SVR ENGINEERING COLLEGE

AYYALURUMETTA (V), NANDYAL, KURNOOL DT.
ANDHRA PRADESH – 518502

2018-19

LABORATORY MANUAL

OF

Operating Systems Laboratory

(15A05510)

(R-15 REGULATION)

Prepared by

Mr. A.HARI PRASAD REDDY
Asst. Professor

For

B.Tech III YEAR – I SEM. (CSE)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

SVR ENGINEERING COLLEGE

(AFFILIATED TO JNTUA ANANTHAPURAM- AICITE-INDIA)
AYYALURUMETTA (V), NANDYAL, KURNOOL DT.
ANDHRA PRADESH – 518502

LAB MANUAL CONTENT

Operating Systems Laboratory

(15A05510)

Institute Vision & Mission, Department Vision & Mission

1. PO, PEO& PSO Statements.
2. List of Experiments
3. CO-PO Attainment
4. Experiment Code and Outputs

1. Institute Vision & Mission, Department Vision & Mission

Institute Vision:

To produce Competent Engineering Graduates & Managers with a strong base of Technical & Managerial Knowledge and the Complementary Skills needed to be Successful Professional Engineers & Managers.

Institute Mission:

To fulfill the vision by imparting Quality Technical & Management Education to the Aspiring Students, by creating Effective Teaching/Learning Environment and providing State – of the – Art Infrastructure and Resources.

Department Vision:

To produce Industry ready Software Engineers to meet the challenges of 21st Century.

Department Mission:

- Impart core knowledge and necessary skills in Computer Science and Engineering through innovative teaching and learning methodology.
- Inculcate critical thinking, ethics, lifelong learning and creativity needed for industry and society.
- Cultivate the students with all-round competencies, for career, higher education and self-employability.

2. PO, PEO& PSO Statements

PROGRAMME OUTCOMES (POs)

PO-1: Engineering knowledge - Apply the knowledge of mathematics, science, engineering fundamentals of Computer Science& Engineering to solve complex real-life engineering problems related to CSE.

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

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

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

PO-5: Modern tool usage - Select/Create and apply appropriate techniques, resources and modern engineering and IT tools and technologies for rapidly changing computing needs, including prediction and modeling to complex engineering activities, with an understanding of the limitations.

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

PO-7: Environment and Sustainability - Understand the impact of the CSE professional engineering solutions in societal and environmental contexts and demonstrate the knowledge of, and need for sustainable development.

PO-8: Ethics - Apply ethical principles and commit to professional ethics and responsibilities and norms of the relevant engineering practices.

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

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

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

PO-12: Life-long learning - Recognize the need for and have the preparation and ability to engage in independent and life-long learning in the broadcast context of technological changes.

Program Educational Objectives (PEOs):

PEO 1:Graduates will be prepared for analyzing, designing, developing and testing the software solutions and products with creativity and sustainability.

PEO 2: Graduates will be skilled in the use of modern tools for critical problem solvingand analyzing industrial and societal requirements.

PEO 3:Graduates will be prepared with managerial and leadership skills for career and starting up own firms.

Program Specific Outcomes (PSOs):

PSO 1: Develop creative solutions by adapting emerging technologies / tools for real time applications.

PSO 2: Apply the acquired knowledge to develop software solutions and innovative mobile apps for various automation applications

2.1 Subject Time Table

SVR ENGINEERING COLLEGE::NANDYAL DEPARTMENT OF CSE								
Mr. A.HARI PRASAD REDDY						**III-I**		
Day/ Time	**9:30 AM**	**10:20 AM**	**11:30 AM**	**12:20 PM-**		**02:00 PM**	**02:50 PM**	**03:40 PM**
	10:20 AM	**11:10AM**	**12:20 PM**	**01:10 PM**		**02:50 PM**	**03:40 PM**	**04:30 PM**
MON					**LUNCH BREAK**			
TUE								
WED								
THU								
FRI						OS LAB		
SAT								

INDEX

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY ANANTAPUR

B. Tech III-I Sem. (CSE)

OPERATING SYSTEMS LABORATORY

Course Objectives:

- To understand the design aspects of operating system
- To solve various synchronization problems

Course out comes:

- Ensure the development of applied skills in operating systems elated a eas.
- Able to write software routines modules or implementing vari us concepts of operatingsystem.

1. Simulate the following CPU scheduling algorithms

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

2. Simulate all file allocation strategies a) Sequential b) Indexed c) Linked

3. Simulate MVT and MFT

4. Simulate all File Organization Techniques a) Single level directory b) Two level c) Hierarchical d) DAG

5. Simulate Bankers Algorithm for Dead Lock Avoidance

6. Simulate Bankers Algorithm for Dead Lock Prevention

7. Simulate all page replacement algorithms a) FIFO b) LRU c) LFU Etc. …

8. Simulate Paging Technique of memory management

9. Control the number of ports opened by the operating system with a) Semaphore b) monitors

10. Simulate how parent and child processes use shared memory and address space

11. Simulate sleeping barber problem

12. Simulate dining philosopher„s problem
13. Simulate producer and consumer problem using threads (use java)
14. Simulate little„s formula to predict next burst time of a process for SJF schedulingAlgorithm.
15. Develop a code to detect a cycle in wait-for graph
16. Develop a code to convert virtual address to physical address
17. Simulate how operating system allocates frame to process
18. Simulate the prediction of deadlock in operating system when allThe processes announce their resource requirement in advance.

SVR ENGINEERING COLLEGE			
Department:	**COMPUTER SCIENCE & ENGINEERING**		
Course Outcome Attainment - Internal Assessments			
Name of the faculty :	A.HARI PRASAD REDDY / SIRAJ HUSSAIN	Academic Year:	2018-2019
Branch & Section:	COMPUTER SCIENCE & ENGINEERING	Exam:	15A05510
Course:	Operating Systems Laboratory	Semester:	III-I SEM

Course Outcomes	**Internal Lab**		**Internal Lab**	**University Exam**
15A05510.1	3		3	3
15A05510.2	3		3	3
15A05510.3	3		3	3
15A05510.4	3		3	3
15A05510.5	3		3	3

Course Outcomes		Attainment Level
15A05510.1	Ensure the development of applied skills in operating systems related areas.	**3**
15A05510.2	Able to write software routines modules or implementing various concepts of operating system.	**3**
15A05510.3	Understand process life cycle and able to tell process status.	**3**
15A05510.4	Understand the concepts of process scheduling, synchronization and its implementation	**3**
15A05510.5	Determine the prevention, avoidance, detection, recovery mechanism of deadlock	**3**
	Average Attainment	**3**

Overall Course Attainment	**3**

SVR ENGINEERING COLLEGE

DEPARTMENT	COMPUTER SCIENCE & ENGINEERING		

PROGRAM OUTCOME ATTAINMENT

Name of Faculty:	A.HARI PRASAD REDDY / SIRAJ HUSSAIN	Academic Year	2018-2019
Branch & Section:	COMPUTER SCIENCE & ENGINEERING	SUB CODE:	15A05510
Course:	Operating Systems Laboratory	Semester:	III-I

COURSE OUTCOME ATTAINMENT

Course outcome attainment	**Internal lab**		**Internal lab**	**External lab**
15A05510.1	3		3	3
15A05510.2	3		3	3
15A05510.3	3		3	3
15A05510.4	3		3	3
15A05510.5	3		3	3

COURSE OUTCOMES AND PROGRAM OUTCOMES MAPPING

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2
15A05510.1	3	3	2	2	1	2			2		1	2	3	3

15A05510.2	3	2	1	1		1		1				1	3	3
15A05510.3	3	3	2	2	2	1	1			1			3	3
15A05510.4	3		1	2		2		3	2		2	2	3	3
15A05510.5	3		2	2		2		2		1		2	3	3
AVERAGE	3.0	2.7	1.6	1.8	1.5	1.6	1.0	2.0	2.0	1.0	1.5	1.8	3.0	3.0

PO-ATTAINMENT

		PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2
INTERNAL	15A05510.1	9	9	6	6	3	6			6		3	6	9	9
	15A05510.2	9	6	3	3		3		3				3	9	9
	15A05510.3	9	9	6	6	6	3	3			3			9	9
	15A05510.4	9		3	6		6		9	6		6	6	9	9
	15A05510.5	9		6	6		6		6		3		6	9	9
UNIVERSITY	15A05510.1	9	9	6	6	3	6			6		3	6	9	9
	15A05510.2	9	6	3	3		3		3				3	9	9
	15A05510.3	9	9	6	6	6	3	3			3			9	9
	15A05510.4	9		3	6		6		9	6		6	6	9	9
	15A05510.5	9		6	6		6		6		3		6	9	9
OVERALL	15A05510.1	3	3	3	3	3	3			3		3	3	3	3
	15A05510.2	3	3	3	3		3		3				3	3	3
	15A05510.3	3	3	3	3	3	3	3			3			3	3
	15A05510.4	3		3	3		3		3	3		3	3	3	3

	15A05510.5	3		3	3		3		3		3		3	3	3
Attainment		**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**	**3**

A.HARI PRASAD REDDY

Head of the Department

EXPERIMENT-1.1

1.1.1 OBJECTIVE:

Simulate the Round Robin CPU scheduling algorithm

1.1.2 PROGRAM LOGIC:

```
Step 1: Start the process
Step 2: Accept the number of processes in the ready Queue and
time quantum (or) time slice Step 3: For each process in the
ready Q, assign the process id and accept the CPU burst time
Step 4: Calculate the no. of time slices for each process where
      No. of time slice 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 Waiting time for
process(n) = waiting time of process(n-1)+ burst time of process(n-1 ) + the time
difference in getting the CPU from process(n-1)
(a)    Turnaround time for process(n) = waiting time of process(n) + burst time
of process(n)+ the time difference in getting CPU from process(n).
Step 7: Calculate
```

