KG REDDY College of Engineering & Technology
AN AUTONOMOUS INSTITUTION

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

OPERATINYG SYSTEMS LABORATOR

Subject Code : KG23ACS223

Regulation : KGR23

Academic Year : 2025-2026

II B .Tech II Sem

COMPUTER SCIENCE AND ENGINEERING

KG REDDY COLLEGE OF ENGINEERING AND TECHNOLOGY

Affiliated o JNTUH, Chilkur,(V), Moinabad(M) R. R Dist, TS-501504

KG REDDY College of Engineering & Technology
AN AUTONOMOUS INSTITUTION

VISION AND MISSION OF THE INSTITUTION

VISION:

To become an institution which is internationally recognized for its holistic approach to engineering, innovative teaching and learning culture, research and entrepreneurial ecosystem, and sustainable social impact in the community.

MISSION:

• To offer undergraduate and post-graduate programs which are supported through industry relevant curriculum and innovative teaching and learning processes that would help students succeed in their professional careers.
• To provide faculty and students with an ecosystem that fosters innovation, research, entrepreneurship, and international exposure through strategic partnerships with governmentorganizations and collaboration with industries.
• To provide holistic learning environment to students which will contribute to their personal and professional growth and enable them to become leaders in their respective fields.
• To contribute to the development of the region by using our technological expertise to work with nearby communities and support them in their social and economic development.

KG REDDY College of Engineering & Technology
AN AUTONOMOUS INSTITUTION

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

VISION:

To be recognized as a department of excellence by stimulating a learning environment in which students and faculty will thrive and grow to achieve their professional, institutional and societal goals.

MISSION:

- To provide high quality technical education to students that will enable life-long learning and build expertise in advanced technologies in Computer Science and Engineering.
- To promote research and development by providing opportunities to solve complex engineering problems in collaboration with industry and government agencies.
- To encourage professional development of students that will inculcate ethical values and leadership skills through entrepreneurship while working with the community to address societalissues.

PROGRAM EDUCATIONAL OBJECTIVES

PEO 1: Graduates will provide solutions to difficult and challenging issues in their profession by applying computer science and engineering theory and principles.

PEO 2: Graduates have successful careers in computer science and engineering fields or will be able to successfully pursue advanced degrees.

PEO 3: Graduates will communicate effectively, work collaboratively and exhibit high levels of professionalism, moral and ethical responsibility.

PEO 4: Graduates will develop the ability to understand and analyze engineering issues in a broader perspective with ethical responsibility towards sustainable development.

KG REDDY College of Engineering & Technology
AN AUTONOMOUS INSTITUTION

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

PROGRAM OUTCOMES

PO I: Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
PO II: 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.
PO III: 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.
PO IV: 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.
PO V: 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.
PO VI: 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.
PO VII: 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.
PO VIII: Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
PO IX: Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
PO X: 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.
PO XI: 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.
PO XII: 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.

KG REDDY College of Engineering & Technology
AN AUTONOMOUS INSTITUTION

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

PROGRAM SPECIFIC OUTCOMES

PSO1: The Computer Science and Engineering graduates are able to analyze, design, develop, test and apply management principles, mathematical foundations in the development of intelligent systems with computational solutions, make them to expert in designing the secure application and hardware prototype
PSO2: The graduating student will be analyze the contemporary research issues in different areas of computer science & engineering and explore research gaps, analyze and carry out research in the specialized/emerging areas.
PSO3: Develop their skills to solve problems in the broad area of programming concepts and appraise environmental and social issues with ethics and manage different projects in multi- disciplinary field to conducive in cultivating skills for successful career, entrepreneurship and higher studies.

KG REDDY College of Engineering & Technology
AN AUTONOMOUS INSTITUTION

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Course Code: KG23ACS223 L T P C

B. Tech. II Year II – Semester 0 0 2 1

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

Course Objectives: The objectives of this course for the student are to:

1.Introduce operating system concepts

2.Introduce CPU scheduling and deadlock concept

3.Introduce the issues to be considered in the design and development of the operating System

4.Introduce basic Unix commands, system call interface for process management, inter communication and I/O in Unix

5.Introduce File System Interface and Operations.

Course Outcomes: After completion of this course, the students will be able to

CO1:Will be able to control access to a computer and the files that may be shared

CO2:Demonstrate knowledge of the components of computers and their respective roles in computing.

CO3:Ability to recognize and resolve user problems with standard operating environments

CO4:Gain practical knowledge of how programming languages, perating systems, and Architectures interact and how to use each effectively.

CO5:File System Interface and Its Operations

List of Experiments:

I.Write C programs to 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 (open, read, write, close, fcntl, seek, stat, opendir, readdir)

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

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

5.Write C programs to illustrate the following IPC mechanisms

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

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

a) Paging b) Segmentation

7.Write C programs to simulate Page replacement policies

a) FCFS b) LRU c) Optimal

EXPERIMENT- 1

CPU SCHEDULING ALGORITHMS

A. *FIRST COME FIRST SERVE:*

AIM: To write a c program to simulate the CPU scheduling algorithm First ComeFirst Serve (FCFS)

DESCRIPTION:

To calculate the average waiting time using the FCFS algorithm first the waiting time of the first process is kept zero and the waiting time of the second process is the burst time of the first process and the waiting time of the third process is the sum of the burst time of the first and the second process and so on. After calculating all the waiting times the average waiting time is calculated as the average of all the waiting times.FCFS mainly says first come first serve the algorithm which came first will be served first.

