

OPERATING SYSTEMS

LAB MANUAL

Regulations: R20
Class: III Year I Semester (IT)

Prepared By
Mrs. shaik karimunnisa
Assistant Professor
IT Department

PROGRAM OUTCOMES	
PO1	Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
PO2	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.
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 of the 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 to the 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 need for 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 leader in diverse teams, and in multidisciplinary 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 receive clear 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 EDUCATIONAL OUTCOMES

PEO1	To induce strong foundation in mathematical and core concepts, which enable them to participate in research , in the field of computer science.
PEO2	To be able to become the part of application development and problem solving by learning the computer programming methods, of the industry and related domains.
PEO3	To Gain the multidisciplinary knowledge by understanding the scope of association of computer science engineering discipline with other engineering disciplines.
PEO4	To improve the communication skills, soft skills, organizing skills which build the professional qualities, there by understanding the social responsibilities and ethical attitude.

OPERATING SYSTEMS LAB SYLLABUS

S. No.	List of Experiments
1	Write a C program simulate the following CPU scheduling algorithms: a)FCFS b)SJF c) Round Robin d) Priority
2	Write programs using the I/O system calls of UNIX/LINUX operating system.
3	Write a C program to simulate Bankers Algorithm for Deadlock Avoidance and Prevention
4	Write a program to implement the Producer – Consumer problem using semaphores using UNIX/LINUX system calls.
5	Write a C program to illustrate the following IPC mechanisms. a)Pipes b) FIFO c) Message Queues d) Shared Memory
6	Write a C program to simulate the following memory segment techniques a) Paging b) Segmentation

COURSE OBJECTIVES:

This lab complements the operating systems course. With this course Students are able to:

- COB 1:** To write programs in Linux Environment using system calls.
- COB 2:** To implement scheduling algorithms.
- COB3:** To implement page replacement algorithm.
- COB4:** To implement file allocation methods.
- COB5:** To understand and implement ipc mechanism using named and unnamed pipe.
- COB6:** To develop solutions for synchronization problems using semaphores.

COURSE OUTCOMES:

- CO1: Understand** and implement basic services and functionalities of the operating system using system calls and able to **Understand** the benefits of thread over process and implement synchronized programs using multithreading concepts.
- CO2: Use** modern operating system calls and synchronization libraries in software/ hardware interfaces.
- CO3: Analyze** and simulate CPU Scheduling Algorithms like FCFS, Round Robin, SJF, and Priority.
- CO4 :Implement** memory management schemes and page replacement schemes.
- CO5: Simulate** file allocation and organization techniques.
- CO6: Understand** the concepts of deadlock in operating systems and implement them in multiprogramming system.

**ATTAINMENT OF PROGRAM OUTCOMES & PROGRAM EDUCATION
OUTCOMES**

Exp. No.	Experiment	Program Outcomes Attained	Program Specific Outcomes Attained
1	Write a C program simulate the following CPU scheduling algorithms: a)FCFS b)SJF c) Round Robin d) Priority	PO1, PO2, PO4	PEO1
2	Write programs using the I/O system calls of UNIX/LINUX operating system.	PO1, PO2, PO4	PEO1
3	Write a C program to simulate Bankers Algorithm for Deadlock Avoidance and Prevention	PO1, PO2, PO4	PEO1
4	Write a program to implement the Producer – Consumer problem using semaphores using UNIX/LINUX system calls.	PO1, PO2, PO4	PEO1
5	Write a C program to illustrate the following IPC mechanisms. a)Pipes b) FIFO c) Message Queues d) Shared Memory	PO1, PO2, PO4	PEO1,PEO2
6	Write a C program to simulate the following memory segment techniques a) Paging b) Segmentation	PO1, PO2	PEO1,PEO2

ATTAINMENT OF COURSE OBJECTIVES & COURSE OUTCOMES

Exp. No.	Experiment	COURSE OBJECTIVES Attained	COURSE OUTCOMES Attained
1	Write a C program simulate the following CPU scheduling algorithms: a)FCFS b)SJF c) Round Robin d) Priority	COB1,COB3	CO3
2	Write programs using the I/O system calls of UNIX/LINUX operating system.	COB1,COB2	CO1,CO5
3	Write a C program to simulate Bankers Algorithm for Deadlock Avoidance and Prevention	COB2	CO1,CO2
4	Write a program to implement the Producer – Consumer problem using semaphores using UNIX/LINUX system calls.	COB1	CO1,CO5
5	Write a C program to illustrate the following IPC mechanisms. a)Pipes b) FIFO c) Message Queues d) Shared Memory	COB5	CO1,CO6
6	Write a C program to simulate the following memory segment techniques c) Paging b) Segmentation	COB5	CO1,CO6

EXPERIMENT-1

Write a C program simulate the following CPU scheduling algorithms:

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

a) FCFS

DESCRIPTION

Assume all the processes arrive at the same time.

FCFS CPU SCHEDULING ALGORITHM

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

CPU SCHEDULING

Maximum CPU utilization obtained with multiprogramming

CPU-I/O Burst Cycle – Process execution consists of a *cycle* of

CPU execution and I/O wait

CPU burst distribution

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

Process	Burst Time
---------	------------

P1	24
----	----

P2	3
----	---

P3	3
----	---

Suppose that the processes arrive in the order: P1 , P2 , P3

The Gantt Chart for the schedule is:



Waiting time for P1 = 0; P2 = 24; P3 = 27

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

ALGORITHM

1. Start
2. Declare the array size
3. Read the number of processes to be inserted
4. Read the Burst times of processes
5. calculate the waiting time of each process
 $wt[i+1]=bt[i]+wt[i]$
6. calculate the turnaround time of each process
 $tt[i+1]=tt[i]+bt[i+1]$
7. Calculate the average waiting time and average turnaround time.
8. Display the values
9. Stop