(a) Average waiting time = Total waiting Time / Number of process

(b) Average Turnaround time = Total Turnaround Time / Number of process

Step 8: Stop the process

1.1.3 PROGRAM:

```
#include<stdio
.h>
#include<conio    int
.h> void main()   ts,need[10],wt[10],tat[10],i
{                 ,j,n,n1; int
                  bt[10],flag[10],ttat=0,twt=0
                  ;
                  float
                  awt,atat;
                  clrscr();
                  printf("\n    Round    Robbin
                  Scheduling   \n");   printf("
                  Enter the number of processes
                  "); scanf("%d",&n);
                  n1=n;
                  printf(" Enter the
                  time slice ");
                  scanf("%d",&ts);
                  for(i=1;i<=n;i++)
                  {
                        printf("Enter the burst time
                        for process %d ",i);
                        scanf("%d",&bt[i]);
                        need[i]=bt[i];
                    }
                    for(i=1;i<=n;i++)
                    {
                        flag[i
```

```
      ]=1;
      wt[i]=
      0;
}
while(n!=0)
{
      for(i=1;i<=n;i++)
      {
            if(need[i]>=ts)
            {
               for(j=1;j<=n;j++)

                  {
                        if((i!=j)&&(flag[i]==1)&&(need[j
                              ]!=0)) wt[j]+=ts;
                  }
                  need[i]-
                  =ts;
                  if(need[i
                  ]==0)
                  {
                        flag[i
                        ]=0;
                  }     n--;
            }
            else
            {
                  for(j=1;j<=n;j++)
                  {
                        if((i!=j)&&(flag[i]==1)&&(need[j
                        ]!=0)) wt[j]+=need[i];
                  }
                  need[i
                  ]=0; n-
                  -;
                  flag[i]=0;
            }
      }
}
for(i=1;i<=n1;i++)
{
      tat[i]=wt[i
      ]+bt[i];
      twt=twt+wt[
      i];
      ttat=ttat+t
      at[i];
}
awt=(float)twt/n1;
atat=(float)ttat/n1;
printf("process bt \t
wt \t tat \n");
for(i=1;i<=n1;i++)
printf("%d \t %d \t %d \t %d
\n",i,bt[i],wt[i],tat[i]);
printf("avgwt=%f \n\n",awt);
printf("avgtat=%f",atat);
```

```
                    getch();
}
```

1.1.4 INPUT & OUTPUT: Round Robbin Scheduling Enter the number of processes 5Enter the time slice 7

```
Enter the burst time for process 1 87 Enter the burst time for process 2 54 Enter the burst time for process 3 65 Enter the burst time for process 4 34 Enter the burst time for process 5 34
process         bt   wt   tat
1               87   165  252
2               54   167  221
3               65   172  237
4               34   133  167
5               34   139  173
avgwt=155.199997
avgtat=210.000000
0
```

EXPERIMENT-1.2

1.2.1 OBJECTIVE

Simulate SJF CPU scheduling algorithm

1.2.2 PROGRAM LOGIC

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 burst time
Step 4: Start the Ready Q according the shortest Burst time by sorting according to lowest to highest burst time. Step 5: Set the waiting time of the first process as „0" and its turnaround time as its burst time.
Step 6: For each process in the ready queue, calculate

(a) Waiting time for process(n)= waiting time of process (n-1) + Burst time of process(n-1)
(b) Turnaround time for Process(n)= waiting time of Process(n)+ Burst time for process(n)

Step 6: Calculate

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

Step 7: Stop the process

1.2.3 PROGRM:

```
#include<stdio
.h>
#include<conio
.h> void main()
{
                int i,j,temp,n,k,h;
                int b[10],w[10],t[10],br[10],pr[10];
                char
                p[10];
                temp=0;
                clrscr();
                printf("enter the number of
                processes"); scanf("%d",&n);
                for(i=0;i<n;i++)
                {
                        p[i]=0;
                        b[i]=0;
                        w[i]=0;
                        t[i]=0;
                        br[i]=0;
                        pr[i]=0;
                  }
                  for(i=0;i<n;i++)
                  {
                        printf("Enter the process
                        name %d \n",i);
                        scanf("%s",&p[i]);
                        printf("Enter the burst time of
                        process %d \n",i);
                        scanf("%d",&b[i]);
                  }
```

```
                    printf("\n
                    ");
                    for(i=0;i<
                    n;i++)
                    {
                         t[i]=p[i];
                    }

                    for(i=0;i<n;i++)
                    {
                         for(j=0;j<(n-i-1);j++)
                         {
                               if(b[j]>b[j+1])
                               {
                                     temp=b[j
                                     ];
                                     b[j]=b[j
                                     +1];
                                     b[j+1]=te
                                     mp;
                                     temp=p[j
                                     ];
                                     p[j]=p[j
                                     +1];
                                     p[j+1]=te
                                     mp;
                               }
                         }
                    }
                    temp=0;
                    for(i=0;i<
                    n;i++)
                    {
                         pr[i]=temp;
                         temp=temp+b
                         [i];
                    }
                    for(i=0;i<n;i++)
                    {          for(j=0;j<n;j++)
                         {
                               if(t[i]==p[j])
                               {
                                   br[i]=b[j];
                                   w[i]=pr[j];
                               }
                         }
                    }
printf("\nprocess\tburst time\twaiting time
\tTurnaround time\n");
printf("====================================\n"
);
                    for(i=0;i<n;i++)
                    {
                         printf("%c\t ",t[i]);
                         printf("%d\t\t   ",br[i]);
                         printf("%d\t\t   ",w[i]);
                         printf("%d\t"
                         ,br[i]);
                         printf("\n");
```

```
                    }
                    k=0,h=0;
                    for(i=0;i<n;i++)
                    {
k=k+w[i];h=h+br[i];
                    }
                    printf("Total waiting
                    time is %d\n",k); i=k/n;
                    printf("Average waiting time
                    is %d\n",i); printf("Total
                    TurnAround Time is %d\n",h);
                    printf("Average waiting time
                    is %d\n",h/n); getch();
}
```

1.2.4 INPUT AND OPUTPUT

```
enter the number of
processes  3 Enter the
process name 0
a
Enter the burst time of
process 0 24
Enter the process
name 1 b
Enter the burst time of
process 1 3
Enter the process
name 2 c
Enter the burst time of process 22

process             burst time          waiting time   Turnaround time
=========================================
  a                       24          5       24
  b                       3           2       3
  c                       2           0       2
  Total waiting time is 7
  Average waiting time is
  2
  Total Turn Around Time
  is 29
  Average waiting time is
  9
```

EXPERIMENT-1.3

1.3.1 OBJECTIVE

Program to perform FCFS CPU scheduling algorithm

1.3.2 PROGRAM LOGIC

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 burst time Step 4: Set the waiting of the first process as „0" and its burst time as its turn around time
Step 5: for each process in the Ready Q calculate

(c) Waiting time for process(n)= waiting time of process (n-1) + Burst time of process(n-1)
(d) Turnaround time for Process(n)= waiting time of Process(n)+ Burst time for process(n)

Step 6: Calculate

(e) Average waiting time = Total waiting Time / Number of process
(f) Average Turnaround time = Total Turnaround Time / Number of process

Step 7: Stop the process

1.3.3 PROGRAM :

```
#include<stdio
.h>
#include<conio
.h> void main()
{
                    int i,j,temp,n,k,h;
                    int
                    b[10],w[10
                    ]; char
                    p[10];
                    temp=0;
                    clrscr();
                    printf("enter the number
                    of processes");
                    scanf("%d",&n);
                    for(i=0;i<n;i++)

                    {
                         printf("Enter the process
                         name %d \n",i);
                         scanf("%s",&p[i]);
                         printf("Enter the burst time of
                         process %d \n",i);
                         scanf("%d",&b[i]);

                    }
                    for(i=0;i<n;i++)
                    {
                         w[i]=temp;
                         temp=temp+b
                         [i];
                    }
```

```
printf("\nprocess\tburst time\twaiting time
\tTurnaround time\n");
printf("====================================\n
");
                    for(i=0;i<n;i++)
                    {
                         printf("%c\t ",p[i]);
                         printf("%d\t\t   ",b[i]);
                         printf("%d\t\t   ",w[i]);
                         printf("%d\t
                         ",b[i]);
                         printf("\n")
                         ;
                    }
                    k=0,h=0;
                    for(i=0;i<n;i++)
                    {
                         k=k+w[i];
                         h=h+b[i];
                    }
                    printf("Total waiting
                    time is %d\n",k); i=k/n;
                    printf("Average waiting time
                    is %d\n",i); printf("Total
                    TurnAround Time is %d\n",h);
                    printf("Average waiting time
                    is %d\n",h/n); getch();
}
```

1.3.4 INPUT AND OUTPUT

```
enter the number of
processes3 Enter the
process name 0
a
Enter the burst time of
process 0 24
Enter the process
name 1 b
Enter the burst time of
process 1 3
Enter the process
name 2 c
Enter the burst time of
process 2 3

process burst time         waiting time    Turnaround time
========================================
  a                  24          0          24
  b                   3         24           3
  c                   3         27           3
  Total waiting time
  is 51
Average waiting time is 17
```

```
Total TurnAround Time
is 30 Average waiting
time is 10
```

EXPERIMENT-1.4

1.4.1 OBJECTIVE

Program to perform priority CPU scheduling algorithm

1.4.2 PROGRAM LOGIC

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 burst time 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 turn around time Step 6: For each process in the Ready Q calculate

(e) Waiting time for process(n)= waiting time of process (n-1) + Burst time of process(n-1)
(f) Turnaround time for Process(n)= waiting time of Process(n)+ Burst time for process(n)

Step 7: Calculate

(g) Average waiting time = Total waiting Time / Number of process
(h) Average Turnaround time = Total Turnaround Time / Number of process

Step 8: Stop the process

1.4.3 PROGRAM:

```
#include<stdio
.h>
#include<conio    int i,j,temp,n,k,h;
.h> void main()   int
{                 b[10],w[10],pr[10
                  ]; char p[10];
                  temp=0;
                  clrscr();
                  printf("enter the number of
                  processes"); scanf("%d",&n);
                  for(i=0;i<n;i++)
                  {
                        p[i]=0;
                        b[i]=0;
                        w[i]=0;
                    }
                    for(i=0;i<n;i++)
                    {
                         printf("Enter the process
                         name %d \n",i);
                         scanf("%s",&p[i]);
                         printf("Enter the burst time of
                         process %d \n",i);
```

```
        scanf("%d",&b[i]);
        printf("Enter the priority of
        process %d \n",i);
        scanf("%d",&pr[i]);
    }
    for(j=0;j<n;j++)
    {
```

```
                        for(i=0;i<n;i++)
                        {
                              if(pr[i]==j)
                              {
                                    w[i]=temp;
                                    temp=temp+b
                                    [i]; break;
                              }
                        }
                  }
printf("\nprocess\tburst time\twaiting time \tTurnaround time\tpriority\n");
                  printf("============================\n");
                  for(i=0;i<n;i++)
                  {
                        printf("%c\t ",p[i]);
                        printf("%d\t\t   ",b[i]);
                        printf("%d\t\t   ",w[i]);
                        printf("%d\t\t   ",b[i]);
                        printf("%d",pr[i]);
                        printf("\n");
                  }
                  k=0,h=0;
                  for(i=0;i<n;i++)
                  {
                        k=k+w[i];
                        h=h+b[i];
                  }
                  printf("Total waiting
                  time is %d\n",k); i=k/n;
                  printf("Average waiting time
                  is %d\n",i); printf("Total
                  TurnAround Time is %d\n",h);
                  printf("Average waiting time
                  is %d\n",h/n); getch();
}
```

1.4.4 INPUT AND OUTPUT

```
enter the number of
processes3 Enter the
process name 0
a
Enter the burst time of
process 0 24
Enter the priority of
process 0 2
Enter the process
name 1 b
Enter the burst time of
process 1 3
Enter the priority of
process 1 0
Enter the process
name 2 c
Enter the burst time of
process 2 3
Enter the priority of
```

```
process 2 1
```

```
process              burst time          waiting time   Turnaround time priority
=========================================
= a                   24        6      24
                      2
b       3        0       3       0
c                     3        3       3      1
Total waiting time is
9 Average waiting time
is 3 Total Turn Around
Time is 30 Average
waiting time is 10
```

1.5 PRE LAB QUESTIONS

1. Define scheduling
2. Discuss different types of scheduling
3. Differentiate FCFS,SJF
4. How Round Robin cpu scheduling algorithm works?
5. What is throughput and response time?