ALGORITHM:

Step 1: Start the process

Step 2: Accept the number of processes in the ready Queue

Step 3: For each process in the ready Q, assign the process name and the burst time

Step 4: Set the waiting of the first process as _0'and its burst time as its turnaround time

Step 5: for each process in the Ready Q calculate Waiting time (n) = waiting time (n-1) +Burst time (n-Turnaround time (n) = waiting time (n)+Burst time(n)

Step 6: Calculate Average waiting time = Total waiting Time Number of Process

Average Turn around time = Total Turnaround Time / Number of process

Step7: stop

PROGRAM:

```
#include<stdio.h> #include<conio.h> main()
{
int bt[20], wt[20], tat[20], i, n;float wtavg, tatavg;
clrscr();
printf("\n Enter the number of processes –");
scanf("%d", &n);
for(i=0; i<n; i++)
{
printf("\n Enter Burst Time for Process %d -- ", i);
scanf("%d", &bt[i]);
}
wt[0] = wtavg = 0;
tat[0] = tatavg = bt[0];
for(i=1;i<n;i++)
{
wt[i] = wt[i-1] +bt[i-1];
tat[i] = tat[i-1] +bt[i];
wtavg = wtavg + wt[i];
tatavg = tatavg + tat[i];
}
printf("\t PROCESS \tBURST TIME \t WAITING TIME\t TURNAROUND TIME\n");
for(i=0;i<n;i++)
printf("\n\t P%d \t\t %d \t\t %d \t\t %d", i, bt[i], wt[i], tat[i]); printf("\nAverage
Waiting Time -- %f", wtavg/n);
printf("\nAverage Turnaround Time -- %f", tatavg/n);
```

getch();

INPUT:

Enter the number of processes --3 Enter Burst Time for Process 0 --24

Enter Burst Time for Process 1 --3 Enter Burst Time for Process 2 --3

OUTPUT:

PROCESS	BURST TIME	WAITING TIME	TURNAROUND TIME
P0	24	0	24
P1	3	24	27
P2	3	27	30

Average Waiting Time à 17.000000

Average Turnaround Time à27.000000

B .SHORTEST JOB FIRST:

AIM: To write a program to stimulate the CPU scheduling algorithm Shortest job first(Non- Preemption)

DESCRIPTION:

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

ALGORITHM:

Step 1: Start the process

Step 2: Accept the number of processes in the ready Queue

Step 3: For each process in the ready Q, assign the process id and accept the CPUburst time

Step 4: Start the Ready Q according.

Step 5: Set the waiting time of the first process as _0'and its turnaround time as its bursttime.

Step 6: Sort the processes names based on their Burt time

Step 7: For each process in the ready queue, calculate

Waiting time(n)= waiting time (n-1) + Burst time (n-1)

Turnaround time (n)= waiting time(n)+Burst time(n)

Step 8: Calculate

Average waiting time = Total waiting Time / Number of process

Average Turnaround time = Total Turnaround Time / Number ofprocess

Step 9: Stop the process

PROGRAM:

```
#include<stdio.h> #include<conio.h> main()
{
int p[20], bt[20], wt[20], tat[20], i, k, n, temp;
float wtavg,tatavg;
clrscr();
printf("\nEnter the number of processes -- ");
scanf("%d", &n);
for(i=0;i<n;i++)
{
p[i]=i;
printf("Enter Burst Time for Process %d -- ", i); scanf("%d", &bt[i]);
}
for(i=0;i<n;i++) for(k=i+1;k<n;k++)if(bt[i]>bt[k])
{
temp=bt[i];bt[i]=bt[k]; bt[k]=temp;
temp=p[i];
p[i]=p[k];
p[k]=temp;
}
wt[0] = wtavg = 0;
tat[0] = tatavg = bt[0];
for(i=1;i<n;i++)
{
wt[i] = wt[i-1] +bt[i-1];
tat[i] = tat[i-1] +bt[i];
```

```
wtavg = wtavg + wt[i];
tatavg = tatavg + tat[i];
}
printf("\n\t PROCESS \tBURST TIME \t WAITING TIME\t TURNAROUND TIME\n");
for(i=0;i<n;i++)
printf("\n\t P%d \t\t %d \t\t %d \t\t %d", p[i], bt[i], wt[i], tat[i]);
printf("\nAverage Waiting Time -- %f", wtavg/n);
printf("\nAverage Turnaround Time -- %f", tatavg/n);
getch();}
```

INPUT:

Enter the number of processes – 4

Enter Burst Time for Process 0 – 6

Enter Burst Time for Process 1 – 8

Enter Burst Time for Process 2 – 7

Enter Burst Time for Process 3 -- 3

OUTPUT:

PROCESS	BURST TIME	WAITING TIME	TURN AROUND TIME
P3	3	0	3
P0	6	3	9
P2	7	9	16
P1	8	16	24

Average Waiting Time à 7.000000

Average Turnaround Time à 13.000000

C.ROUND ROBIN:

AIM: To simulate the CPU scheduling algorithm round-robin.

DESCRIPTION:

To aim is to calculate the average waiting time. There will be a time slice, each process should be executed within that time-slice and if not it will go to the waiting state so first check whether the burst time is less than the time-slice. If it is less than it assign the waiting time to the sum of the total times. If it is greater than the burst-time then subtract the time slot from the actual burst time and increment it by time-slot and the loop continues until all the processes are completed.

ALGORITHM:

Step 1: Start the process

Step 2: Accept the number of processes in the ready Queue and time quantum (or) Step

3: For each process in the ready Q, assign the process id and accept the CPU bursttime

Step timeslice

4: Calculate the no. of time slices for each process where,

No. of timeslice for process (n) = burst time process (n)/time slice

Step 5: If the burst time is less than the time slice then the **no. of time slices =1.**

Step 6: Consider the ready queue is a circular Q, calculate

a) Waiting time for process (n) = waiting time of process(n-1)+ burst time ofprocess(n-1) + the time difference in getting the CPU from process(n-1)

b) Turnaround time for process(n) = waiting time of process(n) + burst time ofprocess(n)+ the time difference in getting CPU from process(n).