PROGRAM:

```
#include<stdio.h>
void main()
{
int i,j,bt[10],n,wt[10],tt[10],w1=0,t1=0;
float aw,at;
printf("enter no. of processes:\n");
scanf("%d",&n);
printf("enter the burst time of processes:");
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\t wt\t tt\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

Enter no of processes

3

enter bursttime

12

8

20

EXPECTED OUTPUT

bt wt tt

12 0 12

8 12 20

20 20 40

aw=10.666670

at=24.00000

b) SJF

SJF CPU SCHEDULING ALGORITHM

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

HARDWARE REQUIREMENTS: Intel based Desktop Pc
RAM of 512 MB

SOFTWARE REQUIREMENTS:
Turbo C/ Borland C.

THEORY:

Example of Non Preemptive SJF

Process	Arrival Time	Burst Time
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

Example of Preemptive SJF

Process	Arrival Time	Burst Time
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
----	----	----	----	----	----

Average waiting time = (9 + 1 + 0 + 2)/4 = 3

ALGORITHM

1. Start
2. Declare the array size
3. Read the number of processes to be inserted
4. Read the Burst times of processes
5. sort the Burst times in ascending order and process with shortest burst time is first executed.
6. calculate the waiting time of each process
 $wt[i+1]=bt[i]+wt[i]$
7. calculate the turnaround time of each process
 $tt[i+1]=tt[i]+bt[i+1]$
8. Calculate the average waiting time and average turnaround time.
9. Display the values
10. Stop

PROGRAM:

```
#include<stdio.h>
void main()
{
int i,j,bt[10],t,n,wt[10],tt[10],w1=0,t1=0;
float aw,at;
printf("enter no. of processes:\n");
scanf("%d",&n);
printf("enter the burst time of processes:");
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\t wt\t tt\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:

```
enter no of processes
3
enter burst time
12
8
20
```

OUTPUT:

```
bt wt tt
12 8 20
8 0 8
20 20 40
aw=9.33
at=22.64
```

c) Round Robin

DESCRIPTION

Assume all the processes arrive at the same time.

ROUND ROBIN CPU SCHEDULING ALGORITHM

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

HARDWARE REQUIREMENTS: Intel based Desktop Pc RAM of 512 MB

SOFTWARE REQUIREMENTS: Turbo C/ Borland C.

THEORY:

Round Robin:

Example of RR with time quantum=3

Process	Burst time
aaa	4
Bbb	3
Ccc	2
Ddd	5
Eee	1

ALGORITHM

1. Start
2. Declare the array size
3. Read the number of processes to be inserted
4. Read the burst times of the processes
5. Read the Time Quantum
6. if the burst time of a process is greater than time Quantum then subtract time quantum form the burst time
 Else
 Assign the burst time to time quantum.
7. calculate the average waiting time and turn around time of the processes.
8. Display the values
9. Stop

PROGRAM:

```
#include<stdio.h>
void main()
{
int st[10],bt[10],wt[10],tat[10],n,tq;
int i,count=0,swt=0,stat=0,temp,sq=0;
float awt=0.0,atat=0.0;
printf("Enter number of processes:");
scanf("%d",&n);
printf("Enter burst time for sequences:");
for(i=0;i<n;i++)
{
scanf("%d",&bt[i]);
st[i]=bt[i];
}
printf("Enter time quantum:");
scanf("%d",&tq);
while(1)
{
for(i=0,count=0;i<n;i++)
{
temp=tq;
if(st[i]==0)
{
count++;
continue;
}
if(st[i]>tq)
st[i]=st[i]-tq;
else
if(st[i]>=0)
{
temp=st[i];
st[i]=0;
}
sq=sq+temp;
tat[i]=sq;
}
if(n==count)
break;
}
for(i=0;i<n;i++)
{
wt[i]=tat[i]-bt[i];
swt=swt+wt[i];
}
```

```
stat=stat+tat[i];
}
awt=(float)swt/n;
atat=(float)stat/n;
printf("Process_no Burst time Wait time Turn around time");
for(i=0;i<n;i++)
printf("\n%d\t %d\t %d\t %d",i+1,bt[i],wt[i],tat[i]);
printf("\nAvg wait time is %f Avg turn around time is %f",awt,atat);
}
```

Input:

Enter no of jobs

4

Enter burst time

5

12

8

20

Output:

Bt wt tt

5 0 5

12 5 13

8 13 25

20 25 45

aw=10.75000

at=22.000000

d) Priority

HARDWARE REQUIREMENTS: Intel based Desktop Pc
RAM of 512 MB

SOFTWARE REQUIREMENTS:
Turbo C/ Borland C.

THEORY:

In Priority Scheduling, each process is given a priority, and higher priority methods are executed first, while equal priorities are executed [First Come First Served](#) or [Round Robin](#).

There are several ways that priorities can be assigned:

- Internal priorities are assigned by technical quantities such as memory usage, and file/IO operations.
- External priorities are assigned by politics, commerce, or user preference, such as importance and amount being paid for process access (the latter usually being for mainframes).

ALGORITHM

1. Start
2. Declare the array size
3. Read the number of processes to be inserted
4. Read the Priorities of processes
5. sort the priorities and Burst times in ascending order
5. calculate the waiting time of each process
 $wt[i+1]=bt[i]+wt[i]$
6. calculate the turnaround time of each process
 $tt[i+1]=tt[i]+bt[i+1]$
7. Calculate the average waiting time and average turnaround time.
8. Display the values
9. Stop

PROGRAM:

```
#include<stdio.h>
void main()
{
int i,j,pno[10],prior[10],bt[10],n,wt[10],tt[10],w1=0,t1=0,s;
float aw,at;
printf("enter the number of processes:");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("The process %d:\n",i+1);
printf("Enter the burst time of processes:");
scanf("%d",&bt[i]);
printf("Enter the priority of processes %d:",i+1);
scanf("%d",&prior[i]);
pno[i]=i+1;
}
for(i=0;i<n;i++)
{
for(j=0;j<n;j++)
{
if(prior[i]<prior[j])
{
s=prior[i];
prior[i]=prior[j];
prior[j]=s;

s=bt[i];
bt[i]=bt[j];
bt[j]=s;

s=pno[i];
pno[i]=pno[j];
pno[j]=s;
}
}
}
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(" \n job \t bt \t wt \t tat \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:

```
Enter no of jobs
4
Enter bursttime
10
2
4
7
Enter priority values
4
2
1
3
```

Output:

```
Bt priority wt tt
4 1 0 4
2 2 4 6
7 3 6 13
10 4 13 23
aw=5.750000
at=12.500000
```

VIVA QUESTIONS:

1. Round Robin scheduling is used in
(A) Disk scheduling. (B) CPU scheduling
(C) I/O scheduling. (D) Multitasking
2. What are the dis-advantages of RR Scheduling Algorithm?
3. What are the advantages of RR Scheduling Algorithm?
4. Super computers typically employ _____.
1 Real time Operating system 2 Multiprocessors OS
3 desktop OS 4 None of the above
5. An optimal scheduling algorithm in terms of minimizing the average waiting time of a given set of processes is _____.
1 FCFS scheduling algorithm 2 Round robin scheduling algorithm
3 Shortest job - first scheduling algorithm 4 None of the above
6. The optimum CPU scheduling algorithm is
(A) FIFO (B) SJF with preemption. (C) SJF without preemption. (D) Round Robin.
7. In terms of average wait time the optimum scheduling algorithm is
(A) FCFS (B) SJF (C) Priority (D) RR
8. What are the dis-advantages of SJF Scheduling Algorithm?
9. What are the advantages of SJF Scheduling Algorithm?
10. Define CPU Scheduling algorithm?
11. What is First-Come-First-Served (FCFS) Scheduling?
12. Why CPU scheduling is required?
13. Which technique was introduced because a single job could not keep both the CPU and the I/O devices busy?
1) Time-sharing 2) SPOOLing 3) Preemptive scheduling 4) Multiprogramming
14. CPU performance is measured through _____.
1) Throughput 2) MHz 3) Flaps 4) None of the above
15. Which of the following is a criterion to evaluate a scheduling algorithm?
1 CPU Utilization: Keep CPU utilization as high as possible.
2 Throughput: number of processes completed per unit time.

- 3 Waiting Time: Amount of time spent ready to run but not running.
- 4 All of the above

- 16. Priority CPU scheduling would most likely be used in a _____ os.
- 17. CPU allocated process to _____ priority.
- 18. Calculate avg waiting time=
- 19. Maximum CPU utilization obtained with _____
- 20. Using _____ algorithms find the min & max waiting time. `]oiui

EXPERIMENT - 2

Objective:

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

PROGRAM:

```
#include <unistd.h>
#include <fcntl.h>
#include <string.h>
#include <stdio.h>
int main()
{
int fd[2];
char buf1[25] = "just a test\n";
char buf2[100];
fd[0] = open("tfile",O_RDWR);
fd[1] = open("tfile",O_RDWR);
write(fd[0],buf1,strlen(buf1));
printf("\nEnter your text now...");
gets(buf1);
write(fd[0],buf1,strlen(buf1));
write(1, buf2, read(fd[1],buf2,sizeof(buf2)));
close(fd[0]);
close(fd[1]);
printf("\n");
return 0;
}
```

VIVA QUESTIONS

1. Whether a system call is a routine built into the kernel and performs a basic function?
 - a) True
 - b) False
2. When we execute a C program, CPU runs in ____ mode.
3. In ____ mode, the kernel runs on behalf of the user.
4. All UNIX and LINUX systems have one thing in common which is ____
5. The chmod command invokes the ____ system call.
6. For reading input, which of the following system call is used?
7. Which of the following system call is used for opening or creating a file?
8. There are ___ modes of opening a file.
9. Which of the following mode is used for opening a file in both reading and writing?
10. open system call returns the file descriptor as ____

EXPERIMENT-3

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

OBJECTIVE

Write a C program to simulate Bankers Algorithm for Deadlock Avoidance

DESCRIPTION

In a multiprogramming environment, several processes may compete for a finite number of resources. A process requests resources; if the resources are not available at that time, the process enters a waiting state. Sometimes, a waiting process is never again able to change state, because the resources it has requested are held by other waiting processes. This situation is called a deadlock. Deadlock avoidance is one of the techniques for handling deadlocks. This approach requires that the operating system be given in advance additional information concerning which resources a process will request and use during its lifetime. With this additional knowledge, it can decide for each request whether or not the process should wait. To decide whether the current request can be satisfied or must be delayed, the system must consider the resources currently available, the resources currently allocated to 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.

NAME OF EXPERIMENT: Simulate Banker's Algorithm for Deadlock Avoidance.

AIM: Simulate Banker's Algorithm for Deadlock Avoidance to find whether the system is in safe state or not.

HARDWARE REQUIREMENTS: Intel based Desktop Pc
RAM of 512 MB

SOFTWARE REQUIREMENTS: Turbo C/ Borland C.

THEORY:

DEAD LOCK AVOIDANCE

To implement deadlock avoidance & Prevention by using Banker's Algorithm.

Banker's Algorithm:

When a new process enters a system, it must declare the maximum number of instances of each resource type it needed. This number may exceed the total number of resources in the system. When the user request a set of resources, the system must determine whether the allocation of each 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.

Data structures

- n-Number of process, m-number of resource types.
- Available: Available[j]=k, k – instance of resource type R_j is available.
- Max: If max[i, j]=k, P_i may request at most k instances resource R_j.
- Allocation: If Allocation [i, j]=k, P_i allocated to k instances of resource R_j
- Need: If Need[I, j]=k, P_i may need k more instances of resource type R_j,

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

Safety Algorithm

1. Work and Finish be the vector of length m and n respectively, Work=Available and Finish[i] =False.
2. Find an i such that both
 - Finish[i] =False
 - Need<=WorkIf no such I exists go to step 4.
3. work=work+Allocation, Finish[i] =True;
4. if Finish[1]=True for all I, then the system is in safe state.

Resource request algorithm

Let Request i be request vector for the process Pi, If request i=[j]=k, then process Pi wants k instances of resource type Rj.

1. if Request<=Need I go to step 2. Otherwise raise an error condition.
2. if Request<=Available go to step 3. Otherwise Pi must since the resources are available.
3. Have the system pretend to have allocated the requested resources to process Pi by modifying the state as follows;
Available=Available-Request I;
Allocation I =Allocation+Request I;
Need i=Need i-Request I;

If the resulting resource allocation state is safe, the transaction is completed and process Pi is allocated 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. Start the program.
2. Get the values of resources and processes.
3. Get the avail value.
4. After allocation find the need value.
5. Check whether its possible to allocate.
6. If it is possible then the system is in safe state.
7. Else system is not in safety state.
8. If the new request comes then check that the system is in safety.
9. or not if we allow the request.
10. stop the program.

PROGRAM:

```
#include<stdio.h>
struct da {
int max[10],al[10],need[10],before[10],after[10];
}p[10];
void main() {
int i,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("\n Enter the no of resources:");
scanf("%d",&r);
for(i=0;i<n;i++) {
printf("process %d \n",i+1);
for(j=0;j<r;j++) {
printf("maximum value for resource %d:",j+1);
scanf("%d",&p[i].max[j]);
}
for(j=0;j<r;j++) {
printf("allocated from resource %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("Enter total value of resource %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\t max allocated needed total avail");
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\t AVAIL BEFORE \t AVAIL AFTER");
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("\n the above sequence is a safe sequence");  
else  
printf("\n deadlock occured");  
}
```

OUTPUT:

//TEST CASE 1:

ENTER THE NO. OF PROCESSES:4

ENTER THE NO. OF RESOURCES:3

PROCESS 1

MAXIMUM VALUE FOR RESOURCE 1:3

MAXIMUM VALUE FOR RESOURCE 2:2

MAXIMUM VALUE FOR RESOURCE 3:2

ALLOCATED FROM RESOURCE 1:1

ALLOCATED FROM RESOURCE 2:0

ALLOCATED FROM RESOURCE 3:0

PROCESS 2

MAXIMUM VALUE FOR RESOURCE 1:6

MAXIMUM VALUE FOR RESOURCE 2:1

MAXIMUM VALUE FOR RESOURCE 3:3

ALLOCATED FROM RESOURCE 1:5

ALLOCATED FROM RESOURCE 2:1

ALLOCATED FROM RESOURCE 3:1

PROCESS 3

MAXIMUM VALUE FOR RESOURCE 1:3

MAXIMUM VALUE FOR RESOURCE 2:1

MAXIMUM VALUE FOR RESOURCE 3:4

ALLOCATED FROM RESOURCE 1:2

ALLOCATED FROM RESOURCE 2:1

ALLOCATED FROM RESOURCE 3:1

PROCESS 4

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

ENTER TOTAL VALUE OF RESOURCE 1:9

ENTER TOTAL VALUE OF RESOURCE 2:3

ENTER TOTAL VALUE OF RESOURCE 3:6

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

AVAIL BEFORE AVAIL AFTER

P 2	010	623
P 1	401	723
P 3	620	934
P 4	514	936

THE ABOVE SEQUENCE IS A SAFE SEQUENCE

VIVA QUESTIONS:

1. Differentiate deadlock avoidance and fragmentation
2. Tell me the real time example where this deadlock occurs?
3. How do we calculate the need for process?
4. What is the name of the algorithm to avoid deadlock?
5. Banker's algorithm for resource allocation deals with
(A) Deadlock prevention. (B) Deadlock avoidance.
(C) Deadlock recovery. (D) Mutual exclusion
6. Each request requires that the system consider the _____ to decide whether the current request can be satisfied or must wait to avoid a future possible deadlock.
7. Given a priori information about the _____ number of resources of each type that maybe requested for each process, it is possible to construct an algorithm that ensures that the system will never enter a deadlock state.
8. A deadlock avoidance algorithm dynamically examines the _____ to ensure that a circular wait condition can never exist.
9. Define Safe State.

10. A system is in a safe state only if there exists a _____.
11. Is All unsafe states are deadlocks?
12. A system has 12 magnetic tape drives and 3 processes : P0, P1, and P2. Process P0 requires 10 tape drives, P1 requires 4 and P2 requires 9 tape drives.
- Process
- P0
- P1
- P2
- Maximum needs (process-wise : P0 through P2 top to bottom)
- 10
- 4
- 9
- Currently allocated (process-wise)
- 5
- 2
- 2
- Which of the following sequence is a safe sequence ?
- a) P0, P1, P2
- b) P1, P2, P0
- c) P2, P0, P1
- d) P1, P0, P2
13. If no cycle exists in the resource allocation graph then _____
14. The resource allocation graph is not applicable to a resource allocation system : with_
-
15. The Banker's algorithm is _____ than the resource allocation graph algorithm.
16. List data structures available in the Banker's algorithm.
- The data structures available in the Banker's algorithm are :
- a) Available
- b) Need
- c) Allocation
17. What is the content of the matrix Need is :
18. A system with 5 processes P0 through P4 and three resource types A, B, C has A with 10 instances, B with 5 instances, and C with 7 instances. At time t0, the following snapshot has been taken :
- Process
- P0
- P1
- P2
- P3
- P4
- Allocation (process-wise : P0 through P4 top TO bottom)
- | A | B | C |
|---|---|---|
| 0 | 1 | 0 |
| 2 | 0 | 0 |
| 3 | 0 | 2 |

2 1 1
0 0 2

MAX (process-wise : P0 through P4 top TO bottom)

A B C

7 5 3

3 2 2

9 0 2

2 2 2

4 3 3

Available

A B C

3 3 2

The sequence <P1, P3, P4, P2, P0> leads the system to :

- a) an unsafe state
- b) a safe state
- c) a protected state
- d) a deadlock

Answer:

19. The wait-for graph is a deadlock detection algorithm that is applicable when : _____

20. What does an edge from process P_i to P_j in a wait for graph indicates ?:

P_i is waiting for P_j to release a resource that P_i needs.

21. If the wait for graph contains a cycle : _____.

22. If deadlocks occur frequently, the detection algorithm must be invoked _____.

23. The disadvantage of invoking the detection algorithm for every request is _____

24. 'm' processes share 'n' resources of the same type. The maximum need of each process doesn't exceed 'n' and the sum of all their maximum needs is always less than $m+n$. In this setup, deadlock :

- a) can never occur
- b) may occur
- c) has to occur
- d) none of the mentioned

20. A system has 3 processes sharing 4 resources. If each process needs a maximum of 2 units then, deadlock :

- a) can never occur
- b) may occur
- c) has to occur
- d) none of the mentioned

OBJECTIVE

Write a C program to simulate Bankers algorithm for Deadlock Prevention

DESCRIPTION

In a multiprogramming environment, several processes may compete for a finite number of resources. A process requests resources; if the resources are not available at that time, the process enters a waiting state. Sometimes, a waiting process is never again able to change state, because the resources it has requested are held by other waiting processes. This situation is called a deadlock. Deadlock avoidance is one of the techniques for handling deadlocks. This approach requires that the operating system be given in advance additional information concerning which resources a process will request and use during its lifetime. With this additional knowledge, it can decide for each request whether or not the process should wait. To decide whether the current request can be satisfied or must be delayed, the system must consider the resources currently available, the resources currently allocated to 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.

NAME OF EXPERIMENT: Simulate Algorithm for Deadlock prevention.

AIM: Simulate Algorithm for Deadlock prevention .

HARDWARE REQUIREMENTS: Intel based Desktop Pc
RAM of 512 MB

SOFTWARE REQUIREMENTS: Turbo C/ Borland C.

THEORY:

Deadlock Definition:

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:

- waiting for access to a critical section
- waiting for a resource Note that it is usually a non-preemptable (resource).
-

Conditions for Deadlock :

- Mutual exclusion: resources cannot be shared.
- Hold and wait: processes request resources incrementally, and hold on to What they've got.
- No preemption: resources cannot be forcibly taken from processes.
- Circular wait: circular chain of waiting, in which each process is waiting for a resource held by the next process in the chain.

Strategies for dealing with Deadlock :

- ignore the problem altogether
- detection and recovery
- avoidance by careful resource allocation
- prevention by structurally negating one of the four necessary conditions.

Deadlock Prevention :

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

Algorithm:

1. Start
2. Attacking Mutex condition: never grant exclusive access. but this may not be possible for several resources.
3. Attacking preemption: not something you want to do.
4. Attacking hold and wait condition : make a process hold at the most 1 resource at a time. make all the requests at the beginning. All or nothing policy. If you feel, retry. eg. 2-phase locking
5. Attacking circular wait: Order all the resources. Make sure that the requests are issued in the correct order so that there are no cycles present in the resource graph. Resources numbered 1 ... n. Resources can be requested only in increasing order. ie. you cannot request a resource whose no is less than any 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\nSIMULATION OF DEADLOCK PREVENTION");
printf("Enter no. of processes, resources");
scanf("%d%d",&p,&r);printf("Enter allocation matrix");
for(i=0;i<p;i++)
for(j=0;j<r;j++)
scanf("%d",&alloc[i][j]);
printf("enter max matrix");
for(i=0;i<p;i++) /*reading the maximum matrix and available matrix*/
for(j=0;j<r;j++)
scanf("%d",&max[i][j]);
printf("enter available matrix");
for(i=0;i<r;i++)
scanf("%d",&avail[i]);
```

```
for(i=0;i<p;i++)
for(j=0;j<r;j++)
need[i][j]=max[i][j]-alloc[i][j];
fun(); /*calling function*/
if(flag==0)
{
i
f(finish[i]!=1)
{
printf("\n\n Failing :Mutual exclusion");
for(j=0;j<r;j++)
{ /*checking for mutual exclusion*/
if(avail[j]<need[i][j])
avail[j]=need[i][j];
} fun();
printf("\n By allocating required resources to process %d dead lock is prevented ",i);
printf("\n\n lack of preemption");
for(j=0;j<r;j++)
{
if(avail[j]<need[i][j])
avail[j]=need[i][j];
alloc[i][j]=0;
}

fun( );
printf("\n\n daed lock is prevented by allocating needed resources");
printf(" \n \n failing:Hold and Wait condition ");
for(j=0;j<r;j++)
{ /*checking hold and wait condition*/
if(avail[j]<need[i][j])
avail[j]=need[i][j];
}
fun( );
printf("\n AVOIDING ANY ONE OF THE CONDITION, U CAN PREVENT DEADLOCK");
}
}
}
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:

SIMULATION OF DEADLOCK PREVENTION

Enter no. of processes, resources 3, 2

enter allocation matrix 2 4 5

3 4 5

Enter max matrix4 3 4

5 6 1

Enter available matrix2

5

Failing : Mutual Exclusion

by allocating required resources to process dead is prevented

Lack of no preemption deadlock is prevented by allocating needed resources

Failing : Hold and Wait condition

VIVA QUESTIONS:

1. The Banker's algorithm is used for _____.
2. _____ is the situation in which a process is waiting on another process, which is also waiting on another process ... which is waiting on the first process. None of the processes involved in this circular wait are making progress.
3. what is safe state?
4. What are the conditions that cause deadlock?
5. How do we calculate the need for process?

6. The number of resources requested by a process : must not exceed the total number of _____
7. The request and release of resources are _____
8. Multithreaded programs are : _____.
9. For a deadlock to arise, which conditions must hold simultaneously
 - a) Mutual exclusion
 - b) No preemption
 - c) Hold and wait
10. For _____ to prevail in the system at least one resource must be held in a non sharable mode.
11. For a Hold and wait condition to prevail what is require?
12. _____ is a set of methods to ensure that at least one of the necessary conditions cannot hold.
13. For _____ resources like a printer, mutual exclusion must exist.
14. For sharable resources, mutual exclusion is not required.
15. To ensure that the hold and wait condition never occurs in the system, it must be ensured what?
 - a) whenever a resource is requested by a process, it is not holding any other resources
 - b) each process must request and be allocated all its resources before it begins its execution
 - c) a process can request resources only when it has none
16. The disadvantage of a process being allocated all its resources before beginning its execution is _____
17. To ensure _____, if a process is holding some resources and requests another resource that cannot be immediately allocated to it then all resources currently being held are pre-empted.
18. A _____ can be broken by abort one or more processes to break the circular wait.
19. What are the two ways of aborting processes and eliminating deadlocks ? :
20. Those processes should be aborted on occurrence of a deadlock, the termination of which : _____.

EXPERIMENT-4

Write a program to implement the Producer – Consumer problem using semaphores using UNIX/LINUX system calls.

OBJECTIVE

To implement the Producer – Consumer problem using semaphores using UNIX/LINUX system calls.

DESCRIPTION

Producer consumer problem is also known as bounded buffer problem. In this problem we have two processes, producer and consumer, who share a fixed size buffer. Producer work is to produce data or items and put in buffer. Consumer work is to remove data from buffer and consume it. We have to make sure that producer do not produce data when buffer is full and consumer do not remove data 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>

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

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

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

return 0;
}

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

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

void producer()
{
    mutex=wait(mutex);
    full=signal(full);
    empty=wait(empty);
    x++;
    printf("\nProducer produces the item %d",x);
    mutex=signal(mutex);
}

void consumer()
{
    mutex=wait(mutex);
    full=wait(full);
    empty=signal(empty);
    printf("\nConsumer consumes item %d",x);
    x--;
    mutex=signal(mutex);
}
```


