20. Thoseprocessesshouldbeabortedonoccurrenceofadeadlock,theterminationofwhich :_____.

EXPERIMENT-3

Writeaprogramtoimplementthe Producer –Consumerproblemusingsemaphoresusing UNIX/LINUX system calls.

OBJECTIVE

ToimplementtheProducer –ConsumerproblemusingsemaphoresusingUNIX/LINUXsystem calls.

DESCRIPTION

Producerconsumerproblemisalsoknownasboundedbufferproblem.Inthisproblemwehavetwo processes, producerand consumer, who share a fixed size buffer. Producerworkisto produce data oritems and put in buffer. Consumer work is to remove data frombuffer and consume it. We have tomakesurethatproducerdonotproducedatawhenbufferisfullandconsumerdonotremovedata when buffer is empty.

The producer should go to sleep when buffer is full. Next time when consumer removes data it notifies the producer and producer starts producing data again. The consumer should go to sleep when buffer is empty. Next time when producer add data it notifies the consumer and consumer starts consuming data. This solution can be achieved using semaphores.

PROGRAM

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

intmutex=1,full=0,empty=3,x=0;

int main()
{
        intn;
        void producer();
        voidconsumer();
        int wait(int);
        int signal(int);
        printf("\n1.Producer\n2.Consumer\n3.Exit");
        while(1)
        {
                printf("\nEnteryourchoice:");
                scanf("%d",&n);
                switch(n)
                {
                        case1:if((mutex==1)&&(empty!=0))
```

```
                                        producer();
                                    else
                                        printf("Bufferisfull!!");
                                    break;
                            case2:if((mutex==1)&&(full!=0))
                                        consumer();
                                    else
                                        printf("Bufferisempty!!");
                                    break;
                            case3:
                                    exit(0);
                                    break;
                    }
            }

            return0;
}

intwait(ints)
{
            return(--s);
}

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

voidproducer()
{
            mutex=wait(mutex);
            full=signal(full);
            empty=wait(empty);
            x++;
            printf("\nProducerproducestheitem%d",x);
            mutex=signal(mutex);
}

voidconsumer()
{
            mutex=wait(mutex);
            full=wait(full);
            empty=signal(empty);
            printf("\nConsumerconsumesitem%d",x);
            x--;
            mutex=signal(mutex);
}
```

Output

1. Producer
2. Consumer
3. Exit

Enter your choice:1

Producerproducestheitem1 Enter your choice:2

Consumerconsumesitem1
Enter your choice:2
Buffer is empty!!
Enteryourchoice:1

Producerproducestheitem1 Enter your choice:1

Producerproducestheitem2 Enter your choice:1

Producerproducestheitem3 Enter your choice:1
Buffer is full!!
Enteryourchoice:3

Experiment -4

Write a C program to simulate the concept of Dining-philosophers problem.

Algorithm

1. **nitialize Constants and Resources:**
   - Set the number of philosophers (`NUM_PHILOSOPHERS`) and forks (`forks`) to manage the resources.
   - Create a list of philosophers (`philosophers`) to store the philosopher threads.
2. **Define the Philosopher Class:**
   - Create a `Philosopher` class that inherits from `threading.Thread`.
   - In the `__init__` method, accept the philosopher's index and initialize it.
   - Define `run` method to simulate the philosopher's actions.
3. **Implement the `think` Method:**
   - Inside the `Philosopher` class, create a `think` method.
   - Print a message indicating that the philosopher is thinking.
   - Introduce a delay using `time.sleep()` to simulate thinking.
4. **Implement the `eat` Method:**
   - Create an `eat` method in the `Philosopher` class.
   - Acquire the forks using `forks` list and philosopher's index.
   - Print a message indicating that the philosopher is eating.
   - Introduce a delay to simulate eating time.
   - Release the acquired forks.
5. **Create Philosopher Threads:**
   - Use a loop to create instances of the `Philosopher` class for each philosopher index.
   - Append each philosopher to the `philosophers` list.
6. **Start Philosopher Threads:**
   - Iterate through the `philosophers` list and start each philosopher thread using the `start()` method.
7. **Wait for Threads to Finish:**
   - Iterate through the `philosophers` list and call the `join()` method for each philosopher thread.
   - This ensures that the main program waits for all philosopher threads to finish before proceeding.

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

#define NUM_PHILOSOPHERS 5
#define NUM_EATING_CYCLES 3

pthread_mutex_t forks[NUM_PHILOSOPHERS];
pthread_t philosophers[NUM_PHILOSOPHERS];

void *philosopher_action(void *philosopher_id) {
   int id = *(int *)philosopher_id;
   int left_fork = id;
   int right_fork = (id + 1) % NUM_PHILOSOPHERS;

   for (int cycle = 0; cycle < NUM_EATING_CYCLES; cycle++) {
      // Thinking
      printf("Philosopher %d is thinking...\n", id);
      usleep(rand() % 1000000);

      // Pick up forks
      pthread_mutex_lock(&forks[left_fork]);
      pthread_mutex_lock(&forks[right_fork]);

      // Eating
      printf("Philosopher %d is eating...\n", id);
      usleep(rand() % 1000000);

      // Put down forks
      pthread_mutex_unlock(&forks[right_fork]);
      pthread_mutex_unlock(&forks[left_fork]);
   }

   printf("Philosopher %d has finished eating.\n", id);
   pthread_exit(NULL);
}
```

```
int main() {
    srand(time(NULL));

    for (int i = 0; i < NUM_PHILOSOPHERS; i++) {
        pthread_mutex_init(&forks[i], NULL);
    }

    for (int i = 0; i < NUM_PHILOSOPHERS; i++) {
        int *id = malloc(sizeof(int));
        *id = i;
        pthread_create(&philosophers[i], NULL, philosopher_action, id);
    }

    for (int i = 0; i < NUM_PHILOSOPHERS; i++) {
        pthread_join(philosophers[i], NULL);
        pthread_mutex_destroy(&forks[i]);
    }

    return 0;
}
```

EXPERIMENT-5

WriteaCprogramto simulatethefollowing techniquesofmemorymanagement
a) Paging b) Segmentation

OBJECTIVE

WriteaCprogramto simulatepagingtechniqueofmemorymanagement.

DESCRIPTION

In computer operating systems, paging is one of the memory management schemes bywhich a computer stores and retrieves data from the secondarystorage for use in main memory. In the paging memory-management scheme,theoperatingsystemretrievesdatafromsecondarystoragein same-size blocks called pages. Paging is a memory-management scheme that permits the physical address space a process to be noncontiguous. The basic method for implementing paging involves breaking physical memory into fixed-sized blocks called frames and breaking logical memory into blocksofthesamesizecalledpages. Whenaprocess isto beexecuted, itspagesare loadedinto any available memory frames from their source.

AIM:Tosimulatepagingtechniqueofmemorymanagement.