Step 7: Calculate

c) Average waiting time = Total waiting Time / Number of process

Average Turnaround time = Total Turnaround Time / Number of process

Step 8: Stop the process

PROGRAM:

```
#include<stdio.h> main()
{
int i,j,n,bu[10],wa[10],tat[10],t,ct[10],max;
float awt=0,att=0,temp=0;
clrscr();
printf("Enter the no of processes -- ");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("\nEnter Burst Time for process %d -- ", i+1);
scanf("%d",&bu[i]);
ct[i]=bu[i];
}
printf("\nEnter the size of time slice -- ");
scanf("%d",&t);
max=bu[0];
for(i=1;i<n;i++) if(max<bu[i]) max=bu[i];
for(j=0;j<(max/t)+1;j++) for(i=0;i<n;i++) if(bu[i]!=0)if(bu[i]<=t)
{
tat[i]=temp+bu[i];
temp=temp+bu[i];
bu[i]=0;
}
else { bu[i]=bu[i]-t; temp=temp+t;
```

```
}
for(i=0;i<n;i++)
{
wa[i]=tat[i]-ct[i];
att+=tat[i]; awt+=wa[i];
}
printf("\nThe Average Turnaround time is -- %f",att/n);
printf("\nThe Average Waiting time is -- %f ",awt/n);
printf("\n\tPROCESS\t BURST TIME \t WAITING TIME\tTURNAROUND TIME\n");
for(i=0;i<n;i++)
printf("\t%d \t %d \t\t %d \t\t %d \n",i+1,ct[i],wa[i],tat[i]);
getch();}
```

INPUT:

Enter the no of processes – 3

Enter Burst Time for process 1 – 24

Enter Burst Time for process 2 – 3

Enter Burst Time for process 3 – 3

Enter the size of time slice – 3

OUTPUT:

PROCESS	BURST TIME	WAITING TIME	TURNAROUND TIME
1	**24**	**6**	**30**
2	**3**	**4**	**7**
3	**3**	**7**	**10**

The Average Turnaround time is – 15.666667

The Average Waiting time is - 5.666667

D. PRIORITY:

AIM: To write a c program to simulate the CPU scheduling priority algorithm.

DESCRIPTION:

To calculate the average waiting time in the priority algorithm, sort the burst times according to their priorities and then calculate the average waiting time of the processes. The waiting time of each process is obtained by summing up the burst times of all the previous processes.

ALGORITHM:

Step 1: Start the process

Step 2: Accept the number of processes in the ready Queue

Step 3: For each process in the ready Q, assign the process id and accept the CPU bursttime

Step 4: Sort the ready queue according to the priority number.

Step 5: Set the waiting of the first process as _0'and its burst time as its turnaround time

Step 6: Arrange the processes based on process priority

Step 7: For each process in the Ready Q

calculate

Step 8: for each process in the Ready Q

calculate

i. Waiting time(n)= waiting time (n-1) +Burst time (n-1)

ii Turnaround time (n)= waiting time(n)+Burst time(n)

Step 9: Calculate Average waiting time = Total waiting Time / Number of process

I Average Turnaround time = Total Turnaround Time / Number of process Printthe results in an order.

Step10: Stop

PROGRAM:

```
#include<stdio.h> main()
{
int p[20],bt[20],pri[20], wt[20],tat[20],i, k, n, temp;
float wtavg,tatavg;
clrscr();
printf("Enter the number of processes --- ");
scanf("%d",&n);
for(i=0;i<n;i++)
{
p[i] = i;
printf("Enter the Burst Time & Priority of Process %d --- ",i);
scanf("%d %d",&bt[i], &pri[i]);
}
for(i=0;i<n;i++) for(k=i+1;k<n;k++) if(pri[i] > pri[k])
{
temp=p[i]; p[i]=p[k];
p[k]=temp; temp=bt[i];
bt[i]=bt[k]; bt[k]=temp;
temp=pri[i];
pri[i]=pri[k];
pri[k]=temp;
}
wtavg = wt[0] = 0;
tatavg = tat[0] = bt[0];
for(i=1;i<n;i++)
```

```
{
wt[i] = wt[i-1] + bt[i-1];
tat[i] = tat[i-1] + bt[i];
wtavg = wtavg + wt[i];
tatavg = tatavg + tat[i];
}
printf("\nPROCESS\t\tPRIORITY\tBURST TIME\tWAITING TIME\tTURN
AROUNDTIME");
for(i=0;i<n;i++)
printf("\n%d \t\t %d \t\t %d \t\t %d \t\t %d ",p[i],pri[i],bt[i],wt[i],tat[i]);
printf("\nAverage Waiting Time is --- %f",wtavg/n);
printf("\nAverage Turnaround Time is --- %f",tatavg/n);
getch();
}
```

INPUT:

```
Enter the number of processes -- 5
Enter the Burst Time & Priority of Process 0 --- 10        3
Enter the Burst Time & Priority of Process 1 --- 1         1
```

```
Enter the Burst Time & Priority of Process 2 --- 2        4
Enter the Burst Time & Priority of Process 3 --- 1        5
Enter the Burst Time & Priority of Process 4 --- 5        2
```

OUTPUT:

PROCESS	PRIORITY	BURST	WAITING TIME	TURNA ROUND TIME
1	1	1	0	1
4	2	5	1	6
0	3	10	6	16
2	4	2	16	18
3	5	1	18	19

Average Waiting Time is à 8.200000

Average Turnaround Time is à 12.000000

EXPERIMENT- 02:

AIM: To write C Programs using the following system calls of UNIX operating system fork, exec, getpid,

exit, wait, close, stat, opendir, readdir**.**

1 PROGRAM FOR SYSTEM CALLS OF UNIX OPERATING SYSTEMS (OPENDIR, READDIR, CLOSEDIR)

ALGORITHM:

STEP 1: Start the program.

STEP 2: Create struct dirent.

STEP 3: declare the variable buff and pointer dptr.

STEP 4: Get the directory name.

STEP 5: Open the directory.

STEP 6: Read the contents in directory and print it.

STEP 7: Close the directory

PROGRAM:

```
#include<stdio.h>
#include<dirent.h>
struct dirent *dptr;
int main(int argc, char *argv[])
{
char buff[100];
DIR *dirp;
printf("\n\n ENTER DIRECTORY NAME");
scanf("%s", buff);
if((dirp=opendir(buff))==NULL)
```

```
{
printf("The given directory does not exist");
exit(1);
}
while(dptr=readdir(dirp))
{
printf("%s\n",dptr->d_name);
}
closedir(dirp);
}
```

2. PROGRAM FOR SYSTEM CALLS OF UNIX OPERATING SYSTEM (fork, getpid, exit)

ALGORITHM:

STEP 1: Start the program.