1.6 LAB ASSIGNMENT

1. In a c program, print the address of a variable and enter into a long loop. Start 3/4 processes of the same programand observe the printed address values. Try the experiment on different architectures/OSs. Are the addresses same?

2. Write a collection of processes p1, p2, p3 which execute with same PID.

1.7 POST LAB QUESTIONS

1. What is the relationship between threads and processes?
2. What is the function of the *ready queue*?
3. What is Dispatcher?
4. Why is round robin algorithm considered better than FCFS algorithm

EXPERIMENT-2.1

2.1.1 OBJECTIVE

Program to perform SEQUENTIAL file allocation technique

2.1.2 PROGRAM LOGIC

Step 1: Start the program.
Step 2: Get the number of memory partition and their sizes.
Step 3: Get the number of processes and values of block size for each process.
Step 4: First fit algorithm searches all the entire memory block until a hole which is big enough is encountered. It allocates that memory block for the requesting process.
Step 5: Best-fit algorithm searches the memory blocks for the smallest hole which can be allocated to requesting process and allocates if.
Step 6: Worst fit algorithm searches the memory blocks for the largest hole and allocates it to the process.
Step 7: Analyses all the three memory management techniques and display the best algorithm which utilizes the memory resources effectively and efficiently.
Step 8: Stop the program.

2.1.3 PROGRAM

```
#include<stdio
.h>
#include<conio
.h> void main()
{
       int
       a[100],s[10],l[10],n,
       m,c,i,j,k; clrscr();
       printf("Enter the total
       memory you want");
       scanf("%d",&m);
       printf("Enter the Total
       number of files");
       scanf("%d",&n);
       for(i=0;i<=m;i++)
          a[i]=-1;
  for(i=0;i<n;i++)
       {
             printf("Enter the starting block of file
             %d\n ",(i+1)); scanf("%d",&s[i]);
             printf("Enter the length of the
             file %d\n",(i+1));
             scanf("%d",&l[i]);
       }
       for(i=0;i<n;i++)
       {
             for(k=s[i],j=0;j<l[i];j++,k++)
             {
                   if(a[k]==-1)
                         c=0;
                                    f(c==1)

                                  }
                                  i
```

```
else
{
                        c
                        =
}                       1
                        ;
                        b
                        r
                        e
                        a
                        k
                        ;
                printf("This block was already filled by
                file %d\n",(a[k]+1)); if(c==0)
                {
                      for(k=s[i],j=0;j<l[i];j++,k++)
```

```
                        a[k]=i;
                printf("The file %d is filled from %d to %d\n",(i+1),s[i],(s[i]+l[i]-
            }   1));
        }
        getch()
        ;
}
```

2.1.4 INPUT AND OUTPUT

```
Enter the total memory
you want35 Enter the Total
number of files5 Enter
the starting block of
file 1
   1
Enter the length of
the file 1 2
Enter the starting
block of file 2 14
Enter the length of
the file 2 4
Enter the starting
block of file 3 19
Enter the length of
the file 3 3
Enter the starting
block of file 4 28
Enter the length of
the file 4 6
  Enter the starting
   block of file 5 6
Enter the length of
the file 5 5
The file 1 is filled from 1 to 2
The file 2 is filled from 14 to 17
The file 3 is filled from 19 to 21
The file 4 is filled from 28 to 33
The file 5 is filled from 6 to 10
```

EXPERIMENT-2.2

2.2.1 OBJECTIVE

Program to perform Linked file allocation mechanism.

2.2.2 PROGRAM LOGIC

```
Step 1: Create a queue to hold all pages in memory
Step 2: When the page is required replace the page
at the head of the queue Step 3: Now the new page is
inserted at the tail of the queue
Step 4: Create a stack
Step 5: When the page fault occurs replace page present
```

at the bottom of the stack Step 6: Stop the allocation.

2.2.3 PROGRAM

```
#include<conio.h>
```

```
#include<stdio
.h> void
main()
{
       int i,m,n,j,count;
       int a[100],s[10],e[10],in[10],inn[10][25];
       clrscr();
       printf("enter the
       total memory");
       scanf("%d",&m);
       for(i=0;i<m;i++)
             a[i]=-1;
      i=0;
       do{
             printf("\n to insert file press
             : 1 else press : 2"); printf("
             \n enter your choice: \n ");
             scanf("%d",&n);
             if(n==1)
             {
                   L1:
                         printf("Enter the starting block
                         of file %d",(i+1));
                         scanf("%d",&s[i]);
                         if(a[s[i]]==0)
                         {
                               printf("this block was
                               already filled \n"); goto
                               L1;
                         }
                         a[s[i]]=0;
                   L2:
                         printf("Enter the ending block
                         of file %d",(i+1));
                         scanf("%d",&e[i]);
                         if(a[e[i]]==0)
                         {
                               printf("this block was
                               already filled \n"); goto
                               L2;
                         }
                         a[e[i]]=0;
       printf("enter the number of intermediate
                   blocks of file %d",(i+1));
                   scanf("%d",&in[i]);
                   printf("enter the intermediate
                   blicks one by one \n");
                   for(j=0;j<in[i];j++)
                   {
                         scanf("%d",&inn[i][j]);
                         if(a[inn[i][j]]==0)
                         {
                          printf("this block was
                          already filled \n");
                          printf("enter another
                          block\n");
                          j--;
                          continue;
```

```
                        }
                        a[inn[i][j]]=0;
                }
                i+
        }       +;
}while(n=
=1);
count=i;
for(i=0;i<count;i++)
{
        printf("the file %d is linked as follows",(i+1));
```

```
            printf("%d",s[i]);
            for(j=0;j<in[i];j++
            )
            {
                        printf("-->%d",inn[i][j]);
            }
            printf("-->%d\n",e[i]);
      }
      getch();
}
```

2.2.4 INPUT AND OUTPUT

```
enter the total memory100
   to insert file press :
   1 else press : 2 enter
   your choice:
   1
Enter the starting block
of file 121 Enter the
ending block of file 171
enter the number of intermediate
blocks of file 14 enter the
intermediate blocks one by one
2
78
77
90
   to insert file press :
   1 else press : 2 enter
   your choice:
   1
Enter the starting block
of file 277 this block
was already filled
Enter the starting block
of file 221 this block
was already filled
Enter the starting block
of file 23 Enter the
ending block of file 26
enter the number of intermediate
blocks of file 22 enter the
intermediate blocks one by one
7980
To insert file press: 1
else press: 2 Enter your
choice:
   2
The file 1 is linked as follows21-->2--
>78-->77-->90-->71 The file 2 is linked
as follows3-->79-->80-->6
```

EXPERIMENT-2.3

2.3.1 OBJECTIVE

Program to simulate indexed file allocation algorithm

2.3.2 PROGRAM LOGIC

Step 1: Start.
Step 2: Let n be the size
of the buffer Step 3: check
if there are any producer
Step 4: if yes check whether the buffer is full

Step 5: If no the producer item is
stored in the buffer Step 6: If the
buffer is full the producer has to
wait
Step 7: Check there is any consumer. If yes check
whether the buffer is empty Step 8: If no, the consumer
consumes them from the buffer
Step 9: If the buffer is empty, the consumer has to wait.
Step 10: Repeat checking for the producer and
consumer till required Step 11: Terminate the
process.

2.3.3 PROGRAM

```
#include<stdio
.h>
#include<conio
.h> void main()
{
int
a[100],b[99],m,n,
i; int *p;
for(i=0;i<100;i+
+) a[i]=-1;
printf("enter the starting
address of file");
scanf("%d",&n);
printf("enter the number
of nodes");
scanf("%d",&m);
L1: for(i=0;i<m;i++)
{
           printf("enter index %d
           \n ",(i+1));
}          scanf("%d",&b[i]);
p=&b[0];
   a[n]=1
   ;
   printf("\n\nfile 1 starting address is: %d\n Your indexed locations are\n",n);

for(i=0;i<m;i++)
{
             if(a[b[i]
             ]==-1)
             {           a[b[i]]=1;
                         printf("%d\n
                         ",*p);
                         p=p+1;
             }
              else
              {
```

```
                    printf("same block already
                    allocated %d\n",b[i]);
getch()             for(i=0;i<m;i++)
;                         a[b[i]]=-1;
}                   printf("sorry, no vacancy
                    try again!"); goto L1;
                    }
            }
```

2.3.4 INPUT AND OUTPUT

```
enter the starting
address of file20 enter
the number of nodes5
enter index 1
   25
enter index 2
   10
enter index 3
   90
enter index 4
   77
enter index 5
   66
file    1    starting
address  is:  20 Your
indexed     locations
are 25
10
90
77
66
```

2.5 PRE LAB QUESTIONS

1. Define file.
2. How many types of file allocation methods are there? Briefly explain with
3. example.
4. What file allocation strategy is most appropriate for random access files?
5. In how many ways we can access a file? What are they?
6. What file access pattern is particularly suited to chained file allocation on disk?

2.6 LAB ASSIGNMENT

```
1.     Write a collection of sufficient no. of processes which carry out the
following different types of tasks independently:

Only
computation
Only printf s
Low computation, heavy console
output (printfs) Heavy File I/O
```

2.7 POST LAB QUESTIONS

1. File systems can support *sparse files,* what does this mean?
2. Give an example of a scenario that might benefit from a file system supporting an append-only access write.
3. Give a scenario where choosing a large file system block size might be a benefit
4. What file access pattern is particularly suited to chained file allocation on disk

EXPERIMENT – 3.1

3.1.1 OBJECTIVE

Simulate Multi programming with fixed number of tasks and Multi programming with variable number of tasks.

MFT

3.1.2 PROGRAM LOGIC

Step1: start the
process. Step2:
Declare variables.
Step3: Enter total
memory size.
Step4: Allocate memory
for os.
Step5: allocate total memory
to the pages. Step6: Display
the wastage of memory.
Step7: Stop the process.