PROGRAM

```
#include<stdio.h>

#include<conio.h>
main()

{

int ms,ps,nop,np,rempages,i, j,x, y,pa,offset;ints[10],fno[10][20]; clrscr();
printf("\nEnterthememorysize--");scanf("%d",&ms);
printf("\nEnter the page size -- "); scanf("%d",&ps);nop
= ms/ps;
printf("\nTheno.ofpagesavailable inmemoryare --%d",nop);
printf("\nEnter number of processes -- "); scanf("%d",&np);
rempages = nop;
for(i=1;i<=np;i++)

{
printf("\nEnterno.ofpagesrequiredforp[%d]--",i);scanf("%d",&s[i]);
```

```
if(s[i]>rempages)
{
```

```
printf("\nMemoryisFull"); break;

}
rempages = rempages - s[i];
printf("\nEnterpagetableforp[%d]-,i);
for(j=0;j<s[i];j++)

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

}
printf("\nEnterLogicalAddresstofindPhysicalAddress");
printf("\nEnter process no. and pagenumber and offset ");
scanf("%d %d %d",&x,&y, &offset);
if(x>np|| y>=s[i]||offset>=ps)
printf("\nInvalidProcessorPageNumberoroffset"); else

{

pa=fno[x][y]*ps+offset;
printf("\nThePhysicalAddress is --%d",pa);

}

getch();
}
```

INPUT
Enterthememorysize–1000 Enter
the page size --100
Theno.ofpagesavailable inmemoryare --10 Enter
number of processes --3
Enter no.ofpagesrequiredforp[1]-- 4
Enterpagetableforp[1]--- 8 6 9 5
Enter no.ofpages requiredforp[2]-- 5
Enterpagetableforp[2]--- 1 4 5 7 3
Enter no.ofpagesrequiredforp[3]-- 5
OUTPUT
MemoryisFull
EnterLogicalAddresstofindPhysical Address
Enterprocessno. andpagenumberandoffset--2 3 60
ThePhysicalAddressis -------- 760

b) OBJECTIVE:Toimplementthememorymanagementpolicy-**segmentation**

PROGRAM LOGIC:
1. Starttheprogram.
2. Getthenumberofsegments.
3. Getthebaseaddressandlengthforeachsegment.
4. Getthelogicaladdress.
5. Checkwhetherthesegmentnumberiswithinthelimit,ifnotdisplaytheerrormessage.
6. Checkwhetherthebytereferenceiswithinthelimit,ifnotdisplaytheerrormessage.
7. Calculatethephysicalmemoryand display it.
8. Stoptheprogram

SOURCE CODE:

```
#include<stdio.h>#i
nclude
<conio.h>#include<
math.h>int sost;
 voidgstinfo();
void  ptladdr();
struct segtab
{ int sno;
int baddr;
int limit;int
val[10];
}st[10];
voidgstinfo()
{inti,j;
printf("\n\tEnterthesizeofthesegmenttable:"); scanf("%d",&sost);
for(i=1;i<=sost;i++)
 {
 printf("\n\tEntertheinformationaboutsegment:%d",i);
 st[i].sno = i;
printf("\n\tEnterthebaseAddress:");
scanf("%d",&st[i].baddr);
printf("\n\tEnter the Limit: ");
scanf("%d",&st[i].limit);
for(j=0;j<=sost;i++)
printf("\t\t%d\t\t%d\t\t%d\n\n",st[i].sno,st[i].baddr,st[i].limit);
 printf("\n\nEnter the logical Address: ");
scanf("%d",&swd);
n=swd;
 while(n!=0)
{
 n=n/10;d++;
```

```
 }
 s=swd/pow(10,d-1);
 disp=swd%(int)pow(10,d-1);
 if(s<=sost)
{
if(disp<st[s].limit)
{
 paddr = st[s].baddr + disp;
printf("\n\t\tLogicalAddressis:%d",swd);
printf("\n\t\tMappedPhysicaladdressis:%d",paddr); printf("\n\tThe
value is: %d",( st[s].val[disp] ) );
 }
 Else
 printf("\n\t\tLimit ofsegment %dis high\n\n",s);
}
else
 printf("\n\t\tInvalidSegmentAddress\n");
 }
 voidmain()
 {charch;
clrscr();
gstinfo();
do
{
 ptladdr();
printf("\n\tDoUwanttoContinue(Y/N)");
flushall();
scanf("%c",&ch);
}while(ch== 'Y'||ch=='y');
getch();
 }
```

INPUTANDOUTPUT:

```
Enter the size of the segment table: 3
Enterthe information about segment: 1
Enter the base Address: 4
EntertheLimit: 5
Enterthe4 addressValue: 11
Enterthe5 addressValue: 12
Enterthe6 addressValue: 13
Enterthe7 addressValue: 14
Enterthe8 addressValue: 15
Entertheinformationaboutsegment:2
 Enter the base Address: 5
EntertheLimit: 4
Enterthe5 addressValue: 21
Enterthe6 addressValue: 31
Enterthe7 addressValue: 41
```

Enterthe8 addressValue: 51
Entertheinformationaboutsegment:3
Enter the base Address: 3
EntertheLimit: 4
Enterthe3 addressValue: 31
Enterthe4addressValue: 41
Enterthe5 addressValue: 41
Enterthe6 addressValue: 51
SEGMENTTABLESEG.NOBASEADDRESSLIMIT
145
254
334
EnterthelogicalAddress:3
Logical Address is: 3
MappedPhysicaladdressis:3 The
value is: 31
DoUwant toContinue(Y/N)
SEGMENTTABLESEG.NOBASEADDRESSLIMIT
145
254
334
EnterthelogicalAddress:1
Logical Address is: 1
MappedPhysicaladdress is:4
Thevalueis:11DoUwanttoContinue(Y/N)

VIVA QUESTIONS:

1. The pagetable contains _____
2. What iscompaction?
3. Operating Systemmaintainsthe page tablefor_____
4. Physicalmemoryis broken into fixed-sized blockscalled _____
5. Logical memoryis broken into blocksofthe same size called _____
6. Everyaddressgenerated bytheCPUisdividedintotwoparts:
7. The _____isusedasanindexintothepagetable.
8. The _____tablecontainsthebase addressofeachpageinphysicalmemory.
9. Thesizeofapageistypically :
10. Withpagingthereisno _____ fragmentation.
11. The operating system maintains a_____ tablethat keepstrackofhowmanyframeshavebeen allocated, how many are there, and how many are available.
12. Paging increases the _____time.
13. Smaller pagetables areimplementedasasetof_____
14. Thepagetableregistersshould bebuiltwith_____
15. For larger page tables, they are kept in main memory and a _____pointstothepage table.
16. For everyprocess there is a _____table.
17. _____addressgeneratedaftercompiletime.
18. _____addressgeneratedafterruntime.
19. Addressbinding isdone in_____phases.
20. Definesegmentation.