STEP 2: Declare the variables pid,pid1,pid2.

STEP 3: Call fork() system call to create process.

STEP 4: If pid==-1, exit.

STEP 5: Ifpid!=-1 , get the process id using getpid().

STEP 6: Print the process id**.**

STEP 7: Stop the program

PROGRAM:

```
#include<stdio.h>
#include<unistd.h> main()
{
int pid,pid1,pid2;
pid=fork(); if(pid==-1)
{
```

```
printf("ERROR IN PROCESS CREATION \n");
exit(1);
}
if(pid!=0)
{
pid1=getpid();
printf("\n the parent process ID is %d\n", pid1);
}
else
{
pid2=getpid();
printf("\n the child process ID is %d\n", pid2);
}
}
```

1. Simulate UNIX commands like cp, ls, grep.

AIM: To write C programs for simulation of cp unix commands

ALGORITHM:

STEP1: Start the program

STEP 2:Declare the variables ch, *fp, sc=0

STEP3: Open the file in read mode

STEP 4: Get the character

STEP 5: If ch== " " then increment sc value by one

STEP 6: Print no of spaces

STEP 7: Close the file

PROGRAM:

```
#include<fcntl.h>
#include<unistd.h>
#include<stdio.h>
main(int argc,char *argv[])
{
FILE *fp; char ch; int sc=0;
fp=fopen(argv[1],"r");
if(fp==NULL)
printf("unable to open a file",argv[1]);
else
{
while(!feof(fp))
{
ch=fgetc(fp);
if(ch==' ')
sc++;
}
printf("no of spaces %d",sc);
printf("\n");
fclose(fp);
}
}
```

2. PROGRAM FOR SIMULATION OF LS UNIX COMMANDS

AIM: Program For Simulation of "ls" Unix Commands

ALGORTIHM:

STEP1 : Start the program

STEP2 : Open the directory with directory object dp

STEP3 : Read the directory content and print it.

STEP4: Close the directory.

PROGRAM:

```
#include<stdio.h>
#include<dirent.h>
main(int argc, char **argv)
{
DIR *dp;
struct dirent *link;
dp=opendir(argv[1]);
printf("\n contents of the directory %s are \n", argv[1]);
while((link=readdir(dp))!=0)
printf("%s",link->d_name);
closedir(dp);
}
```

EXPERIMENT – 03

A. DEADLOCK PREVENTION

AIM: To implement deadlock prevention technique

DESCRIPTION: (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 whet her the allocation no feachre sources will leave the system in safe state. If it will the reso urces are allocation; otherwise the process must wait until some other process release the resources.

Number of process, m-number of resource types.

Available: Available[j]=k, k –instance of resource type Rjisavailable. Max:Ifmax[i,j]=k,Pimay request At mostkinstances resource Rj.

Allocation: If Allocation [i, j]=k, Pi allocated to k instances of resource Rj Need: If Need[I, j]=k, Pi may need k more instances of resource type Rj,

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

ALGORITHM:

Start the program.

Get the values of resources and processes.Get the avail value.

After allocation find the need value.

1.Check whether its possible to allocate.

2.If it is possible then the system is in safe state.

3.After allocation find the need value.

4.Check whether its possible to allocate.

5.If it is possible then the system is in safe state.

6.Else system is not in safety state.

7.Stop the processStop the process.

SOURCE CODE :

```
#include<stdio.h>
#include<conio.h>
void main()
{
char job[10][10];
int time[10],avail,tem[10],temp[10];
int safe[10];
int ind=1,i,j,q,n,t;
clrscr();
printf("Enter no of jobs: ");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("Enter name and time: ");
```

```
scanf("%s%d",&job[i],&time[i]);
}
printf("Enter the available resources:");
scanf("%d",&avail);
for(i=0;i<n;i++)
{
temp[i]=time[i];
tem[i]=i;
}
for(i=0;i<n;i++) for(j=i+1;j<n;j++)
{
if(temp[i]>temp[j])
{
t=temp[i];
temp[i]=temp[j];
temp[j]=t;
t=tem[i];
tem[i]=tem[j];
tem[j]=t;
}
}
for(i=0;i<n;i++)
{
q=tem[i];
if(time[q]<=avail)

safe[ind]=tem[i];
avail=avail-tem[q];
printf("%s",job[safe[ind]]);
```

```
ind++;
}
Else
{
printf("No safe sequence\n");
}
}
printf("Safe sequence is:");
for(i=1;i<ind; i++)
printf("%s %d\n",job[safe[i]],time[safe[i]]); getch();
```

OUTPUT:

```
Enter no of jobs:4
Enter name and time: A 1
Enter name and time: B 4
Enter name and time: C 2
Enter name and time: D 3
Enter the available resources: 20
Safe sequence is: A 1, C 2, D 3, B 4.
```

B. DEAD LOCK AVOIDANCE

AIM: To Simulate bankers algorithm for Dead Lock Avoidance (Banker's Algorithm)

DESCRIPTION:

Deadlock is a situation where in two or more competing actions are waiting f or the other To finish and thus neither ever does. 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

ALGORITHM:

a. Start the program.
b. Get the values of resources and processes.
c. Get the avail value.
d. After allocation find the need value.
e. Check whether its possible to allocate.
f. If it is possible then the system is in safe state.
g. Else system is not in safety state.
h. If the new request comes then check that the system is in safety or not, if we allow the request.
i. Stop the program.