3.1.3 PROGRAM

```
#include<stdio
.h>
#include<conio
.h> main()
{
       int
       ms,i,ps[20],n,size,p[20]
       ,s,intr=0; clrscr();
       printf("Enter size of
       memory:");
       scanf("%d",&ms);
       printf("Enter memory
       for OS:");
       scanf("%d",&s);
       ms-=s;
       printf("Enter no.of partitions
       to be divided:");
       scanf("%d",&n);
       size=ms/n;
       for(i=0;i<
       n;i++)
       {
             printf("Enter process and
             process size");
             scanf("%d%d",&p[i],&ps[i])
             ; if(ps[i]<=size)
             {
                   intr=intr+size-ps[i];
                   printf("process%d is allocated\n",p[i]);
             }
             el
             se    printf("process%d is blocked",p[i]);

       }
       printf("total
       fragmentation is
       %d",intr); getch();
}
```

3.1.4 INPUT AND OUTPUT

```
Enter total memory
size : 50 Enter
memory for OS
                  :1
0
Enter no.of partitions to
be divided:4 Enter size of
page  : 10
Enter   size   of
page  :  9  Enter
size of page : 9
Enter   size   of
```

```
page : 8
Internal Fragmentation is = 4
```

EXPERIMENT-3.2

3.2.1 OBJECTIVE

Simulate Multi programming with fixed number of tasks and Multi programming with variable number of tasks

MVT

3.2.2 PROGRAM LOGIC

Step1: start the process. Step2: Declare variables

Step3: Enter total memory size.
Step4: Allocate memory for os.
Step5: allocate total memory to the pages. Step6: Display the wastage of memory.
Step7: Stop the process.

3.2.3 PROGRAM

```
#include<stdio
.h>
#include<conio
.h> main()
{
       int
       i,m,n,tot,s[20];
       clrscr();
       printf("Enter total
       memory size:");
       scanf("%d",&tot);
       printf("Enter no.
       of pages:");
       scanf("%d",&n);
       printf("Enter memory
       for OS:");
       scanf("%d",&m);
       for(i=0;i<n;i++)
       {
             printf("Enter size of
             page%d:",i+1);
             scanf("%d",&s[i]);
       }
       tot=tot-m;
       for(i=0;i<
       n;i++)
       {
             if(tot>=s[i])
             {
                   printf("Allocate page
                   %d\n",i+1); tot=tot-
             }     s[i];
             el
             se
                   printf("process p%d is blocked\n",i+1);
       }
  printf("External
       Fragmentation
       is=%d",tot); getch();
}
```

3.2.4 INPUT AND OUTPUT

```
Enter total memory
size : 50 Enter no.of
pages              4
Enter memory for OS     10
Enter size of page10
Enter size of page9
Enter size of page9
Enter size of page10
External Fragmentation is = 2
```

3.5 PRE LAB QUESTIONS

1. What is segmentation?
2. Define fragmentation.
3. Explain different types of fragmentations.
4. What are the advantages of external fragmentation?

5. What is swapping?

3.6 LAB ASSIGNMENT

1. Implement file storage allocation techniques: Contiguous (using array)
2. Implementation of Contiguous allocation techniques:

(a) Worst-Fit
(b) Best-Fit
(c) First-Fit

3.7 POST LAB QUESTIONS

1. What is a drawback of MVT?
2. How many jobs can be run concurrently on MVT?
3. What is the problem of dynamic storage allocation problem?
4. What is Translation look aside buffer?
5. What is hash table?

EXPERIMENT-4.1

4.1.1 OBJECTIVE

Simulate Single level directory File Organization Technique

4.1.2 PROGRAM LOGIC

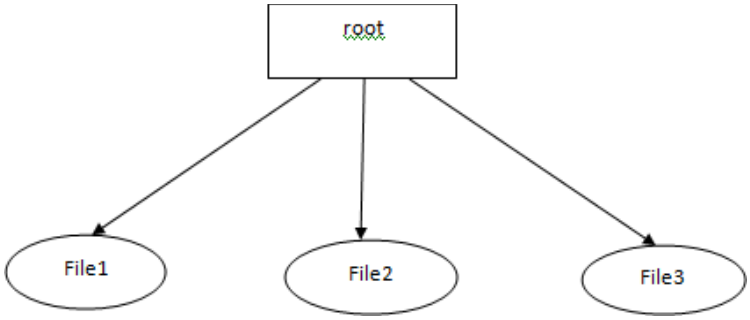
Step 1: Start the program.
Step 2: Get the name of the
directory. Step 3: get the
number of files.
Step 4: Get the name of the each file.
Step 5: Now each file is in the form
of filled circle Step 6: Every file is
connected with the given directory
Step 7: Display the connected gragh along with name using graphics

Step 8: Stop the program.

4.1.3 PROGRAM

```
#include<stdio.h>
#include<conio.h>
void main()
{
int
gd=DETECT,gm,count,i,j,mid,ci
r_x; char fname[10][20];
clrscr();
initgraph(&gd,&gm,"c:\\tc
\\bgi"); cleardevice();
setbkcolor(GREEN);
puts("Enter no of files do
u have?");
scanf("%d",&count);
for(i=0;i<count;i++)
{
cleardevice();
setbkcolor(GREEN);
printf("Enter file %d
name",i+1);
scanf("%s",fname[i]);
setfillstyle(1,MAGENTA);
mid=640/count;
cir_x=mid/3;
bar3d(270,100,370,150,0,
0);
settextstyle(2
,0,4);
settextjustify
(1,1);
outtextxy(320,125,"Root
Directory");
setcolor(BLUE);
for(j=0;j<=i;j++,cir_x+=mid
)
{
line(320,150,cir_x,250);
fillellipse(cir_x,250,30,30);
outtextxy(cir_x,250,fname[j]);
}
getch();
}
}
```

4.1.4 INPUT AND OUTPUT



EXPERIMENT-4.2

4.2.1 OBJECTIVE

Simulate Two level directory File Organization Technique

4.2.2 PROGRAM LOGIC

```
Step 1: Start the program.
Step 2: Get the name of the
directories. Step 3: get the
number of files.
Step 4: Get the name of the each file.
Step 5: Now each file is in the form of fill circle
Step 6: Every file is connected with respective directory
Step 7: Display the connected graph along with
name using graphics Step 8:  Stop the program.
```

4.2.3 PROGRAM

```
#include<stdio.h>
#include<graphics.h>
struct tree_element
{
char name[20];
int
x,y,ftype,lx,rx,nc,le
vel; struct
tree_element
*link[5];
};
typedef struct
tree_element node; void
main()
{
int
gd=DETECT,gm;
node *root;
```

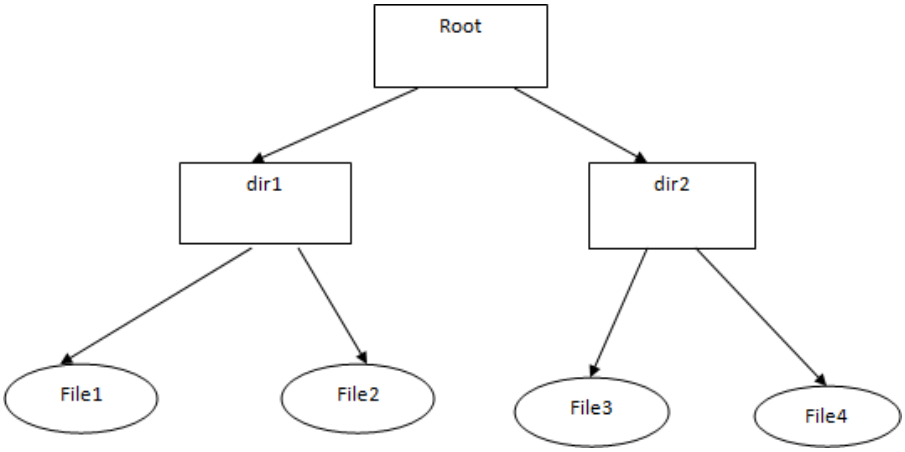
```
root=NULL;
clrscr();
create(&root,0,"null",0,639,
320); clrscr();
initgraph(&gd,&gm,"c:\\tc\\b
gi"); display(root);
```

```
getch();
closegraph(
);
}
create(node **root,int lev,char *dname,int lx,int rx,int x)
{
int i,gap;
if(*root==NULL
)
{
(*root)=(node*)malloc(sizeof(node))
; printf("enter name of
dir/file(under %s):",dname);
fflush(stdin);
gets((*root)-
>name);
if(lev==0||lev==
1) (*root)-
>ftype=1; else
(*root)->ftype=2;
(*root)-
>level=lev;
(*root)-
>y=50+lev*50;
(*root)->x=x;
(*root)-
>lx=lx;
(*root)-
>rx=rx;
for(i=0;i<5;i
++)
(*root)-
>link[i]=NULL;
if((*root)-
>ftype==1)
{
if(lev==0||lev==1)
{
if((*root)-
>level==0)
printf("How many
users"); else
printf("hoe many
files");
printf("(for%s):",(*root
)->name);
scanf("%d",&(*root)->nc);
}
else   (*root)-
>nc=0;
if((*root)-
>nc==0)
gap=rxlx;
else
gap=(rx-lx)/(*root)-
>nc;
for(i=0;i<(*root)-
>nc; i++)
```

```
create(&((*root)->link[i]),lev+1,(*root)- >name,lx+gap*i,lx+gap*i+gap,lx+gap*i+gap/2);
}
else (*root)->nc=0;
}
}
display(node *root)
{
int i;
settextstyle(2,0
,4);
settextjustify(1
,1);
setfillstyle(1,B
LUE);
setcolor(14);
if(root!=NULL)
{
for(i=0;i<root->nc;i++)
{
line(root->x,root->y,root->link[i]->x,root->link[i]->y);
```

```
}
if(root->ftype==1)
  bar3d(root->x-20,root->y-10,root-
   >x+20,root->y+10,0,0); else
fillellipse(root->x,root-
>y,20,20); outtextxy(root-
>x,root->y,root->name);
for(i=0;i<root->nc;i++)
{
display(root->link[i]);
}
} }
```

4.2.4 INPUT AND OUTPUT



EXPERIMENT-4.3

4.3.1 OBJECTIVE

Simulate Hierarchical level directory File Organization Technique

4.3.2 PROGRAM LOGIC

```
Step 1: Start the program.
Step 2: Get the name of the
directories. Step 3: get the
number of files.
Step 4: Get the name of the each file.
Step 5: Now each file is in the form of fill circle
Step 6: Every file is connected with respective directory
Step 7: Display the connected graph along with name in
hierarchical way using graphics Step 8:  Stop the program.
```