EXPERIMENT-6

WriteaCprogramto illustratethefollowing IPCmechanisms

a) Pipes b) FIFOs c)MessageQueues d)Shared Memory

OBJECTIVE:

Write a CprogramtoillustratethePipesIPCmechanism

PROGRAM

```
#include <stdio.h>
#include <unistd.h>
#defineMSGSIZE16
char*msg1="hello,world#1";
char*msg2="hello,world#2";
char*msg3="hello,world#3";

int main()
{
   charinbuf[MSGSIZE];
   int p[2], i;

   if(pipe(p)<0)
      exit(1);

   /*continued*/
   /*writepipe*/

   write(p[1],msg1,MSGSIZE);
   write(p[1],msg2,MSGSIZE);
   write(p[1],msg3,MSGSIZE);

   for(i= 0;i< 3;i++){
      /*readpipe*/
      read(p[0],inbuf,MSGSIZE);
      printf("% s\n", inbuf);
   }
   return0;
}
```

Output:

```
hello,world#1
hello,world#2
hello,world#3
```

OBJECTIVE

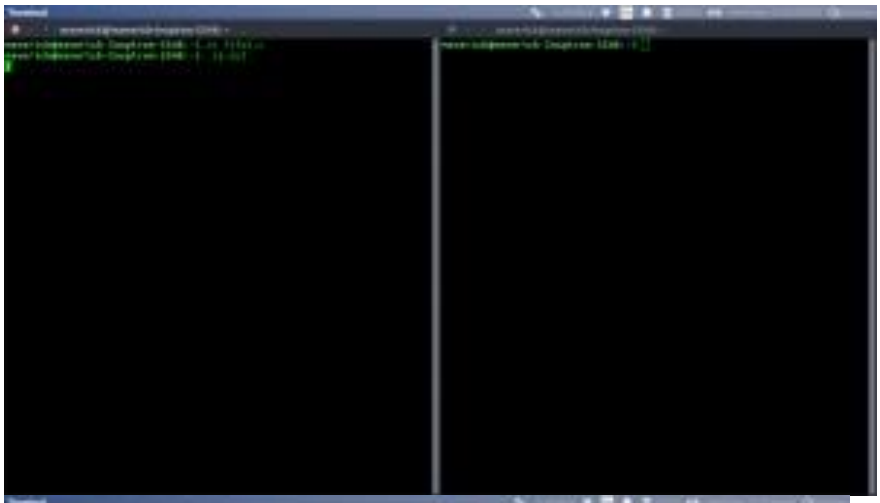
b) Write a CprogramtoillustratetheFIFOIPC mechanism

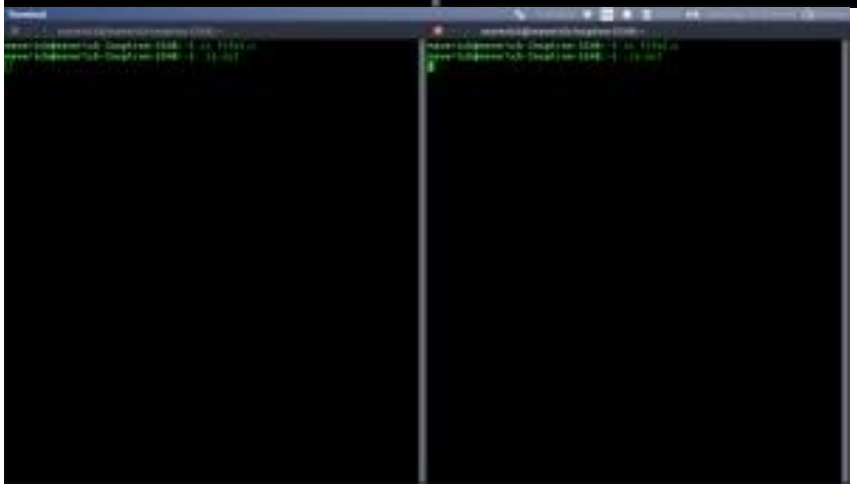
PROGRAM

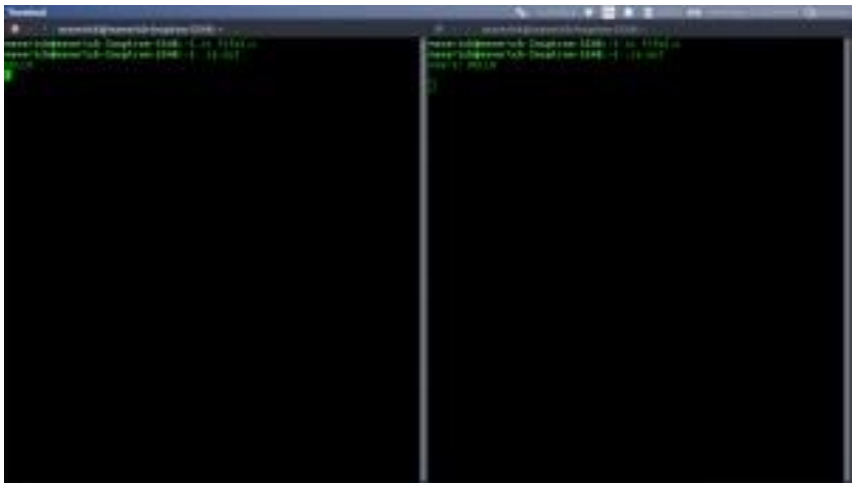
```
#include <stdio.h>
#include <string.h>
#include <fcntl.h>
#include <sys/stat.h>
#include<sys/types.h>
#include <unistd.h>

int main()
{
   intfd;

   //FIFO filepath
   char*myfifo="/tmp/myfifo";

   //Creatingthenamedfile(FIFO)
   //mkfifo(<pathname>,<permission>)
   mkfifo(myfifo, 0666);

   chararr1[80],arr2[80];
   while (1)
   {
      //OpenFIFOforwriteonly
      fd=open(myfifo,O_WRONLY);

      //Takeaninputarr2ingfromuser.
      //80ismaximumlength fgets(arr2,
      80, stdin);

      //Writetheinputarr2ingonFIFO
      //andcloseit
      write(fd,arr2,strlen(arr2)+1); close(fd);

      //OpenFIFOfor Readonly
      fd=open(myfifo,O_RDONLY);

      // Read from
      FIFOread(fd,arr1,sizeof(ar
      r1));

      // Print the read message
      printf("User2:%s\n",arr1);
```

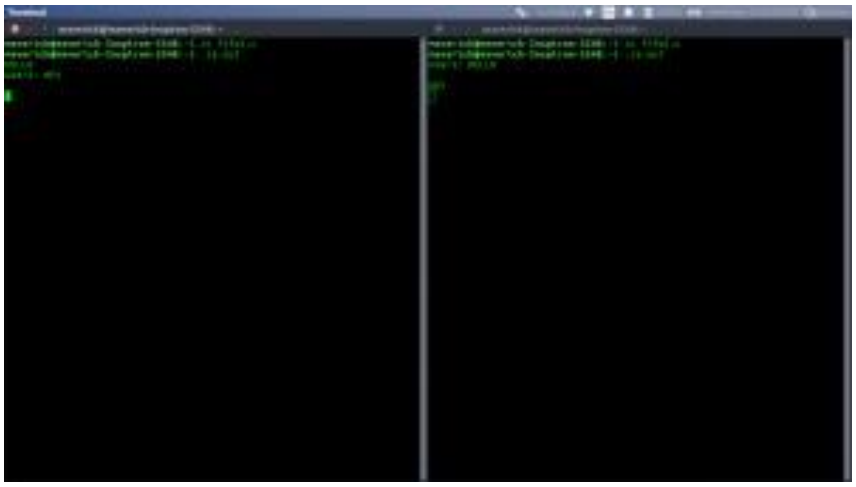
```
        close(fd);
    }
    return0;
}
```