SOURCE CODE :

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
int alloc[10][10],max[10][10];
int avail[10],work[10],total[10];
int i,j,k,n,need[10][10];
int m;
int count=0,c=0;
char finish[10];
clrscr();
printf("Enter the no. of processes and resources:");
scanf("%d%d",&n,&m);
for(i=0;i<=n;i++)
finish[i]='n';
printf("Enter the claim matrix:\n");
for(i=0;i<n;i++)
for(j=0;j<m;j++)
scanf("%d",&max[i][j]);
printf("Enter the allocation matrix:\n");
for(i=0;i<n;i++)
for(j=0;j<m;j++)
scanf("%d",&alloc[i][j]);
printf("Resource vector:");
for(i=0;i<m;i++)
scanf("%d",&total[i]);
for(i=0;i<m;i++)
avail[i]=0;
for(i=0;i<n;i++)
for(j=0;j<m;j++)
avail[j]+=alloc[i][j];
for(i=0;i<m;i++)
work[i]=avail[i];
for(j=0;j<m;j++)
work[j]=total[j]-work[j];
for(i=0;i<n;i++)
for(j=0;j<m;j++)
need[i][j]=max[i][j]-alloc[i][j];
A: for(i=0;i<n;i++)
{
c=0;
for(j=0;j<m;j++)
if((need[i][j]<=work[j])&&(finish[i]=='n'))
c++;
if(c==m)
{
printf("All the resources can be allocated to Process %d", i+1);
printf("\n\nAvailable resources are:");
for(k=0;k<m;k++)
{
```

```
work[k]+=alloc[i][k];
printf("%4d",work[k]);
}
printf("\n");
finish[i]='y';
printf("\nProcess %d executed?:%c \n",i+1,finish[i]); count++;
}
}
if(count!=n) goto A;
else
printf("\n System is in safe mode"); printf("\n The given state is safe state"); getch();
}
```

OUTPUT

Enter the no. of processes and resources: 4 3 Enter the claim matrix:
3 2 2
6 1 3
3 1 4
4 2 2
Enter the allocation matrix:
1 0 0
6 1 2
2 1 1
0 0 2
Resource vector:9 3 6
All the resources can be allocated to Process 2 Available resources are: 6 2 3 Process 2 executed?:y
All the resources can be allocated to Process 3 Available resources are: 8 3 4 Process 3 executed?:y
All the resources can be allocated to Process 4 Available resources are: 8 3 6 Process 4 executed?:y
All the resources can be allocated to Process 1 Available resources are: 9 3 6 Process 1 executed?:y System is in safe mode The given state is safe state

EXPERIMENT- 04

AIM: To Write a C program to simulate producer-consumer problem using semaphores.

DESCRIPTION:

Producer consumer problem is a synchronization problem. There is a fixed size buffer where The producer produces it emsand that is consumed by a consumer process. One solution to the producer- consumer problem uses shared memory. To allow producer and consumer processes to run concurrently, there must be available a buffer of items that can be filled by the producer and emptied by the consumer. This buffer will reside in a region of memory that is shared by the producer and consumer processes. The producer and consumer must be synchronized, so that the consumer does not try to consume an item that has not yet been produced.

PROGRAM:

PROGRAM

```
#include<stdio.h>
void main()
{
int buffer[10], bufsize, in, out, produce, consume, choice=0;
in = 0;
out = 0;
bufsize = 10;
while(choice !=3)
{
printf("\n1. Produce\t 2.Consume\t 3.Exit");
printf("\n Enter your choice: ");
scanf("%d",&choice);
switch(choice)
{
case 1: if((in+1)%bufsize==out)
            printf("\nBuffer is Full");
        else
        {
         printf("\nEnter the value: ");
         scanf("%d", &produce);
         buffer[in] = produce;
         in = (in+1)%bufsize;
        }
         break;
 case 2: if(in == out)
            printf("\nBuffer is Empty");
         else
         {
          consume = buffer[out];
          printf("\nThe consumed value is %d", consume);
          out = (out+1)%bufsize;
         }
         break;
        }
    }
}
```

EXPERIMENT- 05:

Write C programs to illustrate the following IPC mechanisms

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

Pipes

a) AIM: To implement the concept of interprocess communication using pipes using c program**.**

ALGORITHM:

create the pipe and create the process.

get the input in the main process and pass the output to the child process using pipe.

perform the operation given in the child process and print the output.

stop the program**.**

```
#include<stdio.h>
#include<unistd.h>
#include<string.h>
main()
{
 int p1[2],p2[2],p3[2],p4[2];
 int i,j=0,k=0,l=0;
 char r[10],s[10],t[10],u[10];
 printf("\t PROCESS 1.ENTER THE STRING");
 scanf("%s",r);
 pipe(p1);
 pipe(p2);
 write(p1[1],r,sizeof(r));
 write(p2[1],r,sizeof(r));
 int a=fork();
 if(a==0)
 {
  printf("\n\t PROCESS 2:it splits the given string\n");
  read(p1[0],r,sizeof(r));
  int n=strlen(r);
  for(i=0;i<n/2;i++)
  {
    s[i]=r[i];
  }
  for(i=n/2;i<=n;i++)
  {
   t[j++]=r[i];
  }
  pipe(p3);
  pipe(p4);
  write(p3[1],s,sizeof(s));
  write(p4[1],t,sizeof(t));
  int b=fork();
  if(b==0)
  {
```

□

```
printf("p4 %d\t",getpid());
printf("p2 %d\n",getppid());
read(p3[0],s,sizeof(s));
printf("\t PROCESS 4:sub string \t %s \t",s);
printf("no of char=%d \n",strlen(s));
}
else
{
int c=fork(); if(c==0)
{
printf("p5 %d\t",getpid());
printf("p2 %d\n",getppid());
read(p4[0],t,sizeof(t));
printf("\t PROCESS 5:sub string \t %s \t",t); printf("no of char=%d \n",strlen(t));
}
else
{
wait();
printf("p2 %d\t",getpid()); printf("p1 %d\n",getppid());
}  }}
else
{
wait();
int d=fork();
if(d==0)
{
printf("p3 %d\t",getpid()); printf("p1 %d\n",getppid());
read(p2[0],r,sizeof(r));
for(i=strlen(r)-1;i>=0;i--)
{
u[l++]=r[i];
}
for(i=0;i<strlen(r);i++)
{
if(u[i]==r[i]) k++;
else continue;
}
if(k==strlen(r))
printf("\t PROCESS 3: the given string is palindrome\n"); else
printf("\t PROCESS 3: the given string is not palindrome\n");
}
```