4.3.3 PROGRAM

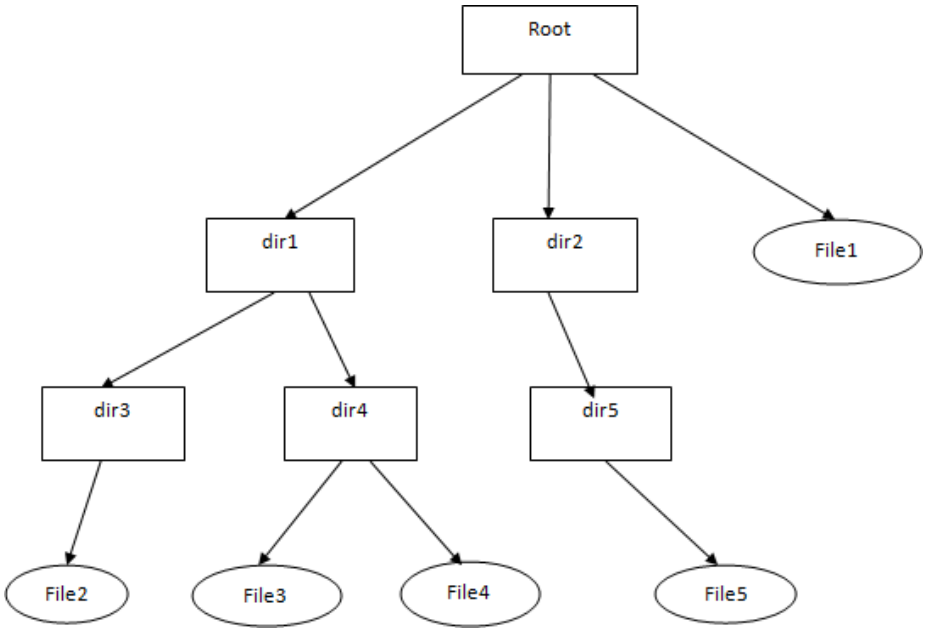
```
#include<stdio.h>
#include<graphics.h>
struct tree_element
```

```
{
char name[20];
int
x,y,ftype,lx,rx,nc,le
vel; struct
tree_element
*link[5];
};
typedef struct
tree_element node;
void main()
{
int
gd=DETECT,gm;
node *root;
root=NULL;
clrscr();
create(&root,0,"root",0,639,
320); clrscr();
initgraph(&gd,&gm,"c:\\tc\\B
GI"); display(root);
getch();
closegraph(
);
}
create(node **root,int lev,char *dname,int lx,int rx,int x)
{
int i,gap;
if(*root==NULL
)
{
(*root)=(node
*)malloc(sizeof(node));
printf("Enter name of dir/file(under
%s) :",dname); fflush(stdin);
gets((*root)->name);
printf("enter 1 for Dir/2
for file :");
scanf("%d",&(*root)-
>ftype); (*root)-
>level=lev;
(*root)-
>y=50+lev*50;
(*root)->x=x;
(*root)-
>lx=lx;
(*root)-
>rx=rx;
for(i=0;i<5;i
++)
(*root)-
>link[i]=NULL;
if((*root)-
>ftype==1)
{
printf("No of sub directories/files(for %s):",(*root)-
>name); scanf("%d",&(*root)->nc); if((*root)->nc==0)
gap=rx-lx;
```

```
else gap=(rx-
lx)/(*root)- >nc;
for(i=0;i<(*root)-
>nc;i++)
create(&((*root)->link[i]),lev+1,(*root)- >name,lx+gap*i,lx+gap*i+gap,lx+gap*i+gap/2);
}
else (*root)- >nc=0;
}}
display(node *root)
{
int i;
settextstyle(2,0
,4);
settextjustify(1
,1);
setfillstyle(1,B
LUE);
setcolor(14);
if(root!=NULL)
```

```
{
for(i=0;i<root->nc;i++)
{
line(root->x,root->y,root->link[i]->x,root->link[i]->y);
}
if(root->ftype==1)
   bar3d(root->x-20,root->y-10,root-
   >x+20,root->y+10,0,0); else
fillellipse(root->x,root-
>y,20,20); outtextxy(root-
>x,root->y,root- >name);
for(i=0;i<root->nc;i++)
{
display(root->link[i]);
}
}
}
```

4.3.4 INPUT AND OUTPUT



EXPERIMENT-4.4

4.4.1 OBJECTIVE

Simulate DAG File Organization Technique

4.4.2 PROGRAM LOGIC

```
Step 1: Start the program.
Step 2: Get the name of the
directories. Step 3: get the
number of files.
Step 4: Get the name of the each file.
Step 5: Now each file is in the form of fill circle
Step 6: Every file is connected with respective directory
Step 7: Display the connected graph along with
name using graphics Step 8:  Stop the program.
```

4.4.3 PROGRAM

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
#include<string.h>
struct tree_element
{
char name[20];
int
x,y,ftype,lx,rx,nc,le
vel; struct
tree_element
*link[5];
};
typedef struct
tree_element node;
typedef struct
{
char
from[20];
char to[20];
}link;
 Link
L[10]; Int
nofl; node
* root;
void
main()
{
int
gd=DETECT,gm;
root=NULL;
clrscr();
create(&root,0,"root",0,639,320);
read_links();
clrscr();
initgraph(&gd,&gm,"c:\\tc\\B
GI"); draw_link_lines();
display(roo
t);
getch();
closegraph(
);
}
read_links()
```

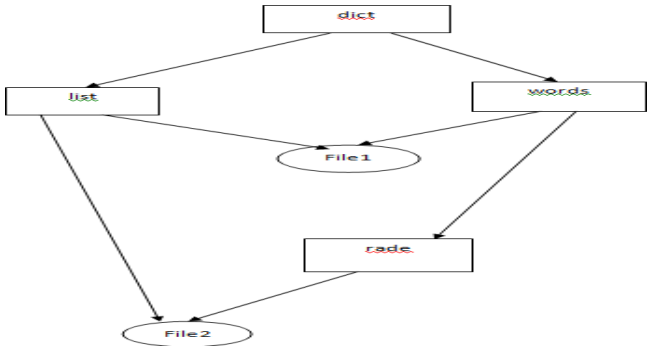
```
{
int i;
printf("how many
links");
scanf("%d",&nofl);
for(i=0;i<nofl;i++)
{
printf("File/dir
:");
fflush(stdin);
gets(L[i].from);
printf("user
name:");
```

```
fflush(std
in);
gets(L[i].
to);
}
}
draw_link_lines()
{
int
i,x1,y1,x2,y2;
for(i=0;i<nofl
;i++)
{
search(root,L[i].from,&x1,
&y1);
search(root,L[i].to,&x2,&
y2);
setcolor(LIGHTGREEN);
setlinestyle(3,0,1);
line(x1,y1,x2,y2);
setcolor(YELLOW);
setlinestyle(0,0,1);
}
}
search(node *root,char *s,int *x,int *y)
{
int i;
if(root!=NULL
)
{
if(strcmpi(root->name,s)==0)
{
*x=root->x;
*y=root-
>y;
return;
}
else
{
for(i=0;i<root-
>nc;i++)
search(root-
>link[i],s,x,y);
}
}
} create(node **root,int
lev,char * dname,int
lx,int rx,int x)
{
int i,gap;
if(*root==NULL
)
{
(*root)=(node
*)malloc(sizeof(node));
printf("enter name of
dir/file(under %s):",dname);
fflush(stdin);
```

```
gets((*root)->name);
printf("enter 1 for dir/
2 for file:");
scanf("%d",&(*root)-
>ftype); (*root)-
>level=lev;
(*root)-
>y=50+lev*50;
(*root)->x=x;
(*root)-
>lx=lx;
(*root)-
>rx=rx;
for(i=0;i<5;i
++)
(*root)-
>link[i]=NULL;
if((*root)-
>ftype==1)
{
printf("no of sub directories /files (for %s):",(*root)->name); scanf("%d",&(*root)->nc);
```

```
if((*root)-
>nc==0) gap=rx-
lx;
else
gap=(rx-
lx)/(*root)->nc;
for(i=0;i<(*root)-
>nc;i++)
create( & ( (*root)->link[i] ) , lev+1 ,
(*root)->name,lx+gap*i,lx+gap*i+gap,lx+gap*i+gap/2);
}
else
(*root)->nc=0;
}
}
display(node *root)
{
int i;
settextstyle(2,0
,4);
settextjustify(1
,1);
setfillstyle(1,B
LUE);
setcolor(14);
if(root!=NULL)
{
for(i=0;i<root->nc;i++)
{
line(root->x,root->y,root->link[i]->x,root->link[i]->y);
}
if(root->ftype==1) bar3d(root->x-20,root->y-10,root-
>x+20,root->y+10,0,0); else
fillellipse(root->x,root-
>y,20,20); outtextxy(root-
>x,root->y,root->name);
for(i=0;i<root->nc;i++)
{
display(root->link[i]);
}
}}
```

4.4.4 INPUT AND OUTPUT



4.5 PRE LAB QUESTIONS

1. Define file
2. What are the attributes of a file?
3. Define directory.
4. In how many ways we can implement the directory?
5. Differentiate DAG and other directories

4.6 LAB ASSIGNMENT

1. Write a C program to simulate acyclic graph directory structure?
2. Write a C Program to simulate general graph directory structure?

4.7 POST LAB QUESTIONS

1. Which of the directory structures is efficient? Why?
2. Which directory structure does not provide user-level isolation and protection?
3. What is the advantage of hierarchical directory structure?

EXPERIMENT – 5

5.1 OBJECTIVE

Program for Bankers algorithm for deadlock avoidance and deadlock prevention.

5.2 PROGRAM LOGIC

```
Step-1: Start the program.
Step- 2: Get the values of
resources and processes. Step-3: Get
the avail value.
Step-4: After allocation find
the need value. Step-5: Check
whether its possible to allocate.
Step-6: If it is possible then the
system is in safe state. Step-7: Else
system is not in safety state.
  Step-8: If the new request comes then check that
       the system is in safety or not if we allow
       the request.
Step-9: stop the program.
```

5.3 PROGRAM

```
#include<stdio
.h>
#include<conio
.h>  #define  s
15
void main()
{
   int n,m,a[5],e[5],c[5][5],r[5][5];
int
i,j,k,cou,l,flag,count
; cou=flag=count=0;
printf("enter no of
processes");
scanf("%d",&n);
printf("enter no of
resource classes");
scanf("%d",&m);
for(i=0;i<m;i++)
{
printf("\n enter instances of
resources class %d "i+1);
scanf("%d",&e[i]);
printf("\n enter free vectors of resources class %d
(resources available):",i+1); scanf("%d",&a[i]);
}
  printf("\n enter the current
     allocation matrix \n");
     for(i=0;i<n;i++)
   for(j=0;j<m;j++)
```

```
   scanf("%d",&c[i][j])
   ;
   printf("\n enter the request
matrix \n"); for(i=0;i<n;i++)
 for(j=0;j<m;j++)
scanf("%d",&r[i][j]);
for(l=0;l<n;l++)
{
    cou=0;fla
   g=0;
   for(i=0;i<
   n;i++)
{
    cou=0;flag=0
     ;
     for(j=0;j<m
     ;j++)
```