Output:









OBJECTIVE

c) WriteaCprogramtoillustratetheMessage Queue IPCmechanism

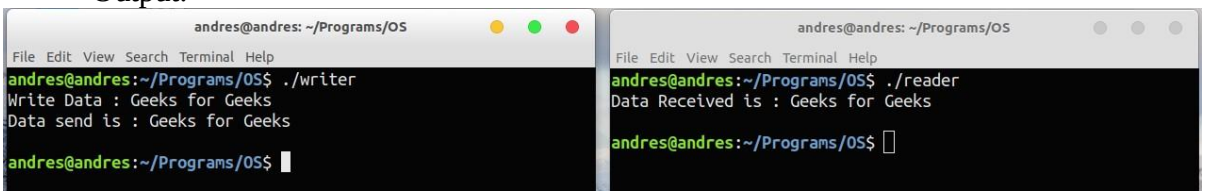
PROGRAM:

```
//CProgramforMessageQueue(WriterProcess)
#include <stdio.h>
#include <sys/ipc.h>
#include<sys/msg.h>

//structureformessagequeue
struct mesg_buffer {
    long
    mesg_type;charmesg
    _text[100];
} message;

int main()
{
    key_tkey;
    intmsgid;

    //ftoktogenerateuniquekey
    key = ftok("progfile", 65);

    //msggetcreatesamessage queue
    //andreturnsidentifier
    msgid=msgget(key,0666|IPC_CREAT);
    message.mesg_type = 1;

    printf("Write Data : ");
    gets(message.mesg_text);

    //msgsndtosendmessage
    msgsnd(msgid,&message,sizeof(message),0);

    //displaythemessage
    printf("Datasend is:%s\n",message.mesg_text);

    return0;
}


//CProgramforMessageQueue(ReaderProcess) #include
<stdio.h>
#include <sys/ipc.h>
```

```
#include<sys/msg.h>
```

```
//structureformessagequeue
struct mesg_buffer {
    long
    mesg_type;charmesg
    _text[100];
} message;

int main()
{
    key_tkey;
    intmsgid;

    //ftoktogenerateuniquekey
    key = ftok("progfile", 65);

    //msggetcreatesamessage queue
    //andreturnsidentifier
    msgid=msgget(key,0666|IPC_CREAT);

    //msgrcvtoreceivemessage
    msgrcv(msgid,&message,sizeof(message),1,0);

    // display the message
    printf("DataReceivedis:%s\n",
                message.mesg_text);

    // to destroy the message queue
    msgctl(msgid,IPC_RMID,NULL);

    return0;
}
```

Output:



OBJECTIVE

d) Writea CprogramtoillustratetheSharedMemoryIPCmechanism

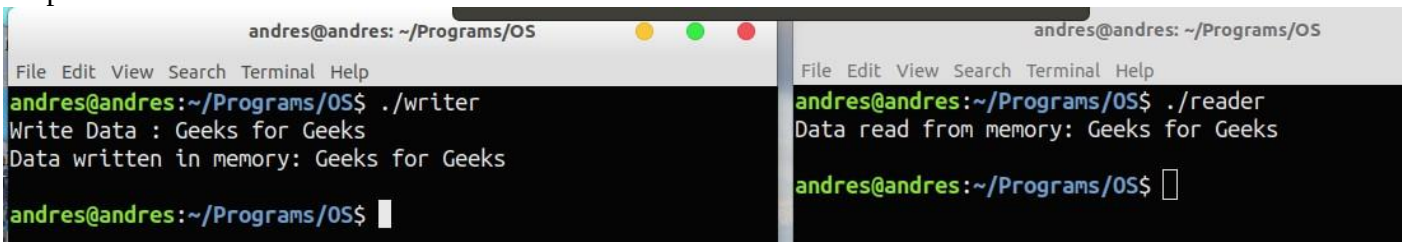
PROGRAM:

```
#include <iostream>
#include <sys/ipc.h>
#include<sys/shm.h>
#include <stdio.h>
using namespace std;

int main()
{
   // ftok to generate unique key
   key_tkey=ftok("shmfile",65);

   //shmgetreturnsanidentifierinshmid
   intshmid=shmget(key,1024,0666|IPC_CREAT);

   //shmattoattachtosharedmemory
   char *str=(char*)shmat(shmid,(void*)0,0);

   cout<<"WriteData:";
   gets(str);

   printf("Datawritteninmemory:%s\n",str);

   //detachfromsharedmemory
   shmdt(str);

   return0;
}
```

SHAREDMEMORY FORREADER PROCESS

```
#include <iostream>
#include <sys/ipc.h>
#include<sys/shm.h>
#include <stdio.h>
using namespace std;

int main()
{
    // ftok to generate unique key
    key_tkey=ftok("shmfile",65);

    //shmgetreturnsanidentifierinshmid
    intshmid=shmget(key,1024,0666|IPC_CREAT);

    //shmattoattachtosharedmemory
    char*str=(char*)shmat(shmid,(void*)0,0);

    printf("Data read from memory: %s\n",str);

    //detachfromsharedmemory
    shmdt(str);

    // destroy the shared memory
    shmctl(shmid,IPC_RMID,NULL);

    return0;
}
```

Output:



VIVA QUESTIONS

1. Whatarelocaland globalpagereplacements?
2. Define latency,transferandseektimewithrespecttodiskI/O.
3. DescribetheBuddysystemofmemoryallocation.
4. Whatistime-stamping?
5. Howarethewait/signaloperationsfor monitor differentfromthosefor semaphores?
6. Inthecontextofmemorymanagement, whatareplacementandreplacementalgorithms?
7. Inloadingprogramsintomemory,whatisthedifferencebetweenload-timedynamiclinking and run-time dynamic linking?
8. Whataredemand-pagingandpre-paging?
9. Pagingamemorymanagementfunction,whilemultiprogrammingaprocessormanagement function, are the two interdependent?
10. Whatispagecannibalizing?
11. Whathastriggeredtheneed formultitasking inPCs?
12. What arethefourlayersthatWindowsNThaveinordertoachieveindependence?
13. Explaincompaction.
14. Whatarepageframes?
15. What arepages?
16. Differentiatebetweenlogicalandphysicaladdress.
17. Whendoespagefaulterroroccur?
18. Explainthrashing.
19. Whatisthecriteria forthebest pagereplacementalgorithm?
20. WhatisBelady'sanomaly?

Experiment 7a.