```
else
{
printf("p1 %d\t",getpid()); printf("kernal %d\t\n",getppid());
}
}}
```

OUTPUT:

Process 1: enter the string ARUN Process 2: it splits the string

P4 8137 p2 8136

Process 3: sub string a r no of char=2 P5 8138 p2 8136

Process 4; substring u n no of char=2 P2 8136 p1 8132

P3 8139 p1 8132

Process 3: the given string is not palindrom

Process 1: enter the string MADAM Process 2: it splits the string

P4 8137 p2 8136

Process 4: sub string m a no of char=2 P5 8138 p2 8136

Process 5: sub string dam no of char=3 P2 8136 p1 8132

P3 8139 p1 8132

Process 3: the given string is palindrome

FIFOs :

```
#include < stdio.h >
int main()
{
   int incomingStream[] = {4 , 1 , 2 , 4 , 5};
   int pageFaults = 0;
   int frames = 3, m, n, s, pages;
   pages = sizeof(incomingStream)/sizeof(incomingStream[0]);
   printf(" Incoming \ t Frame 1 \ t Frame 2 \ t Frame 3 ");
   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(incomingStream[m] == temp[n])
```

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

Output:

```
Incoming Frame 1 Frame 2 Frame 3
4               4   -         -
1               4   1         -
2               4   1         2
4               4   1         2
5               5   1         2
Total Page Faults: 4
```

c). Message queues:

```
/* Filename: msgq_send.c */
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#define PERMS 0644
struct my_msgbuf {
   long mtype;
   char mtext[200];
};
int main(void)
{
   struct my_msgbuf buf;
   int msqid;
   int len;
   key_t key;
   system("touch msgq.txt");
   if ((key = ftok("msgq.txt", 'B')) == -1)
  {
      perror("ftok");
      exit(1);
   }
 if ((msqid = msgget(key, PERMS | IPC_CREAT)) == -1)
   {
      perror("msgget");
      exit(1);
   }
   printf("message queue: ready to send messages.\n");
   printf("Enter lines of text, ^D to quit:\n");
   buf.mtype = 1; /* we don't really care in this case */
      while(fgets(buf.mtext, sizeof buf.mtext, stdin) != NULL)
  {
      len = strlen(buf.mtext);
      /* remove newline at end, if it exists */
      if (buf.mtext[len-1] == '\n') buf.mtext[len-1] = '\0';
      if (msgsnd(msqid, &buf, len+1, 0) == -1) /* +1 for '\0' */
      perror("msgsnd");
   }
   strcpy(buf.mtext, "end");
   len = strlen(buf.mtext);
   if (msgsnd(msqid, &buf, len+1, 0) == -1) /* +1 for '\0' */
   perror("msgsnd");
```

```
  int complete;
  char buf[BUF_SIZE];
};
int fill_buffer(char * bufptr, int size);

int main(int argc, char *argv[]) { int shmid,
  numtimes;
  struct shmseg *shmp; char *bufptr;
  int spaceavailable;
  shmid = shmget(SHM_KEY, sizeof(struct shmseg), 0644|IPC_CREAT); if (shmid
  == -1) {
    perror("Shared memory"); return 1;
  }

  // Attach to the segment to get a pointer to it. shmp =
  shmat(shmid, NULL, 0);
```

```
  if (shmp == (void *) -1) {
    perror("Shared memory attach");
    return 1;
  }

  /* Transfer blocks of data from buffer to shared memory
  */ bufptr = shmp->buf;
  spaceavailable = BUF_SIZE;
  for (numtimes = 0; numtimes < 5;
    numtimes++) { shmp->cnt =
    fill_buffer(bufptr, spaceavailable); shmp-
    >complete = 0;
    printf("Writing Process: Shared Memory Write: Wrote %d bytes\n", shmp-
    >cnt); bufptr = shmp->buf;
    spaceavailable =
    BUF_SIZE; sleep(3);
  }
  printf("Writing Process: Wrote %d times\n",
  numtimes); shmp->complete = 1;

  if (shmdt(shmp) == -
    1) {
    perror("shmdt");
    return 1;
  }

  if (shmctl(shmid, IPC_RMID, 0) == -1) {
    perror("shmctl");
    return 1;
  }
  printf("Writing Process:
  Complete\n"); return 0;
}

int fill_buffer(char * bufptr, int size) {
  static char ch = 'A';
  int filled_count;

  //printf("size is %d\n",
  size); memset(bufptr, ch,
  size - 1); bufptr[size-1] =
  '\0';
  if (ch >
  122) ch
  = 65;
  if ( (ch >= 65) && (ch <= 122)
    ) { if ( (ch >= 91) && (ch <=
```

```
}
```

Output:

```
Writing Process: Shared Memory Write: Wrote 1023 bytes Writing Process: Shared Memory Write: Wrote 1023 bytes Writing Process: Shared Memory Write: Wrote 1023 bytes Writing Process: Shared Memory Write: Wrote 1023 bytes Writing Process: Shared Memory Write: Wrote 1023 bytes Writing Process: Wrote 5 times
Writing Process: Complete
```

Shared memory for Reading Process: /* Filename: shm_read.c */

```
#include<stdio.h>
#include<sys/ipc.h>
#include<sys/shm.h>
#include<sys/types.h>
#include<string.h>
#include<errno.h>
#include<stdlib.h>

#define BUF_SIZE 1024
#define SHM_KEY 0x1234

struct shmseg {
   int cnt;
   int complete;
   char buf[BUF_SIZE];
};
int main(int argc, char *argv[])
{
   int shmid;
   struct shmseg *shmp;
   shmid = shmget(SHM_KEY, sizeof(struct shmseg), 0644|IPC_CREAT);
   if (shmid == -1)
{
      perror("Shared memory");
      return 1;
   }
   shmp = shmat(shmid, NULL, 0);
   if (shmp == (void *) -1) {
      perror("Shared memory attach");
      return 1;
   }
      while (shmp->complete != 1)
{
      printf("segment contains : \n\"%s\"\n", shmp->buf);
      if (shmp->cnt == -1)
{
         perror("read");
         return 1;
      }
      printf("Reading Process: Shared Memory: Read %d bytes\n", shmp->cnt);
      sleep(3);
```