Output

- 1.Producer
- 2.Consumer
- 3.Exit

Enter your choice:1

Producer produces the item 1

Enter your choice:2

Consumer consumes item 1

Enter your choice:2

Buffer is empty!!

Enter your choice:1

Producer produces the item 1

Enter your choice:1

Producer produces the item 2

Enter your choice:1

Producer produces the item 3

Enter your choice:1

Buffer is full!!

Enter your choice:3

EXPERIMENT-5

Write a C program to illustrate the following IPC mechanisms

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

OBJECTIVE:

Write a C program to illustrate the Pipes IPC mechanism

PROGRAM

```
#include <stdio.h>
#include <unistd.h>
#define MSGSIZE 16
char* msg1 = "hello, world #1";
char* msg2 = "hello, world #2";
char* msg3 = "hello, world #3";

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

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

    /* continued */
    /* write pipe */

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

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

Output:

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

OBJECTIVE

b) Write a C program to illustrate the FIFO IPC mechanism

PROGRAM

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

int main()
{
    int fd;

    // FIFO file path
    char * myfifo = "/tmp/myfifo";

    // Creating the named file(FIFO)
    // mkfifo(<pathname>, <permission>)
    mkfifo(myfifo, 0666);

    char arr1[80], arr2[80];
    while (1)
    {
        // Open FIFO for write only
        fd = open(myfifo, O_WRONLY);

        // Take an input arr2ing from user.
        // 80 is maximum length
        fgets(arr2, 80, stdin);

        // Write the input arr2ing on FIFO
        // and close it
        write(fd, arr2, strlen(arr2)+1);
        close(fd);

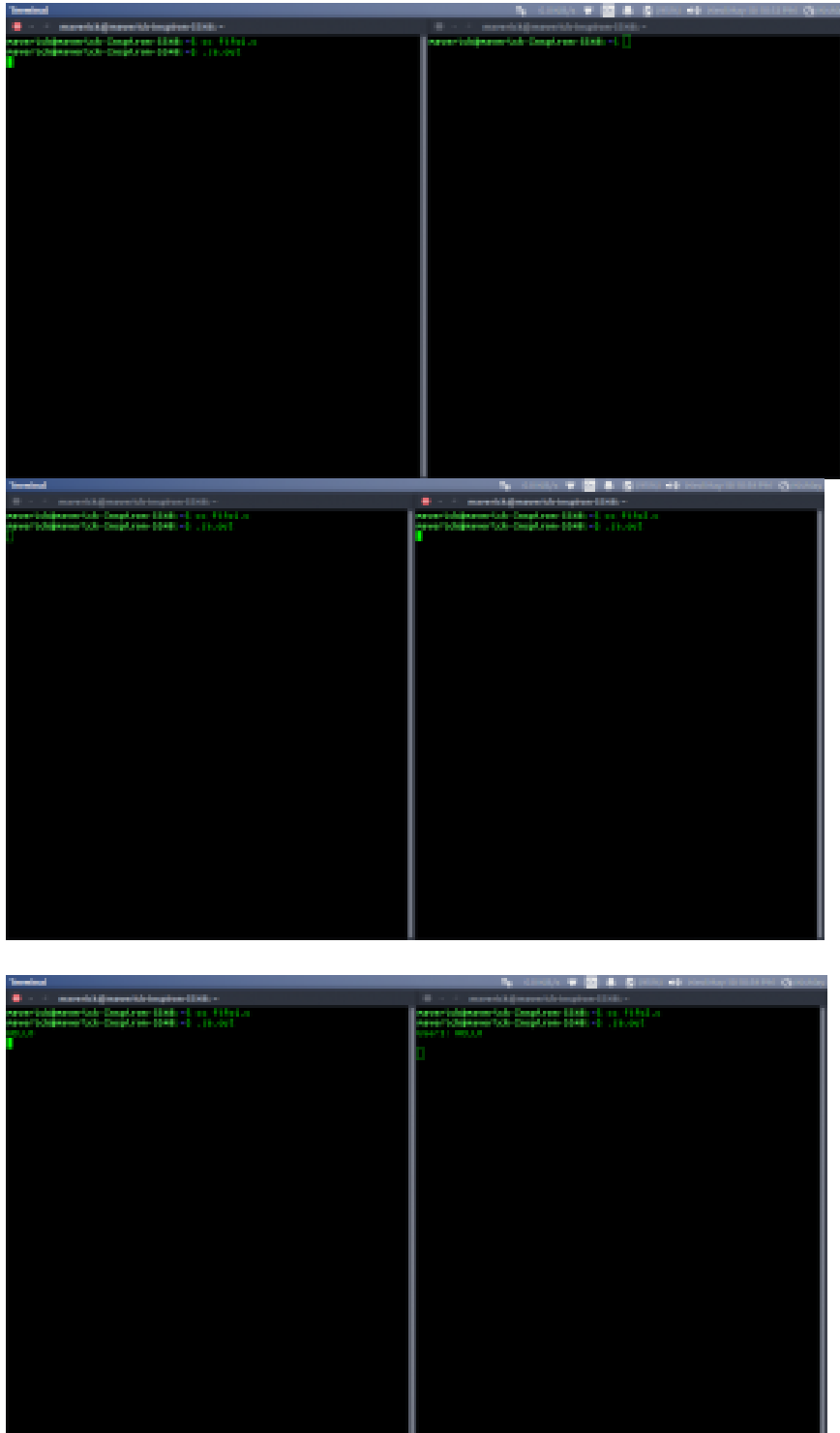
        // Open FIFO for Read only
        fd = open(myfifo, O_RDONLY);

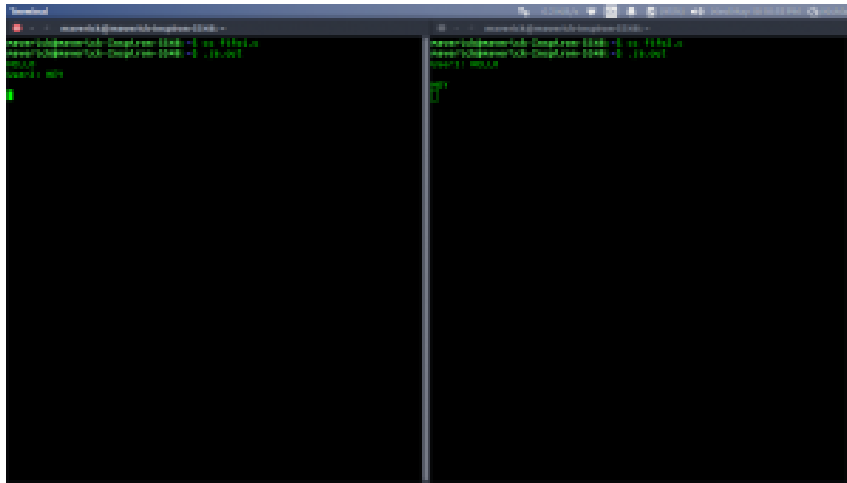
        // Read from FIFO
        read(fd, arr1, sizeof(arr1));

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

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

Output:





OBJECTIVE

c) Write a C program to illustrate the Message Queue IPC mechanism

PROGRAM:

```
// C Program for Message Queue (Writer Process)
#include <stdio.h>
#include <sys/ipc.h>
#include <sys/msg.h>

// structure for message queue
struct mesg_buffer {
    long mesg_type;
    char mesg_text[100];
} message;

int main()
{
    key_t key;
    int msgid;

    // ftok to generate unique key
    key = ftok("progfile", 65);

    // msgget creates a message queue
    // and returns identifier
    msgid = msgget(key, 0666 | IPC_CREAT);
    message.mesg_type = 1;

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

    // msgsnd to send message
    msgsnd(msgid, &message, sizeof(message), 0);

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

    return 0;
}

// C Program for Message Queue (Reader Process)
#include <stdio.h>
#include <sys/ipc.h>
#include <sys/msg.h>
```

```
// structure for message queue
struct mesg_buffer {
    long mesg_type;
    char mesg_text[100];
} message;

int main()
{
    key_t key;
    int msgid;

    // ftok to generate unique key
    key = ftok("progfile", 65);

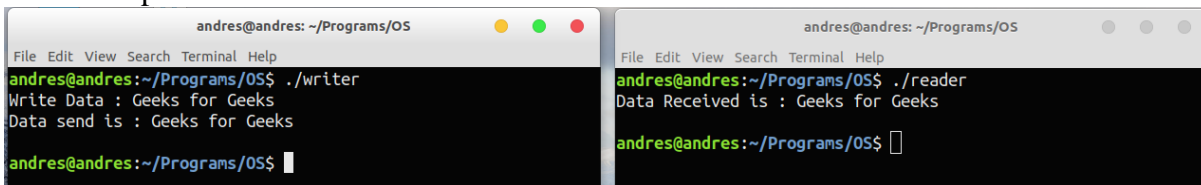
    // msgget creates a message queue
    // and returns identifier
    msgid = msgget(key, 0666 | IPC_CREAT);

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

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

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

    return 0;
}
Output:
```



```
andres@andres: ~/Programs/OS
File Edit View Search Terminal Help
andres@andres:~/Programs/OS$ ./writer
Write Data : Geeks for Geeks
Data send is : Geeks for Geeks
andres@andres:~/Programs/OS$

andres@andres: ~/Programs/OS
File Edit View Search Terminal Help
andres@andres:~/Programs/OS$ ./reader
Data Received is : Geeks for Geeks
andres@andres:~/Programs/OS$
```

OBJECTIVE

d) Write a C program to illustrate the Shared Memory IPC mechanism

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_t key = ftok("shmfile",65);

    // shmget returns an identifier in shmid
    int shmid = shmget(key,1024,0666|IPC_CREAT);

    // shmat to attach to shared memory
    char *str = (char*) shmat(shmid,(void*)0,0);

    cout<<"Write Data : ";
    gets(str);

    printf("Data written in memory: %s\n",str);

    //detach from shared memory
    shmdt(str);

    return 0;
}
```

SHARED MEMORY FOR READER 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_t key = ftok("shmfile",65);

    // shmget returns an identifier in shmid
    int shmid = shmget(key,1024,0666|IPC_CREAT);

    // shmat to attach to shared memory
    char *str = (char*) shmat(shmid,(void*)0,0);

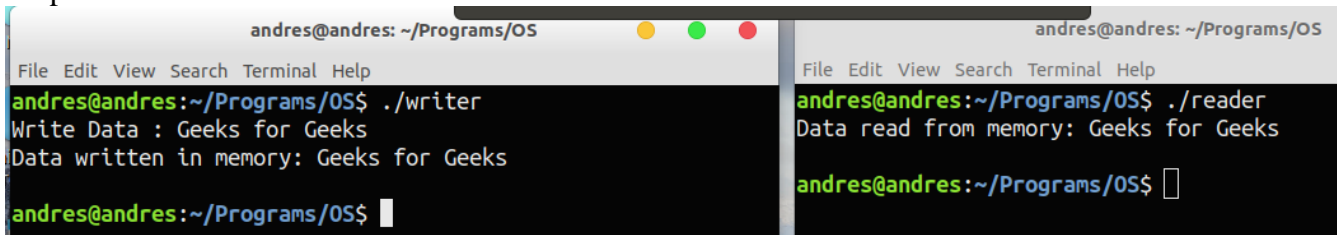
    printf("Data read from memory: %s\n",str);

    //detach from shared memory
    shmdt(str);

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

    return 0;
}
```

Output:



```
andres@andres: ~/Programs/OS
File Edit View Search Terminal Help
andres@andres:~/Programs/OS$ ./writer
Write Data : Geeks for Geeks
Data written in memory: Geeks for Geeks
andres@andres:~/Programs/OS$ █

andres@andres: ~/Programs/OS
File Edit View Search Terminal Help
andres@andres:~/Programs/OS$ ./reader
Data read from memory: Geeks for Geeks
andres@andres:~/Programs/OS$ █
```


VIVA QUESTIONS

1. What are local and global page replacements?
2. Define latency, transfer and seek time with respect to disk I/O.
3. Describe the Buddy system of memory allocation.
4. What is time-stamping?
5. How are the wait/signal operations for monitor different from those for semaphores?
6. In the context of memory management, what are placement and replacement algorithms?
7. In loading programs into memory, what is the difference between load-time dynamic linking and run-time dynamic linking?
8. What are demand-paging and pre-paging?
9. Paging a memory management function, while multiprogramming a processor management function, are the two interdependent?
10. What is page cannibalizing?
11. What has triggered the need for multitasking in PCs?
12. What are the four layers that Windows NT have in order to achieve independence?
13. Explain compaction.
14. What are page frames?
15. What are pages?
16. Differentiate between logical and physical address.
17. When does page fault error occur?
18. Explain thrashing.
19. What is the criteria for the best page replacement algorithm?
20. What is Belady's anomaly ?

EXPERIMENT -6

Write a C program to simulate the following techniques of memory management

- a) Paging b) Segmentation

9.1 OBJECTIVE

Write a C program to simulate paging technique of memory management.

9.2 DESCRIPTION

In computer operating systems, paging is one of the memory management schemes by which a computer stores and retrieves data from the secondary storage for use in main memory. In the paging memory-management scheme, the operating system retrieves data from secondary storage in 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 blocks of the same size called pages. When a process is to be executed, its pages are loaded into any available memory frames from their source.

AIM: To simulate paging technique of memory management.

PROGRAM