Write a C Program to simulate worst fit contiguous memory allocation Techniques

Algorithm

1. **Define Structures and Constants:**
   - Define a structure named `MemoryBlock` with fields to represent memory blocks: `id`, `size`, and `allocated`.
2. **Input Memory Block Information:**
   - Prompt the user to enter the number of memory blocks (`numBlocks`).
   - Initialize an array of `MemoryBlock` structures named `blocks`.
3. **Initialize Memory Blocks:**
   - Use a loop to initialize each memory block by assigning an ID and prompting the user for the size of each block.
   - Set the `allocated` field to 0 to indicate that the block is not allocated.
4. **Input Process Information:**
   - Prompt the user to enter the number of processes (`numProcesses`).
5. **Allocate Memory to Processes (Worst Fit):**
   - For each process, prompt the user to enter the size of the process (`processSize`).
   - Iterate through the `blocks` array to find the largest available memory block that can accommodate the process.
   - If a suitable block is found, mark it as `allocated`, and print the allocation details.
   - If no suitable block is found, print a message indicating that memory allocation failed for the process.
6. **Main Function:**
   - Inside the `main` function, call the steps above in sequence.

```
#include <stdio.h>

#define MAX_BLOCKS 10

// Structure to represent a memory block
struct MemoryBlock {
   int id;
   int size;
   int allocated;
};
```

```
// Function to allocate memory using worst fit technique
void allocateMemory(struct MemoryBlock blocks[], int numBlocks, int processSize) {
    int largestBlockIndex = -1;

    // Find the largest available block that can accommodate the process
    for (int i = 0; i < numBlocks; i++) {
        if (!blocks[i].allocated && blocks[i].size >= processSize) {
            if (largestBlockIndex == -1 || blocks[i].size > blocks[largestBlockIndex].size) {
                largestBlockIndex = i;
            }
        }
    }

    if (largestBlockIndex != -1) {
        // Allocate memory to the process
        blocks[largestBlockIndex].allocated = 1;
        printf("Allocated Memory Block %d to Process of Size %d\n",
blocks[largestBlockIndex].id, processSize);
    } else {
        printf("Unable to Allocate Memory for Process of Size %d\n", processSize);
    }
}

int main() {
    struct MemoryBlock blocks[MAX_BLOCKS];
    int numBlocks;

    printf("Enter the number of memory blocks: ");
    scanf("%d", &numBlocks);

    // Initialize memory blocks
    for (int i = 0; i < numBlocks; i++) {
        blocks[i].id = i + 1;
        printf("Enter size of Memory Block %d: ", blocks[i].id);
        scanf("%d", &blocks[i].size);
        blocks[i].allocated = 0;
    }

    int numProcesses;
    printf("Enter the number of processes: ");
```

```
    scanf("%d", &numProcesses);

    // Allocate memory to processes
    for (int i = 0; i < numProcesses; i++) {
        int processSize;
        printf("Enter size of Process %d: ", i + 1);
        scanf("%d", &processSize);
        allocateMemory(blocks, numBlocks, processSize);
    }

    return 0;
}
```

Experiment 7b.

Write a C Program to simulate best fit contiguous memory allocation Techniques

Algorithm

1. **Define Structures and Constants:**
   - Define a structure named `MemoryBlock` with fields to represent memory blocks: `id`, `size`, `allocated`, and optionally, a field to track the remaining size after allocation.
2. **Input Memory Block Information:**
   - Prompt the user to enter the number of memory blocks (`numBlocks`).
   - Initialize an array of `MemoryBlock` structures named `blocks`.
3. **Initialize Memory Blocks:**
   - Use a loop to initialize each memory block by assigning an ID and prompting the user for the size of each block.
   - Set the `allocated` field to 0 to indicate that the block is not allocated.
4. **Input Process Information:**
   - Prompt the user to enter the number of processes (`numProcesses`).
5. **Allocate Memory to Processes (Best Fit):**
   - For each process, prompt the user to enter the size of the process (`processSize`).
   - Iterate through the `blocks` array to find the smallest available memory block that can accommodate the process.
   - If a suitable block is found, mark it as `allocated`, update its remaining size, and print the allocation details.
   - If no suitable block is found, print a message indicating that memory allocation failed for the process.
6. **Main Function:**
   - Inside the `main` function, call the steps above in sequence.

```
#include <stdio.h>

#define MAX_BLOCKS 10

// Structure to represent a memory block
struct MemoryBlock {
    int id;
    int size;
    int allocated;
```

```
};

// Function to allocate memory using best fit technique
void allocateMemory(struct MemoryBlock blocks[], int numBlocks, int processSize) {
   int bestFitIndex = -1;
   int smallestSize = -1;

   // Find the smallest available block that can accommodate the process
   for (int i = 0; i < numBlocks; i++) {
      if (!blocks[i].allocated && blocks[i].size >= processSize) {
         if (bestFitIndex == -1 || blocks[i].size < smallestSize) {
            bestFitIndex = i;
            smallestSize = blocks[i].size;
         }
      }
   }

   if (bestFitIndex != -1) {
      // Allocate memory to the process
      blocks[bestFitIndex].allocated = 1;
      blocks[bestFitIndex].size -= processSize;
      printf("Allocated Memory Block %d to Process of Size %d\n",
blocks[bestFitIndex].id, processSize);
   } else {
      printf("Unable to Allocate Memory for Process of Size %d\n", processSize);
   }
}

int main() {
   struct MemoryBlock blocks[MAX_BLOCKS];
   int numBlocks;

   printf("Enter the number of memory blocks: ");
   scanf("%d", &numBlocks);

   // Initialize memory blocks
   for (int i = 0; i < numBlocks; i++) {
      blocks[i].id = i + 1;
      printf("Enter size of Memory Block %d
```

Experiment 7c.

Write a C Program to simulate First fit contiguous memory allocation Techniques

Algorithm

1. **Define Structures and Constants:**
   - Define a structure named `MemoryBlock` with fields to represent memory blocks: `id`, `size`, and `allocated`.
2. **Input Memory Block Information:**
   - Prompt the user to enter the number of memory blocks (`numBlocks`).
   - Initialize an array of `MemoryBlock` structures named `blocks`.
3. **Initialize Memory Blocks:**
   - Use a loop to initialize each memory block by assigning an ID and prompting the user for the size of each block.
   - Set the `allocated` field to 0 to indicate that the block is not allocated.
4. **Input Process Information:**
   - Prompt the user to enter the number of processes (`numProcesses`).
5. **Allocate Memory to Processes (First Fit):**
   - For each process, prompt the user to enter the size of the process (`processSize`).
   - Iterate through the `blocks` array to find the first available memory block that can accommodate the process.
   - If a suitable block is found, mark it as `allocated`, update its remaining size, and print the allocation details.
   - If no suitable block is found, print a message indicating that memory allocation failed for the process.
6. **Main Function:**
   - Inside the `main` function, call the steps above in sequence.