```
}
printf("Reading Process: Reading Done, Detaching Shared Memory\n"); if (shmdt(shmp) == -1)
{
perror("shmdt"); return 1;
}
printf("Reading Process: Complete\n"); return 0;
}
```

Output:

```
segment contains :
"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA"
Reading Process: Shared Memory: Read 1023 bytes segment contains :
"BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
```

BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
Reading Process: Shared Memory: Read 1023 bytes segment contains :
"CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
Reading Process: Shared Memory: Read 1023 bytes segment contains :
"DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD
DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD”

EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE
EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE
EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE
EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE
EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE
EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE
EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE
EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE
EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE
EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE
EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE
EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE
EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE
EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE
EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE
EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE
EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE
EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE
EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE”
Reading Process: Shared Memory:Read 1023 bytes
Reading Process: Reading Done,
Detaching Shared Memory
Reading Process: Complete

EXPERIMENT- 06

MEMORY MANAGEMENT TECHNIQUES

MEMORY MANAGEMENT WITH FIXED PARTITIONING TECHNIQUE (MFT)

AIM: To implement and simulate the MFT algorithm.

DESCRIPTION:

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

ALGORITHM:

Step1: Start the process.
Step2: Declare variables.
Step3: Enter total memory size ms.
Step4: Allocate memory for os.
Ms=ms-os
Step5: Read the no partition to be divided n Partition size=ms/n.
Step6: Read the process no and process size.
Step 7: If process size is less than partition size allot also block the process. While allocatingupdate memory wastage-external fragmentation.

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

Step 8: Print the results

SOURCE CODE :

```
#include<stdio.h> #include<conio.h> main()
{
int ms, bs, nob, ef, n, mp[10], tif=0; int i,p=0;
clrscr();
printf("Enter the total memory available (in Bytes) -- "); scanf("%d",&ms);
printf("Enter the block size (in Bytes) -- "); scanf("%d", &bs);
nob=ms/bs; ef=ms - nob*bs;
printf("\nEnter the number of processes -- "); scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("Enter memory required for process %d (in Bytes)-- ",i+1); scanf("%d",&mp[i]);
}
printf("\n No. of Blocks available in memory--%d",nob);
printf("\n\nPROCESS\t MEMORYREQUIRED\t ALLOCATED\t INTERNAL
FRAGMENTATION");
for(i=0;i<n && p<nob;i++)
{
printf("\n %d\t\t%d",i+1,mp[i]); if(mp[i] > bs)
printf("\t\tNO\t\t---"); else
{
printf("\t\tYES\t%d",bs-mp[i]); tif = tif + bs-mp[i];
p++;
}
}
if(i<n)
printf("\nMemory is Full, Remaining Processes cannot be accommodated"); printf("\n\nTotal
Internal Fragmentation is %d",tif);
printf("\nTotal External Fragmentation is %d",ef); getch();
}
```

INPUT

Enter the total memory available (in Bytes) -- 1000 Enter the block size (in Bytes)-- 300
Enter the number of processes – 5
Enter memory required for process 1 (in Bytes) -- 275
Enter memory required for process 2 (in Bytes) -- 400
Enter memory required for process 3 (in Bytes) -- 290
Enter memory required for process 4 (in Bytes) -- 293
Enter memory required for process 5 (in Bytes) -- 100
No. of Blocks available in memory 3

OUTPUT:

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

Memory is Full, Remaining Processes cannot be accommodated Total
Internal Fragmentation is 42
Total External Fragmentation is 100

C. MEMORY VARIABLE PARTIONING TYPE (MVT):

AIM: To write a program to simulate the MVT algorithm

ALGORITHM:

Step1: Start the process
Step2: Declare variables
Step3: Enter total memory size ms.
Step4: Allocate memory for os.
Ms=ms-os
Step5: Read the no partition to be divided n Partition size=ms/n.
Step6: Read the process no and process size.
Step 7: If process size is less than partition size allot alse blocked the process. While allocatingupdate memory wastage-external fragmentation.

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

}

}

Step 8: Print the results

Step 9: Stop the process

SOURCE CODE:

```
#include<stdio.h>
#include<conio.h> main()
{
int ms,mp[10],i, temp,n=0;
char ch = 'y';
clrscr();
printf("\nEnter the total memory available (in Bytes)-- ");
scanf("%d",&ms);
temp=ms;
for(i=0;ch=='y';i++,n++)            {
printf("\n Enter memory required for process %d (in Bytes) -- ",i+1);
scanf("%d",&mp[i]);
if(mp[i]<=temp)
{
printf("\nMemory is allocated for Process %d ",i+1);
temp = temp - mp[i];
}
else
{
printf("\nMemory is Full");
break;
}
printf("\nDo you want to continue(y/n) -- ");
scanf(" %c", &ch);
}
printf("\n\nTotal Memory Available -- %d", ms);
printf("\n\n\tPROCESS\t\t MEMORY ALLOCATED ");
for(i=0;i<n;i++)
printf("\n \t%d\t\t%d",i+1,mp[i]);
printf("\n\nTotal Memory Allocated is %d",ms-temp);
printf("\nTotal External Fragmentation is %d",temp);
getch();
}
```

OUTPUT:

Enter the total memory available (in Bytes) – 1000 Enter memory required for process 1 (in Bytes) – 400 Memory is allocated for Process 1
Do you want to continue(y/n) -- y
Enter memory required for process 2 (in Bytes) -- 275 Memory is allocated for Process 2
Do you want to continue(y/n) – y
Enter memory required for process 3 (in Bytes) – 550 Memory is Full
Total Memory Available – 1000 PROCESS MEMORY ALLOCATED
1 400
2 275
Total Memory Allocated is 675 Total External Fragmentation is 325

EXPERIMENT- 07

Write C programs to simulate Page replacement policies

a) FCFS b) LRU c) Optimal

a) FIRST COME FIRST SERVE:

AIM: To write a c program to simulate the CPU scheduling algorithm First ComeFirst Serve (FCFS)

DESCRIPTION:

To calculate the average waiting time using the FCFS algorithm first the waiting time of the first process is kept zero and the waiting time of the second process is the burst time of the first process and the waiting time of the third process is the sum of the burst time of the first and the second process and so on. After calculating all the waiting times the average waiting time is calculated as the average of all the waiting times.FCFS mainly says first come first serve the algorithm which came first will be served first.

ALGORITHM:

Step 1: Start the process

Step 2: Accept the number of processes in the ready Queue

Step 3: For each process in the ready Q, assign the process name and the burst time

Step 4: Set the waiting of the first process as _0'and its burst time as its turnaround time

Step 5: for each process in the Ready Q calculate Waiting time (n) = waiting time (n-1) +Burst time (n-Turnaround time (n) = waiting time (n)+Burst time(n)

Step 6: Calculate Average waiting time = Total waiting Time Number of Process

Average Turn around time = Total Turnaround Time / Number of process

Step7: stop

PROGRAM:

```
#include<stdio.h> #include<conio.h> main()
{
int bt[20], wt[20], tat[20], i, n;float wtavg, tatavg;
clrscr();
printf("\n Enter the number of processes –");
scanf("%d", &n);
for(i=0; i<n; i++)
{
printf("\n Enter Burst Time for Process %d -- ", i);
scanf("%d", &bt[i]);
}
wt[0] = wtavg = 0;
tat[0] = tatavg = bt[0];
for(i=1;i<n;i++)
{
wt[i] = wt[i-1] +bt[i-1];
tat[i] = tat[i-1] +bt[i];
wtavg = wtavg + wt[i];
tatavg = tatavg + tat[i];
}
printf("\t PROCESS \tBURST TIME \t WAITING TIME\t TURNAROUND TIME\n");
for(i=0;i<n;i++)
printf("\n\t P%d \t\t %d \t\t %d \t\t %d", i, bt[i], wt[i], tat[i]); printf("\nAverage
Waiting Time -- %f", wtavg/n);
printf("\nAverage Turnaround Time -- %f", tatavg/n);
```

```
getch();
```

INPUT:

Enter the number of processes --3 Enter Burst Time for Process 0 --24

Enter Burst Time for Process 1 --3 Enter Burst Time for Process 2 --3

OUTPUT:

PROCESS	BURST TIME	WAITING TIME	TURNAROUND TIME
P0	24	0	24
P1	3	24	27
P2	3	27	30

Average Waiting Time à 17.000000 Average Turnaround Time à27.000000

B) LRU (Least Recently Used) Page Replacement:

Description:

LRU replaces the page that has not been used for the longest period of time

Code: LRU Page Replacement

```
#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];
int counter = 0, time[10], flag1, flag2, 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(int i = 0; i < no_of_pages; ++i)
   scanf("%d", &pages[i]);

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

printf("\nPage Replacement Process (LRU):\n");

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

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

```
}

if(flag1 == 0) {
    for(int 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("\nPage %d -> ", pages[i]);
for(int j = 0; j < no_of_frames; ++j) {
    if(frames[j] != -1)
        printf("%d ", frames[j]);
    else
```

```
            printf("- ");
        }
    }

    printf("\n\nTotal Page Faults = %d\n", faults);
    return 0;
}
```

OUTPUT:

Enter number of frames: 3

Enter number of pages: 12

Enter reference string: 7 0 1 2 0 3 0 4 2 3 0 3

Page Replacement Process (LRU):

Page 7 -> 7 - -

Page 0 -> 7 0 -

Page 1 -> 7 0 1

Page 2 -> 2 0 1

Page 0 -> 2 0 1

Page 3 -> 2 0 3

Page 0 -> 2 0 3

Page 4 -> 4 0 3

Page 2 -> 4 2 3

Page 3 -> 4 2 3

Page 0 -> 0 2 3

Page 3 -> 0 2 3

Total Page Faults = 8

C) Optimal Page Replacement

Description:

Optimal replacement replaces the page that will not be used for the longest periodinthefuture.

Code: Optimal Page Replacement

```
#include <stdio.h>

int findOptimal(int pages[], int frames[], int n, int index, int frame_count) {
    int pos = -1, farthest = index;
    for(int i = 0; i < frame_count; i++) {
        int j;
        for(j = index; j < n; j++) {
            if(frames[i] == pages[j]) {
                if(j > farthest) {
                    farthest = j;
                    pos = i;
                }
                break;
            }
        }
        if(j == n)
            return i; // Page not used again
```

```
    }
    if(pos == -1)
        return 0;
    else
        return pos;
}

int main() {
    int n, frame_count, pages[30], frames[10];
    int faults = 0, flag1, flag2;

    printf("Enter number of pages: ");
    scanf("%d", &n);

    printf("Enter reference string: ");
    for(int i = 0; i < n; i++)
        scanf("%d", &pages[i]);

    printf("Enter number of frames: ");
    scanf("%d", &frame_count);

    for(int i = 0; i < frame_count; i++)
        frames[i] = -1;

    printf("\nPage Replacement Process (Optimal):\n");
```

```
for(int i = 0; i < n; i++) {
    flag1 = flag2 = 0;

    for(int j = 0; j < frame_count; j++) {
        if(frames[j] == pages[i]) {
            flag1 = flag2 = 1;
            break;
        }
    }

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

    if(flag2 == 0) {
        int pos = findOptimal(pages, frames, n, i + 1, frame_count);
        frames[pos] = pages[i];
        faults++;
    }
```

```
        printf("\nPage %d -> ", pages[i]);
        for(int j = 0; j < frame_count; j++) {
            if(frames[j] != -1)
                printf("%d ", frames[j]);
            else
                printf("- ");
        }
    }

    printf("\n\nTotal Page Faults = %d\n", faults);
    return 0;
}
```

OUTPUT:

Page Replacement Process (Optimal):

Page 7 -> 7 - -

Page 0 -> 7 0 -

Page 1 -> 7 0 1

Page 2 -> 2 0 1

Page 0 -> 2 0 1

Page 3 -> 2 0 3

Page 0 -> 2 0 3

Page 4 -> 4 0 3

Page 2 -> 4 0 2

Total Page Faults = 6