```
#include<stdio.h>

#include<conio.h>
main()

{

int ms, ps, nop, np, rempages, i, j, x, y, pa, offset; int s[10], fno[10][20];
clrscr();
printf("\nEnter the memory size -- "); scanf("%d",&ms);
printf("\nEnter the page size -- "); scanf("%d",&ps);
nop = ms/ps;
printf("\nThe no. of pages available in memory are -- %d ",nop);
printf("\nEnter number of processes -- "); scanf("%d",&np);
rempages = nop;
for(i=1;i<=np;i++)

{
printf("\nEnter no. of pages required for p[%d]-- ",i); scanf("%d",&s[i]);

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

```
printf("\nMemory is Full"); break;

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

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

}
printf("\nEnter Logical Address to find Physical Address ");
printf("\nEnter process no. and pagenumber and offset ");
scanf("%d %d %d",&x,&y, &offset);
if(x>np || y>=s[i] || offset>=ps)
printf("\nInvalid Process or Page Number or offset");

else

{

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

}

getch();
}
```

INPUT

```
Enter the memory size -- 1000
Enter the page size -- 100
The no. of pages available in memory are -- 10
Enter number of processes -- 3
Enter no. of pages required for p[1] --      4
Enter pagetable for p[1] ---   8      6      9      5
Enter no. of pages required for p[2] --      5
Enter pagetable for p[2] ---   1      4      5      7      3
Enter no. of pages required for p[3] --      5
```

OUTPUT

```
Memory is Full
Enter Logical Address to find Physical Address
Enter process no. and pagenumber and offset -- 2      3      60
The Physical Address is --      760
```

b) OBJECTIVE: To implement the memory management policy-segmentation

PROGRAM LOGIC:

1. Start the program.
2. Get the number of segments.
3. Get the base address and length for each segment.
4. Get the logical address.
5. Check whether the segment number is within the limit, if not display the error message.
6. Check whether the byte reference is within the limit, if not display the error message.
7. Calculate the physical memory and display it.
8. Stop the program

SOURCE CODE:

```
#include<stdio.h>
#include <conio.h>
#include<math.h>
int sost;
void gstinfo();
void ptladdr();
struct segtab
{ int sno;
  int baddr;
  int limit;
  int val[10];
}st[10];
void gstinfo()
{ int i,j;
printf("\n\tEnter the size of the segment table: ");
scanf("%d",&sost);
for(i=1;i<=sost;i++)
{
printf("\n\tEnter the information about segment: %d",i);
st[i].sno = i;
printf("\n\tEnter the base Address: ");
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",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\tLogical Address is: %d",swd);
    printf("\n\t\tMapped Physical address is: %d",paddr);
    printf("\n\t\tThe value is: %d",( st[s].val[disp] ) );
    }
    Else
    printf("\n\t\tLimit of segment %d is high\n\n",s);
    }
    else
    printf("\n\t\tInvalid Segment Address \n");
    }
    void main()
    { char ch;
    clrscr();
    gstinfo();
    do
    {
    ptladdr();
    printf("\n\t Do U want to Continue(Y/N)");
    fflush();
    scanf("%c",&ch);
    }while (ch == 'Y' || ch == 'y' );
    getch();
    }
```

INPUT AND OUTPUT:

```
Enter the size of the segment table: 3
Enter the information about segment: 1
Enter the base Address: 4
Enter the Limit: 5
Enter the 4 address Value: 11
Enter the 5 address Value: 12
Enter the 6 address Value: 13
Enter the 7 address Value: 14
Enter the 8 address Value: 15
Enter the information about segment: 2
Enter the base Address: 5
Enter the Limit: 4
Enter the 5 address Value: 21
Enter the 6 address Value: 31
Enter the 7 address Value: 41
```

Enter the 8 address Value: 51
Enter the information about segment: 3
Enter the base Address: 3
Enter the Limit: 4
Enter the 3 address Value: 31
Enter the 4 address Value: 41
Enter the 5 address Value: 41
Enter the 6 address Value: 51
SEGMENT TABLE SEG.NO BASE ADDRESS LIMIT
1 4 5
2 5 4
3 3 4
Enter the logical Address: 3
Logical Address is: 3
Mapped Physical address is: 3
The value is: 31
Do U want to Continue(Y/N)
SEGMENT TABLE SEG.NO BASE ADDRESS LIMIT
1 4 5
2 5 4
3 3 4
Enter the logical Address: 1
Logical Address is: 1
Mapped Physical address is: 4
The value is: 11 Do U want to Continue(Y/N)

VIVA QUESTIONS:

1. The page table contains _____
2. What is compaction?
3. Operating System maintains the page table for _____
4. Physical memory is broken into fixed-sized blocks called _____
5. Logical memory is broken into blocks of the same size called _____
6. Every address generated by the CPU is divided into two parts :
7. The _____ is used as an index into the page table.
8. The _____ table contains the base address of each page in physical memory.
9. The size of a page is typically :
10. With paging there is no _____ fragmentation.
11. The operating system maintains a _____ table that keeps track of how many frames have been allocated, how many are there, and how many are available.
12. Paging increases the _____ time.
13. Smaller page tables are implemented as a set of _____
14. The page table registers should be built with _____
15. For larger page tables, they are kept in main memory and a _____ points to the page table.
16. For every process there is a _____ table.
17. _____ address generated after compile time.
18. _____ address generated after runtime.
19. Address binding is done in _____ phases.
20. Define segmentation.