Here's the summarized step-by-step procedure:

1. Define structures and constants.
2. Input the number of memory blocks (`numBlocks`).
3. Initialize an array of `MemoryBlock` structures and input memory block sizes.
4. Input the number of processes (`numProcesses`).
5. For each process:
   - Input the size of the process (`processSize`).
   - Find the first available memory block that can accommodate the process (first fit).
   - If found, mark the block as allocated, update its remaining size, and print allocation details.
6. End the program

```
// first fit
#include <stdio.h>

#define MAX_BLOCKS 10

// Structure to represent a memory block
struct MemoryBlock {
   int id;
   int size;
   int allocated;
};

// Function to allocate memory using first fit technique
void allocateMemory(struct MemoryBlock blocks[], int numBlocks, int processSize) {
   int firstFitIndex = -1;

   // Find the first available block that can accommodate the process
   for (int i = 0; i < numBlocks; i++) {
      if (!blocks[i].allocated && blocks[i].size >= processSize) {
         firstFitIndex = i;
         break;
      }
   }

   if (firstFitIndex != -1) {
      // Allocate memory to the process
      blocks[firstFitIndex].allocated = 1;
      blocks[firstFitIndex].size -= processSize;
      printf("Allocated Memory Block %d to Process of Size %d\n",
blocks[firstFitIndex].id, processSize);
   } else {
      printf("Unable to Allocate Memory for Process of Size %d\n", processSize);
   }
}

int main() {
   struct MemoryBlock blocks[MAX_BLOCKS];
   int numBlocks;

   printf("Enter the number of memory blocks: ");
```

```
    scanf("%d", &numBlocks);

    // Initialize memory blocks
    for (int i = 0; i < numBlocks; i++) {
        blocks[i].id = i + 1;
        printf("Enter size of Memory Block %d: ", blocks[i].id);
        scanf("%d", &blocks[i].size);
        blocks[i].allocated = 0;
    }

    int numProcesses;
    printf("Enter the number of processes: ");
    scanf("%d", &numProcesses);

    // Allocate memory to processes
    for (int i = 0; i < numProcesses; i++) {
        int processSize;
        printf("Enter size of Process %d: ", i + 1);
        scanf("%d", &processSize);
        allocateMemory(blocks, numBlocks, processSize);
    }

    return 0;
}
```

Experiment 8a.

Write a File Organization Technique C program to simulate Single level directory

Algorithm

1. **Define Structures and Constants:**
   - Define a structure named `File` with fields to represent a file: `filename` and `size`.
   - Define a structure named `Directory` with an array of `File` structures and an `int` to track the number of files (`fileCount`).
   - Define constants such as `MAX_FILES` and `MAX_FILENAME_LENGTH` for the maximum number of files and filename length.
2. **Create File Function:**
   - Define a function named `createFile` that accepts pointers to a `Directory`, a `filename`, and a `size`.
   - Check if the directory is full. If so, print a message indicating that no more files can be created.
   - Check if a file with the same name already exists. If so, print an error message.
   - Otherwise, add the file information to the `Directory`'s `files` array and increment the `fileCount`.
3. **Delete File Function:**
   - Define a function named `deleteFile` that accepts pointers to a `Directory` and a `filename` to be deleted.
   - Search for the file in the `files` array by comparing filenames.
   - If the file is found, shift files to close the gap and decrement the `fileCount`.
   - If the file is not found, print an error message.
4. **List Files Function:**
   - Define a function named `listFiles` that accepts a pointer to a `Directory`.
   - Check if there are no files in the directory. If so, print a message indicating there are no files.
   - Otherwise, iterate through the `files` array and print the filenames and sizes.
5. **Main Function:**
   - Inside the `main` function, create a `Directory` structure.
   - Use a loop to display a menu to the user with options for creating, deleting, listing files, and exiting.
   - Depending on the user's choice, call the corresponding functions.

Here's the summarized step-by-step procedure:

1. Define structures and constants.
2. Define a function to create a file.
3. Define a function to delete a file.

4. Define a function to list files.
5. Inside the `main` function:
    - Create a `Directory` structure.
    - Display a menu to the user.
    - Based on the user's choice, call the appropriate functions.

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

#define MAX_FILES 100
#define MAX_FILENAME_LENGTH 50

struct File {
   char filename[MAX_FILENAME_LENGTH];
   int size;
};

struct Directory {
   struct File files[MAX_FILES];
   int fileCount;
};

void createFile(struct Directory *directory, char *filename, int size) {
   if (directory->fileCount >= MAX_FILES) {
      printf("Directory is full. Cannot create more files.\n");
      return;
   }

   for (int i = 0; i < directory->fileCount; i++) {
      if (strcmp(directory->files[i].filename, filename) == 0) {
         printf("File with the same name already exists.\n");
         return;
      }
   }

   strcpy(directory->files[directory->fileCount].filename, filename);
```

```
    directory->files[directory->fileCount].size = size;
    directory->fileCount++;

    printf("File '%s' created successfully.\n", filename);
}

void deleteFile(struct Directory *directory, char *filename) {
    for (int i = 0; i < directory->fileCount; i++) {
        if (strcmp(directory->files[i].filename, filename) == 0) {
            // Shift files to fill the gap
            for (int j = i; j < directory->fileCount - 1; j++) {
                directory->files[j] = directory->files[j + 1];
            }
            directory->fileCount--;

            printf("File '%s' deleted successfully.\n", filename);
            return;
        }
    }

    printf("File '%s' not found.\n", filename);
}

void listFiles(struct Directory *directory) {
    if (directory->fileCount == 0) {
        printf("No files in the directory.\n");
        return;
    }

    printf("List of files:\n");
    for (int i = 0; i < directory->fileCount; i++) {
        printf("File: %s, Size: %d bytes\n", directory->files[i].filename, directory-
>files[i].size);
    }
}

int main() {
    struct Directory directory;
    directory.fileCount = 0;
```

```
while (1) {
    printf("\nMenu:\n");
    printf("1. Create File\n");
    printf("2. Delete File\n");
    printf("3. List Files\n");
    printf("4. Exit\n");
    printf("Enter your choice: ");

    int choice;
    scanf("%d", &choice);

    switch (choice) {
        case 1:
            char filename[MAX_FILENAME_LENGTH];
            int size;
            printf("Enter filename: ");
            scanf("%s", filename);
            printf("Enter file size: ");
            scanf("%d", &size);
            createFile(&directory, filename, size);
            break;

        case 2:
            printf("Enter filename to delete: ");
            scanf("%s", filename);
            deleteFile(&directory, filename);
            break;

        case 3:
            listFiles(&directory);
            break;

        case 4:
            printf("Exiting program.\n");
            exit(0);

        default:
            printf("Invalid choice. Please enter a valid option.\n");
            break;
    }
```