```
     if(a[j]>=r[i][j])
      {
 cou++;
if(c[i][j]=
=-1)
flag++;
}
if(cou==m)
{
    if(flag!=m)
{
    printf("\n process %d is satisfied",i+1);
  count++;
   for(k=0;k<m;
   k++)
  {
    a[k]+=c[i][k];
printf("\n a[%d]=%d
\t",k,a[k]); getch();
c[i][k]=-1;
} }   }   } }
if(count==n)
printf("\n process sre
completed \n"); else
printf("\n unsafe state
occurs \n"); getch();}
```

5.4 INPUT AND OUTPUT

```
enter no of processes5
enter no of resource classes3
   enter instances of resources class 1 10
   enter free vectors of resources class 1
   (resources available):3 enter instances
   of resources class 2 5
   enter free vectors of resources class 2
   (resources available):3 enter instances
   of resources class 3 7
   enter free vectors of resources class 3
   (resources available):2 enter the
   current allocation matrix
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2
 enter the request
matrix 7 4 3
1 2 2
6 0 0
0 1 1
4 3 1
   process 2 is
   satisfied
   a[0]=5
   a[1]=3
   a[2]=2
```

```
process 4 is
satisfied
a[0]=7
a[1]=4
a[2]=3
process 5 is
satisfied
a[0]=7
```

```
   a[1]=4
   a[2]=5
   process 1 is
   satisfied
   a[0]=7
   a[1]=5
   a[2]=5
   process 3 is
   satisfied
   a[0]=10
   a[1]=5
   a[2]=7
process are completed
```

5.5 PRE LAB QUESTIONS

1. Is it possible to have a deadlock involving only one process?
2. Define Resource Allocation Graph
3. State Bankers algorithm.
4. Differentiate preemption and No preemption

5.6 LAB ASSIGNMENT

1. Write a program with following specifications:
   Command line input: of a file name
   - The file contains the initial state of the system as given below (example included in right column)

```
#no of resources                                                    4
#no of instances of each resour                  2   4   5   3
#no of process                                                      3
#no of instancesof each resource tha   1 1 1 1, 2 3 1 2, 2 2 1    3
t
```

each processes needs in its lifetime.The rule is that once a process uses a resource instance and returns it, the process no more needs it.

2. Implementation of resource allocation graph (RAG).
3. Conversion of resource allocation graph (RAG) to wait-for-graph (WFG) for each type of method used for storing graph.

5.7 POST LAB QUESTIONS

1. What is a Safe State and what is its use in deadlock avoidance?
2. What is deadlock?
3. What are read write locks?
4. What are the necessary conditions for deadlock to occur?

EXPERIMENT-6.1

6.1.1 OBJECTIVE

Program to perform FIFO page replacement algorithm.

6.1.2 PROGRAM LOGIC

Step 1: Create a queue to hold all pages in memory
Step 2: When the page is required replace the page
at the head of the queue Step 3: Now the new page is
inserted at the tail of the queue
Step 4: When the page fault occurs replace page present
at the head of thequeue Step 5: Stop the program

6.1.3 PROGRAM

```
#include<conio
.h>
#include<stdio
.h>        void
main()
{
int i,j,n,k,m,r,pagef;
int a[100],b[100];
k=0,pagef=0
; clrscr();
printf("enter the number of
pages\n"); scanf("%d",&n);
printf("enter the number of frames in
main memory\n"); scanf("%d",&m);
   for(i=0;i<m;i++)
       b[i]=-1;
   for(i=0;i<n;i++)
   {
       printf("enter the
       element %d\n",i);
       scanf("%d",&a[i]);
   }
   for(i=0;i<n;i++)
   {
        for(j=0;j<m;j++)
        {
              if(b[j]==a[i])
              {
                    printf("the page %d is already present in
                    main memory\n",a[i]); break;
              }
        }
        if(j==m)
        {
              b[k]=a
              [i];
              k++;
              if(k==m)
                    k=0;
              for(r=0;r<m;r++)
```

```
                printf("%d
            ",b[r]);
            printf("\n");
            pagef++;
        }
    }
    printf("number of page faults: %d",pagef);
```

```
getch();
}
```

6.1.4 INPUT AND OUTPUT

```
enter the number of
pages 20
enter the number of frames in
main memory 3
enter the
element 0 7
enter the
element 1 0
enter the
element 2 1
enter the
element 3 2
enter the
element 4 0
enter the
element 5 3
enter the
element 6 0
enter the
element 7 4
enter the
element 8 2
enter the
element 9 3
enter the
element 10 0
enter the
element 11 3
enter the
element 12 2
enter the
element 13 1
enter the
element 14 2
enter the
element 15 0
enter the
element 16 1
enter the
element 17 7
enter the
element 18 0
enter the
element 19 1
7  -1  -1
7  0  -1
7  0  1
2  0  1
the page 0 is already present in
main memory 2  3  1
2  3  0
```

```
4  3  0
4  2  0
4  2  3
0  2  3
the page 3 is already present in
main  memory  the  page  2  is
already present in main memory 0
1  3
0  1  2
the page 0 is already present in
main  memory  the  page  1  is
already present in main memory 7
1  2
7  0  2
7  0  1
number of page faults: 15
```

EXPERIMENT-6.2

6.2.1 OBJECTIVE

Program to perform LRU page replacement algorithm

6.2.2 PROGRAM LOGIC

```
Step 1: Create a queue to hold all pages in memory
Step 2: When the page is required replace the page
at the head of the queue Step 3: Now the new page is
inserted at the tail of the queue
Step 4: Create a stack
Step 5: When the page fault occurs replace page present at the bottom of the stack
```

6.2.3 PROGRAM

```
#include<conio
.h>
#include<stdio
.h>
int lrupage(int
w[],int m); void
main()
{
int i,j,n,m,r,pagef,l,h,p;
int a[100],b[100],w[10];
pagef=0;
clrscr()
;
printf("enter the number of
pages\n"); scanf("%d",&n);
printf("enter the number of frames in
main memory\n"); scanf("%d",&m);
   for(i=0;i<m;i++)
       b[i]=-1;
   for(i=0;i<n;i++)
   {
```

```
        printf("enter the
        element %d\n",i);
        scanf("%d",&a[i]);
    }
    for(i=0;i<m;i++)
    {
        b[i]=a[i];
         for(r=0;r<m
         ;r++)
```

```
                printf("%d
            ",b[r]);
            printf("\n");
            pagef++;
    }
    for(i=m;i<n;i++)
    {
         for(j=0;j<m;j++)
         {
               if(b[j]==a[i])
               {
               printf("the page %d is already present in
                     main memory\n",a[i]); break;
               }
         }
         if(j=
         =m)
         {           for(l=0;l<m;l++)
                     {
                           for(p=i-1;p>=0;p--)
                           {
                                 if(b[l]==a[p])
                                 {
                                       w[l]=
                                       p;
                                 }     break
                           }           ;
                           if(p==n)
                           {
                           b[l]=a
                           [i];
                           break;
                     }     }
                     if(l=
                     =m)
                     {
                           h=lrupage(w
                           ,m);
                     }     b[h]=a[i];
               for(r=0;r<m
               ;r++)
                     printf("%d ",b[r]);
         printf("\n");
               pagef++;
          }
    }
    printf("number of page
    faults: %d",pagef); getch();
    }
    int lrupage(int w[],int m)
    {
         int max,k,h=0;
         max=w[0];
         for(k=1;k<m;k++)
                           if(w[k]<max)
                           {
                                 max=w[k
                                 ]; h=k;
```

```
                    }
                    return
                    h;
}
```

6.2.4 INPUT AND OUTPUT

```
enter the number of
pages 20
enter the number of frames in
main memory 3
enter the
element 0 7
enter the
element 1 0
enter the
element 2 1
enter the
element 3 2
enter the
element 4 0
enter the
element 5 3
enter the
element 6 0
enter the
element 7 4
enter the
element 8 2
enter the
element 9 3
enter the
element 10 0
enter the
element 11 3
enter the
element 12 2
enter the
element 13 1
enter the
element 14 2
enter the
element 15 0
enter the
element 16 1
enter the
element 17 7
enter the
element 18 0
enter the
element 19 1
7  -1  -1
7  0 -1
7  0  1
2  0  1
the page 0 is already present in
main memory 2  0  3
the page 0 is already present in
main memory 4  0  3
4  0  2
4  3  2
```

```
0  3  2
the page 3 is already present in
main  memory  the  page  2  is
already present in main memory 1
3  2
the page 2 is already present in
main memory 1  0  2
the page 1 is already present in
main memory 1  0  7
the page 0 is already present in
main  memory  the  page  1  is
already present in main memory
number of page faults: 12
```

EXPERIMENT-6.3

6.3.1 OBJECTIVE

Program to perform LFU page replacement algorithm

6.4.1 PROGRAM LOGIC

```
Step 1: Create a queue to hold all pages in memory
Step 2: When the page is required replace the page
at the head of the queue Step 3: Now the new page is
inserted at the tail of the queue
Step 4: Create a stack
Step 5: When the page fault occurs replace page present at the bottom of the stack
```

6.4.2 PROGRAM

```
#include<stdio
.h> int main()
{
   int f,p;
   int
   pages[50],frame[10],hit=0,count[5
   0],time[50]; int
   i,j,page,flag,least,minTime,temp;

   printf("Enter no of
   frames : ");
   scanf("%d",&f);
   printf("Enter no of
   pages : ");
   scanf("%d",&p);

   for(i=0;i<f;i++)
   {
    frame[i]=-1;
   }
   for(i=0;i<50;i++)
   {
```

```
 count[i]=0;
}
printf("Enter page
no : \n");
for(i=0;i<p;i++)
{
 scanf("%d",&pages[i]);
}
```

```
    printf("\n");
    for(i=0;i<p;i++)
    {
     count[pages[
     i]]++;
     time[pages[i
     ]]=i;
     flag=1;
     least=frame[
     0];
     for(j=0;j<f;
     j++)
     {
      if(frame[j]==-1 || frame[j]==pages[i])
      {
       if(frame[j]!=-1)
       {
         hit++;
       }
       flag=0;
       frame[j]=pag
       es[i];
       break;
      }
      if(count[least]>count[frame[j]])
      {
       least=frame[j];
      }
     }
     if(flag)
     {
      minTime=50
      ;
      for(j=0;j<
      f;j++)
      {
       if(count[frame[j]]==count[least] && time[frame[j]]<minTime)
       {
        temp=j;
        minTime=time[fram
        e[j]];
       }
      }
      count[frame[tem
      p]]=0;
      frame[temp]=pag
      es[i];
     }
     for(j=0;j<f;j++)
     {
      printf("%d ",frame[j]);
     }
     printf("\n");
    }
    printf("Page hit =
    %d",hit); return
    0;
}
```

6.4.3 INPUT AND OUTPUT