```
    }

    return 0;
}
```

Experiment 8b.

Write a File Organization Technique C program to simulate Two level directory

Algorithm

Step 1: Initialize the Root Directory

1. Create a structure to represent a file. Each file structure should contain a name field.
2. Create a structure to represent a directory. Each directory structure should contain a name field, an array of file structures (representing files contained in the directory), and a counter to keep track of the number of files in the directory.
3. Initialize the root directory structure. Set its name to "root" and the number of files to 0.

Step 2: Creating Files

1. Display a menu to the user with options to perform various actions.
2. If the user selects the option to create a file:
    - Prompt the user to enter the name of the file.
    - Check if the number of files in the current directory is less than the maximum allowed.
        - If yes, add the new file's name to the array of files in the current directory and increment the file counter.
        - If no, display an error message indicating that the directory is full.

Step 3: Displaying Directory Contents

1. If the user selects the option to display directory contents:
    - Iterate through the array of files in the current directory.
    - Display the names of all files in the directory.

Step 4: Creating Subdirectories

1. If the user selects the option to create a subdirectory:

- Prompt the user to enter the name of the subdirectory.
- Check if the number of subdirectories is less than the maximum allowed.
    - If yes, create a new directory structure with the provided name and initialize its file counter to 0.
    - If no, display an error message indicating that the maximum number of subdirectories has been reached.

Step 5: Exiting the Program

1. If the user selects the option to exit the program:
    - Display a farewell message.
    - Terminate the program.

Step 6: Repeat

1. After each operation, return to the main menu and allow the user to perform more actions until they choose to exit.

Step 7: Compile and Run

1. Compile the program using a C compiler.
2. Run the compiled executable.
3. Interact with the program by choosing options from the menu to create files, display directory contents, create subdirectories, and exit the program

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

#define MAX_DIRS 100
#define MAX_FILES 100
#define MAX_NAME_LENGTH 50

typedef struct {
   char name[MAX_NAME_LENGTH];
} File;

typedef struct {
   char name[MAX_NAME_LENGTH];
   File files[MAX_FILES];
```

```
    int num_files;
} Directory;

Directory root_directory;
Directory subdirectories[MAX_DIRS];
int num_subdirectories = 0;

void createFile(Directory *dir, char *name) {
    if (dir->num_files < MAX_FILES) {
        strcpy(dir->files[dir->num_files].name, name);
        dir->num_files++;
        printf("File '%s' created in directory '%s'.\n", name, dir->name);
    } else {
        printf("Directory '%s' is full.\n", dir->name);
    }
}

void displayDirectory(Directory *dir) {
    printf("Contents of directory '%s':\n", dir->name);
    for (int i = 0; i < dir->num_files; i++) {
        printf("- %s\n", dir->files[i].name);
    }
}

void createSubdirectory(char *name) {
    if (num_subdirectories < MAX_DIRS) {
        strcpy(subdirectories[num_subdirectories].name, name);
        subdirectories[num_subdirectories].num_files = 0;
        num_subdirectories++;
        printf("Subdirectory '%s' created.\n", name);
    } else {
        printf("Maximum number of subdirectories reached.\n");
    }
}

int main() {
    strcpy(root_directory.name, "root");
    root_directory.num_files = 0;

    int choice;
```

```
    char name[MAX_NAME_LENGTH];

    do {
        printf("\nTwo-Level Directory Simulation Menu:\n");
        printf("1. Create file\n");
        printf("2. Display directory contents\n");
        printf("3. Create subdirectory\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter file name: ");
                scanf("%s", name);
                createFile(&root_directory, name);
                break;
            case 2:
                displayDirectory(&root_directory);
                break;
            case 3:
                printf("Enter subdirectory name: ");
                scanf("%s", name);
                createSubdirectory(name);
                break;
            case 4:
                printf("Exiting program.\n");
                break;
            default:
                printf("Invalid choice. Please enter a valid option.\n");
        }
    } while (choice != 4);

    return 0;
}
```

Experiment 9a.

Write a C Program for FIFO Page Replacement Algorithm

FIFO Page Replacement Algorithm: Step-by-Step Procedure

Step 1: Initialize

1. Initialize an empty queue to represent the page frames in memory.
2. Initialize a variable to keep track of the current position in the queue (the position where the next page will be placed).

Step 2: Page Request

1. When a page request is received:
    - Check if the requested page is already in a page frame.
        - If yes, it's a page hit. No need to make any changes. Continue to the next page request.
        - If no, it's a page fault (miss):
            - Check if there is an empty page frame available.
                - If yes, place the requested page in the empty page frame and enqueue it in the queue. Update the current position in the queue.
                - If no, a page replacement is required:
                    - Dequeue the page at the front of the queue (the oldest page) to make space.
                    - Enqueue the requested page in the queue.
                    - Update the current position in the queue.

Step 3: Repeat

1. Continue processing page requests one by one, following the steps outlined in Step 2.
2. Keep track of the number of page faults that occur during this process.

Step 4: End

1. Once all page requests have been processed, calculate and report the total number of page faults.

Note: The FIFO algorithm is relatively simple and straightforward to implement. However, it suffers from the "Belady's Anomaly," where increasing the number of page frames can lead to an increase in the number of page faults. This is because the oldest pages are replaced, regardless of how frequently they are accessed.

Example: Let's consider a simple example with a page frame size of 3 and the following page request sequence: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5.

Initial state: Queue is empty. Page request 1: Queue: [1], Page Fault. Page request 2: Queue: [1, 2], Page Fault. Page request 3: Queue: [1, 2, 3], Page Fault. Page request 4: Queue: [2, 3, 4], Page Fault. Page request 1: Queue: [2, 3, 4], Page Hit. Page request 2: Queue: [2, 3, 4], Page Hit. Page request 5: Queue: [3, 4, 5], Page Fault (Replace 2). ...and so on.

Total page faults: 7.

```
#include <stdio.h>
int main()
{
int referenceString[50], pageFaults = 0, m, n, s, pages, frames;
printf("\nEnter the number of Pages:\t");
scanf("%d", &pages);
printf("\nEnter reference string values:\n");
for( m = 0; m < pages; m++)
{
printf("Value No. [%d]:\t", m + 1);
scanf("%d", &referenceString[m]);
}
printf("\n What are the total number of frames:\t");
{
scanf("%d", &frames);
}
int temp[frames];
for(m = 0; m < frames; m++)
{
  temp[m] = -1;
}
for(m = 0; m < pages; m++)
{
  s = 0;
for(n = 0; n < frames; n++)
   {
     if(referenceString[m] == temp[n])
       {
```

```
        s++;
pageFaults--;
      }
  }
pageFaults++;
if((pageFaults<= frames) && (s == 0))
    {
      temp[m] = referenceString[m];
    }
  else if(s == 0)
    {
temp[(pageFaults - 1) % frames] = referenceString[m];
    }
printf("\n");
for(n = 0; n < frames; n++)
     {
printf("%d\t", temp[n]);
     }
}
printf("\nTotal Page Faults:\t%d\n", pageFaults);
return 0;
}
```

Experiment 9b.

Write a C Program for LRU Page Replacement Algorithm

LRU Page Replacement Algorithm: Step-by-Step Procedure

Step 1: Initialize

1. Initialize a data structure (such as a queue or linked list) to maintain the order of page frames in memory.

Step 2: Page Request

1. When a page request is received:

- Check if the requested page is already in a page frame.
  - If yes, it's a page hit. Update the position of the page in the data structure to indicate that it was recently used. No need to make any changes. Continue to the next page request.
  - If no, it's a page fault (miss):
    - Check if there is an empty page frame available.
      - If yes, place the requested page in the empty page frame and insert it into the data structure to represent its recent use.
      - If no, a page replacement is required:
        - Find the page that was least recently used in the data structure (typically at the front of the queue or the beginning of the linked list).
        - Remove the least recently used page from the page frame and the data structure.
        - Place the requested page in the freed page frame and insert it into the data structure to represent its recent use.