```
enter the number of
pages 20
enter the number of frames in
main memory 3
enter the
element 0 7
```

```
enter the
element 1 0
enter the
element 2 1
enter the
element 3 2
enter the
element 4 0
enter the
element 5 3
enter the
element 6 0
enter the
element 7 4
enter the
element 8 2
enter the
element 9 3
enter the
element 10 0
enter the
element 11 3
enter the
element 12 2
enter the
element 13 1
enter the
element 14 2
enter the
element 15 0
enter the
element 16 1
enter the
element 17 7
enter the
element 18 0
enter the
element 19 1
7  -1  -1
7  0 -1
7  0  1
2  0  1
the page 0 is already present in
main memory 2  0  3
the page 0 is already present in
main memory 2  4  3
the page 2 is already present in
main  memory  the  page  3  is
already present in main memory 2
0  3
the page 3 is already present in
main  memory  the  page  2  is
already present in main memory 2
0  1
the page 2 is already present in
main  memory  the  page  0  is
already present in main memory
the page 1 is already present in
```

```
main memory 7  0  1
the page 0 is already present in main memory
```

```
the page 1 is already present in
main memory number of page
faults: 9
```

6.5 PRE LAB QUESTIONS

1. What are the different Dynamic Storage-Allocation methods?
2. Under what circumstances do page faults occur?
3. What are local and global page replacements
4. Define latency, transfer and seek time with respect to disk I/O
5. What are demand-paging and pre-paging?

6.6 LAB ASSIGNMENT

1. Write a memory allocator simulator that implements following memory allocation scheme:

a. The calling processes requests memory chunks of needed sizes
b. The returned chunk is treated as contiguous by the process
c. The process may free an allocated chunk

6.7 POST LAB QUESTIONS

1. Define paging.
2. Discuss the advantages and disadvantages of paging
3. Differentiate segmentation and fragmentation
4. Differentiate LFU and LRU
5. Why Optimal page replacement is better than other algorithms?

EXPERIMENT-7.1

7.1.1 OBJECTIVE

Simulate the Paging Technique of memory management

a) Memory management policy- Paging

7.1.2 PROGRAM LOGIC

```
Step 1: Read all the necessary input from the keyboard.
Step 2: Pages - Logical memory is broken into
fixed - sized blocks. Step 3: Frames – Physical
memory is broken into fixed – sized blocks. Step
4: Calculate the physical address using the
following
      Physical address = ( Frame number * Frame
size ) + offset Step 5: Display the physical
address.
Step 6: Stop the process.
```

7.1.3 PROGRAM

```
#include
<stdio.h>
#include
<conio.h>
struct pstruct
{
       int
       fno;
       int
       pbit;
}ptable[10];

int pmsize,lmsize,psize,frame,page,ftable[20],frameno;

      void info()
{
       printf("\n\nMEMORY MANAGEMENT USING
       PAGING\n\n"); printf("\n\nEnter the Size of
       Physical memory: "); scanf("%d",&pmsize);
       printf("\n\nEnter the size of
       Logical memory: ");
       scanf("%d",&lmsize);
       printf("\n\nEnter the
       partition size: ");
       scanf("%d",&psize);
       frame = (int)
       pmsize/psize; page
       = (int)
       lmsize/psize;
       printf("\nThe physical memory is divided into %d
       no.of frames\n",frame); printf("\nThe Logical
       memory is divided into %d no.of pages",page);
}
void assign()
{
```

```
int i;
for (i=0;i<page;i++)
{
ptable[i].fno = -1;
ptable[i].pbit= -1;
}
for(i=0; i<frame;i++)
      ftable[i] =
32555; for
(i=0;i<page;i++)
{
printf("\n\nEnter the Frame number where page %d
      must be placed: ",i); scanf("%d",&frameno);
```

```
                ftable[frameno
                ] = i;
                if(ptable[i].p
                bit == -1)
                {
                      ptable[i].fno =
                      frameno;
                      ptable[i].pbit =
                      1;
                }
        }
        getch();
        printf("\n\nPAGE TABLE\n\n");
        printf("PageAddress FrameNo.
        PresenceBit\n\n"); for
        (i=0;i<page;i++)
              printf("%d\t\t%d\t\t%d\n",i,ptable[i].fno,ptab
        le[i].pbit); printf("\n\n\n\tFRAME TABLE\n\n");
        printf("FrameAddress PageNo\n\n");
        for(i=0;i<frame;i++)
              printf("%d\t\t%d\n",i,ftable[i]);
}
void cphyaddr()
{
        int
        laddr,paddr,disp,phyadd
        r,baddr; getch();
        printf("\n\n\n\tProcess to create the
        Physical Address\n\n"); printf("\nEnter
        the Base Address: ");
        scanf("%d",&baddr);
        printf("\nEnter theLogical
        Address: ");
        scanf("%d",&laddr);
        paddr = laddr
        / psize; disp
        = laddr %
        psize;
         if(ptable[paddr].pbit == 1 )
              phyaddr = baddr + (ptable[paddr].fno*psize) + disp;
        printf("\nThe Physical Address where the instruction present: %d",phyaddr);
}
void main()
{
        clrscr();
        info();
        assign(
        );
        cphyadd
        r();
        getch()
        ;
}
```

7.1.4 INPUT AND OUTPUT

```
MEMORY MANAGEMENT USING PAGING
```

```
 Enter the Size of Physical
memory: 16 Enter the size of
Logical memory: 8 Enter the
partition size: 2
The physical memory is divided into
8 no.of frames The Logical memory is
divided into 4 no.of pages
Enter the Frame number where page 0
must be placed: 5 Enter the Frame number
where page 1 must be placed: 6 Enter the
Frame number where page 2 must be
placed: 7 Enter the Frame number where
page 3 must be placed: 2 PAGE TABLE
```

PageAddress	FrameNo.	PresenceBit
0	5	1
1	6	1

```
      2            7               1
      3            2               1
   FRAME TABLE
    FrameAddress     PageNo
      0            32555
      1            32555
      2               3
      3            32555
      4            32555
      5             0
      6             1
      7             2
   Process to create the
    Physical Address Enter
    the Base Address: 1000
    Enter theLogical Address: 3
The Physical Address where the instruction present: 1013
```

EXPERIMENT-7.2

7.2.1 OBJECTIVE

To implement the memory management policy-segmentation

7.2.2 PROGRAM LOGIC

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

7.2.3 PROGRAM

```
#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<=so
           st;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<st[i].limit;j++
            )
                  {
                  printf("Enter the %d address Value:
                  ",(st[i].baddr + j));
                  scanf("%d",&st[i].val[j]);
                  }
            }
}

void ptladdr()
{
      int
      i,swd,d=0,n,s,disp,
      paddr; clrscr();
      printf("\n\n\t\t\t SEGMENT TABLE \n\n");
      printf("\n\t
               SEG.NO\tBASE ADDRESS\t LIMIT
      \n\n"); for(i=1;i<=sost;i++)
            printf("\t\t%d
                        \t\t%d\t\t%d\n\n",st[i].sno,st[i
      ].baddr,st[i].limit); printf("\n\nEnter the
      logical Address: ");
      scanf("%d",&s
      wd); n=swd;
      while (n != 0)
      {
            n=n/1
            0;
            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);
            el    printf("\n\tThe value is: %d",(
            se    st[s].val[disp] ) );
      }
      el
      se          printf("\n\t\tLimit of segment %d is high\n\n",s);
            printf("\n\t\tInvalid Segment Address \n");
```

```
}
void main()
{
char
ch;
clrscr
();
gstinf
o();
do
{
ptladdr();
```

```
        printf("\n\t Do U want to
        Continue(Y/N)"); flushall();
        scanf("%c",&ch);
        }while (ch == 'Y' || ch == 'y' );

        getch();
}
```

7.2.4 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)
```

7.5 PRE LAB QUESTION

1. Define Memory Partitioning, Paging, Segmentation.
2. In loading programs into memory, what is the difference between load-time dynamic linking and run-time dynamic linking?
3. Give the functionalities of OS in memory management
4. Why might the direct blocks be stored in the inode itself?
5. List out memory allocation algorithms which was commonly used

7.6 LAB ASSIGNMENT

1. Implement memory allocation algorithms first fit and best fit.
2. Implement different paging techniques

7.7 POST LAB QUESTIONS

1. What is random access time?
2. What is head crash
3. What is storage area network
4. What is NAS
5. What is Disk scheduling

EXPERIMENT-8

8.1 OBJECTIVE

Simulate how parent and child processes use shared memory and address space.

8.2 PROGRAM LOGIC

Step 1: Pointer Shm points to the shared memory segment
Step 2: Parent forks a child process to run function ClientProcess
Step 3: Two *identical* copies of address spaces are created, each of which has a variable Shm whose value is a pointer to the shared memory
Step 4: The child process has already known the location of the shared memory segment and does not have to use shmget() and shmat()
Step 5: Both the process will share the a to z characters which is resided in shared memory region having same address space

8.3 PROGRAM

Client program to demonstrate shared

memory#include<sys/types.h>

```
#include<sys/ipc.h>
#include<sys/s
hm.h>
#include<stdio
.h>
#include<stdli
b.h> #define
SHMSZ 27
main()
{
 int
 shmid;
 key_t
 key; char
 *shm, *s;
 key=5678;
  if((shmid=shmget(key,SHMSZ,0666))<0)
{
 perror("shm
 get");
 exit(1);
}
if((shm=shmat(shmid,NULL,0))==(char*)-1)
{
perror("shm
at");
exit(1);
}
for(s=shm;*s!=
NULL;s++)
putchar(*s);
putchar('\n');
*shm='*';
exit(0);
}
```

SERVER:

```
#include<sys/ty
pes.h>
#include<sys/i
pc.h>
#include<sys/s
hm.h>
#include<stdio
.h>
#include<stdli
b.h> #define
SHMSZ 27
main()
```

```
{
 char c;
 int
 shmid;
 key_t
 key;
 char*shm
 ,*s;
 key=5678
 ;
  if((shmid=shmget(key,SHMSZ,IPC_CREAT | 0666))<0)
 {
 perror("shm
 get");
 exit(1);
 }
if((shm=shmat(shmid,NULL,0))==(char*)-1)
 {
 perror("shm
 at");
 exit(1);
 }
s=shm;
 for(c='a';c<=
 'z';c++)
  *s++=c
  ;
  s=NULL
  ;
 while(*shm!
 ='*')
 sleep(1);
 exit(0);
}
```

8.4 INPUT AND OUTPUT

```
Sh
client.s
h Sh
server.s
h
abcdefghijklmnopqrstuvwxyz
```

8.5 PRE-LAB QUESTIONS

1.What is the need for process synchronization? 2.Define a semaphore?
3.Define Readers writers problem?