Step 3: Repeat

1. Continue processing page requests one by one, following the steps outlined in Step 2.
2. Keep track of the number of page faults that occur during this process.

Step 4: End

1. Once all page requests have been processed, calculate and report the total number of page faults.

Note: The LRU algorithm aims to minimize page faults by replacing the page that has not been used for the longest time. However, implementing the LRU algorithm can be challenging in practice, as maintaining an accurate record of the order of page usage can be resource-intensive. Various data structures and strategies can be used to implement the LRU algorithm efficiently.

Example: Let's consider a simple example with a page frame size of 3 and the following page request sequence: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5.

Initial state: Queue is empty. Page request 1: Queue: [1], Page Fault. Page request 2: Queue: [1, 2], Page Fault. Page request 3: Queue: [1, 2, 3], Page Fault. Page request 4: Queue: [2, 3, 4], Page Fault. Page request 1: Queue: [3, 4, 1], Page Fault (Replace 2, as it's the least recently used). Page request 2: Queue: [4, 1, 2], Page Fault (Replace 3). Page request 5: Queue: [1, 2, 5], Page Fault (Replace 4). ...and so on.

Total page faults: 7.

```
#include<stdio.h>

int findLRU(int time[], int n){
int i, minimum = time[0], pos = 0;

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

int main()
{
   int no_of_frames, no_of_pages, frames[10], pages[30], counter = 0,
time[10], flag1, flag2, i, j, pos, faults = 0;
printf("Enter number of frames: ");
scanf("%d" , &no_of_frames);
printf("Enter number of pages: ");
scanf("%d", &no_of_pages);
printf("Enter reference string: ");
for(i = 0; i<no_of_pages; ++i){
scanf("%d", &pages[i]);
   }

for(i = 0; i<no_of_frames; ++i){
    frames[i] = -1;
   }

for(i = 0; i<no_of_pages; ++i){
    flag1 = flag2 = 0;

for(j = 0; j <no_of_frames; ++j){
```

```
    if(frames[j] == pages[i]){
    counter++;
    time[j] = counter;
  flag1 = flag2 = 1;
  break;
  }
    }

if(flag1 == 0){
for(j = 0; j <no_of_frames; ++j){
    if(frames[j] == -1){
    counter++;
    faults++;
    frames[j] = pages[i];
    time[j] = counter;
    flag2 = 1;
    break;
    }
    }
    }

if(flag2 == 0){
pos = findLRU(time, no_of_frames);
    counter++;
    faults++;
    frames[pos] = pages[i];
    time[pos] = counter;
    }

printf("\n");

for(j = 0; j <no_of_frames; ++j){
printf("%d\t", frames[j]);
    }
}
```

```
printf("\n\nTotal Page Faults = %d", faults);

   return 0;
}
```

Experiment 9b.

Write a C Program for LFU Page Replacement Algorithm

LFU Page Replacement Algorithm: Step-by-Step Procedure

Step 1: Initialize Data Structures

1. Initialize a data structure to hold the page frames in memory. Each entry should include the page number and a counter to track its frequency of access.
2. Initialize a variable to keep track of the total number of page frames.

Step 2: Page Request

1. When a page request is received:
    - Check if the requested page is already in a page frame.
        - If yes, it's a page hit. Increment the frequency counter of the corresponding page frame. Continue to the next page request.
        - If no, it's a page fault (miss):
            - Check if there is an empty page frame available.
                - If yes, place the requested page in the empty page frame and set its frequency counter to 1.
                - If no, a page replacement is required:
                    - Find the page frame with the lowest frequency count. If there are ties, choose the one that arrived earliest.
                    - Replace the chosen page frame with the requested page and set its frequency counter to 1.

Step 3: Repeat

1. Continue processing page requests one by one, following the steps outlined in Step 2.
2. Keep track of the number of page faults that occur during this process.

Step 4: End

1. Once all page requests have been processed, calculate and report the total number of page faults.

Example: Let's consider a simple example with a page frame size of 3 and the following page request sequence: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5.

Initial state: Page frames are empty. Page request 1: Page frames: [1 (1)], Page Fault. Page request 2: Page frames: [1 (1), 2 (1)], Page Fault. Page request 3: Page frames: [1 (1), 2 (1), 3 (1)], Page Fault. Page request 4: Page frames: [2 (1), 3 (1), 4 (1)], Page Fault. Page request 1: Page frames: [2 (1), 3 (1), 4 (2)], Page Hit (Increment frequency). Page request 2: Page frames: [2 (1), 3 (1), 4 (3)], Page Hit (Increment frequency). Page request 5: Page frames: [3 (1), 4 (3), 5 (1)], Page Fault (Replace 2 with the lowest frequency). ...and so on.

Total page faults: 6.

```
#include <stdio.h>
#include <stdbool.h>
#include <limits.h>

#define MAX_FRAMES 3
#define MAX_PAGES 20

typedef struct {
   int page_number;
   int frequency;
} Page;

Page frames[MAX_FRAMES];
int num_frames = 0;

int page_queue[MAX_PAGES];
int num_pages = 0;

void initializeFrames() {
   for (int i = 0; i < MAX_FRAMES; i++) {
      frames[i].page_number = -1;
      frames[i].frequency = 0;
   }
}
```

```
int findPage(int page_number) {
   for (int i = 0; i < num_frames; i++) {
      if (frames[i].page_number == page_number) {
         return i;
      }
   }
   return -1;
}

int findLFUPage() {
   int min_frequency = INT_MAX;
   int min_frequency_index = -1;

   for (int i = 0; i < num_frames; i++) {
      if (frames[i].frequency < min_frequency) {
         min_frequency = frames[i].frequency;
         min_frequency_index = i;
      }
   }
   return min_frequency_index;
}

void displayFrames() {
   printf("Current frames: ");
   for (int i = 0; i < num_frames; i++) {
      if (frames[i].page_number != -1) {
         printf("%d (%d) ", frames[i].page_number, frames[i].frequency);
      }
   }
   printf("\n");
}

void replacePage(int page_number) {
   int replace_index = findLFUPage();
   frames[replace_index].page_number = page_number;
   frames[replace_index].frequency = 1;
}

int main() {
   initializeFrames();
```

```
    int page_number;
    printf("Enter the number of pages: ");
    scanf("%d", &num_pages);

    printf("Enter the page reference sequence:\n");
    for (int i = 0; i < num_pages; i++) {
        scanf("%d", &page_queue[i]);
    }

    for (int i = 0; i < num_pages; i++) {
        page_number = page_queue[i];
        int frame_index = findPage(page_number);

        if (frame_index != -1) {
            frames[frame_index].frequency++;
        } else {
            if (num_frames < MAX_FRAMES) {
                frames[num_frames].page_number = page_number;
                frames[num_frames].frequency = 1;
                num_frames++;
            } else {
                replacePage(page_number);
            }
        }

        printf("Page reference: %d\n", page_number);
        displayFrames();
    }

    return 0;
}
```