8.6 LAB ASSIGNMENT

1.Write a C program to simulate Readers Writers problem using message-passing system.

8.7 POST-LAB QUESTIONS

1. Discuss the consequences of considering bounded and unbounded buffers in producer-consumer problem?

2. Can producer and consumer processes access the shared memory concurrently? If not which techniqueprovidessuch a benefit?

EXPERIMENT-9

9.1 OBJECTIVE

Simulate sleeping barber problem

9.2 PROGRAM LOGICdef

```
Barber():
while true:
wait(custReady)
wait(accessWRSeats)
numberOfFreeWRSeats
+= 1
signal(barberReady)
signal(accessWRSeat
s)
  # (Cut hair here.)
def Customer():
  wait(accessWRS
  eats)
if
numberOfFreeWRSeats
> 0:
numberOfFreeWRSeats
-= 1
signal(custReady)
signal(accessWRSeats
) wait(barberReady)
  else:
signal(accessWR
Seats)
# (Leave without a haircut.)
```

9.3 PROGRAM

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

```
<pthread.h>
#include <semaphore.h> // The maximum number of
customer threads. #define MAX_CUSTOMERS 25 //
Function prototypes...

void *customer(void
*num); void
*barber(void *);
void
randwait(int
secs); sem_t
waitingRoom;
sem_t
barberPillow;
sem_t seatBelt;
int allDone = 0;

int main(int argc, char *argv[])
{
  pthread_t btid;
  pthread_t tid[MAX_CUSTOMERS];
```

```
  int i, x, numCustomers, numChairs; int Number[MAX_CUSTOMERS];
  printf("Maximum number of customers can only be 25. Enter number
  of customers and chairs.\n"); scanf("%d",&x);
  numCustomers
  = x;
  scanf("%d",&x
  ); numChairs
  = x;
  if (numCustomers > MAX_CUSTOMERS) {
printf("The maximum number of Customers is %d.\n",
   MAX_CUSTOMERS); system("PAUSE");
   return 0;
  }
  printf("A solution to the sleeping barber
  problem using semaphores.\n"); for (i = 0; i <
  MAX_CUSTOMERS; i++) {
    Number[i] = i;
  }
  sem_init(&waitingRoom, 0, numChairs);
  sem_init(&barberChair, 0, 1);
  sem_init(&barberPillow, 0, 0);
  sem_init(&seatBelt, 0, 0);
  pthread_create(&btid, NULL,
  barber, NULL);

   for (i = 0; i < numCustomers; i++) {
    pthread_create(&tid[i], NULL, customer, (void *)&Number[i]);
  }
  for (i = 0; i <
    numCustomers; i++) {
    pthread_join(tid[i],NU
    LL);
  }
 sem_post(&barberPillow); // Wake the
  barber so he will exit.
  pthread_join(btid,NULL);
  system("PAUS
  E"); return
  0;
}

void *customer(void
  *number) { int num =
  *(int *)number;
  printf("Customer %d leaving for
  barber shop.\n", num); randwait(5);
  printf("Customer %d arrived at barber shop.\n", num);
                                        sem_wait(&waitingR
  oom); printf("Customer %d entering waiting room.\n",
  num);
                                        sem_wait(&barberCh
  air); sem_post(&waitingRoom);
  printf("Customer %d waking the
  barber.\n", num);
  sem_post(&barberPillow);
  sem_wait(&seatBelt); // Give
  up the chair.
  sem_post(&barberChair);
```

```
  printf("Customer %d leaving barber shop.\n", num);
}

void *barber(void *junk)
{

 while (!allDone) {
  printf("The barber is
  sleeping\n");
  sem_wait(&barberPillow
  );
  if (!allDone)
  {
   printf("The barber is
   cutting hair\n");
   randwait(3);
```

```
        printf("The barber has finished cutting hair.\n");   sem_post(&seatBelt);
       }
       else {
         printf("The barber is going home for the day.\n");
       }
      }
     }

     void randwait(int secs) {
       int len = 1; // Generate an
       arbit number... sleep(len);
     }
```

9.4 INPUT AND OUTPUT

```
A solution to the sleeping barber
problem using semaphores. Maximum number
of customers can only be 25.
 Enter number of customers
and chairs. 4
4
The barber is
sleeping The
barber is
cutting hair
The barber has finished
cutting hair. Customer 1
leaving for barber shop.
Customer 2 arrived at
barber shop. Customer 2
entering waiting room.
The barber is sleeping
Customer 2 wakingup
the barber The barber
is cutting hair
The barber has finished
cutting hair. Customer 2
leaving barber shop.
Customer 3 arrived at
barber shop. Customer 3
entering waiting room.
The barber is sleeping
Customer 3 wakingup
the barber The barber
is cutting hair
The barber has finished cutting hair.
```

9.5 PRE-LAB QUESTIONS

1. Differentiate between a monitor, semaphore and a binary semaphore?
2. Define clearly the Sleeping Barber problem?
3. Define Semaphore.
4. Write Test and set method.

9.6 LAB ASSIGNMENT

1. Write a C program to simulate readers-writers problem using semaphores?

9.7 POST-LAB QUESTIONS

1. Identify the scenarios in the sleeping barber problem that leads to the deadlock situations?
2. Define critical Section.
3. What are three properties to satisfy to overcome critical section problem

EXPERIMENT-10

10.1 OBJECTIVE

Simulate dining philosopher„s problem.

10.2 PROGRAM LOGIC

```
process
P[i]
while
true do
 {  THINK;
  PICKUP(CHOPSTICK[i], CHOPSTICK[i+1
  mod 5]); EAT;
  PUTDOWN(CHOPSTICK[i], CHOPSTICK[i+1 mod 5])
 }
```

10.3 PROGRAM

```
int tph, philname[20], status[20], howhung, hu[20], cho; main()
{
int
i;
clrsc
r();
printf("\n\nDINING PHILOSOPHER PROBLEM"); printf("\nEnter the total no. of
philosophers: "); scanf("%d",&tph);
for(i=0;i<tph;i++)
{
philname[i] =
(i+1);
status[i]=1;
}
printf("How many are
hungry : ");
scanf("%d", &howhung);
if(howhung==tph)
{
printf("\nAll are hungry..\nDead lock stage will occur");
printf("\nExiting..");
}
else
{
for(i=0;i<howhung;i++)
{
printf("Enter philosopher %d
position: ",(i+1)); scanf("%d",
&hu[i]); status[hu[i]]=2;
}
do
{
```

```
 printf("1.One can eat at a time\t2.Two can eat at a
time\t3.Exit\nEnter your choice:"); scanf("%d", &cho);
switch(cho)
{
case 1:
one();
break;
case 2:
two();
break;
case
3:
exit(
0);
```

```
default: printf("\nInvalid option..");
}
}while(1);
}
}
one()
{
int pos=0, x, i;
printf("\nAllow one philosopher to eat at any time\n"); for(i=0;i<howhung;
i++, pos++)
{
printf("\nP %d is granted to eat", philname[hu[pos]]);
for(x=pos;x<howhung;x++) printf("\nP %d is waiting",
philname[hu[x]]);
}
}
two()
{
int i, j, s=0, t, r, x;
printf("\n Allow two philosophers to eat at same time\n");
for(i=0;i<howhung;i++)
{
for(j=i+1;j<howhung;j++)
{
if(abs(hu[i]-hu[j])>=1&& abs(hu[i]-hu[j])!=4)
{
printf("\n\ncombination %d
\n", (s+1)); t=hu[i];
r=hu[j];
s++;
printf("\nP %d and P %d are granted to eat", philname[hu[i]],
philname[hu[j]]); for(x=0;x<howhung;x++)
{
if((hu[x]!=t)&&(hu[x]!=r))
printf("\nP %d is waiting", philname[hu[x]]);
}
} } } }
```

10.4 INPUT AND OUTPUT

```
DINING PHILOSOPHER PROBLEM
Enter the total no. of
philosophers: 5 How many
are hungry : 3
Enter philosopher 1 position: 2
Enter philosopher 2 position: 4
Enter philosopher 3 position: 5
```

OUTPUT

```
1.One  can  eat
at  a  time 2.Two
can  eat  at  a
time 3.Exit
Enter   your   choice:   1Allow   one
philosopher  to  eat  at  any  time P  3  is
granted to eat
P   3   is
```

```
waiting P
5      is
waiting P
0      is
waiting
P 5 is
granted to
eat P 5 is
waiting
P 0 is waiting
P 0 is
granted to
eat P 0 is
waiting
```

```
1.One  can  eat
at  a  time 2.Two
can  eat  at  a
time 3.Exit
Enter your choice: 2
Allow two philosophers to eat
at same time combination 1
P 3 and P 5 are
granted to eat P 0
is waiting c
ombination 2
P 3 and P 0 are
granted to eat P 5
is waiting
combination 3
P 5 and P 0 are
granted to eat P 3
is waiting
1.One  can  eat
at  a  time 2.Two
can  eat  at  a
time 3.Exit
Enter your choice: 3
```

10.5 PRE-LAB QUESTIONS

1. Differentiate between a monitor, semaphore and a binary semaphore?
2. Define clearly the dining-philosophers problem?

10.6 LAB ASSIGNMENT

1. Write a C program to simulate readers-writers problem using monitors?

10.7 POST-LAB QUESTIONS

1. Identify the scenarios in the dining-philosophers problem that leads to thedeadlock situations?

EXPERIMENT-11

11.1 OBJECTIVE

Simulate producer and consumer problem using threads (use java).

11.2 PROGRAM LOGIC

Producer algorithm:

```
while (true)
{
  /* produce an item in
  nextproduced */ while
  (counter == BUFFER_SIZE)
  ;
  /* do nothing */
  buffer[in] = next_produced;
  in = (in + 1) % BUFFER_SIZE;
  counter++;
}
```

Consumer algorithm:

```
while (true)
{
  while (counter == 0) ; /*
  do nothing */
  next_consumed =
  buffer[out];
  out = (out + 1) % BUFFER_SIZE;
  counter--;
  /* consume the item in next consumed */
}
```

11.3 PROGRAM

```
import
java.io.*;
import
java.lang.*;
class Q
{
int n;
boolean
valueSet=false;
synchronized int
get()
{
if(!valu
eSet)
try
{
wait();
}
catch(InterruptedException e){
System.out.println("InterruptedExceptio
n caught");
}
System.out.println("Got:
```

```
"+n); valueSet=false;
notif
y();
retur
n n;
}
synchronized void put(int n)
{
if(value
Set) try
{
wait();
}catch(InterruptedException e){
```

```
System.out.println("InterruptedException caught");
}
this.n=n;
valueSet=tru
e;
System.out.println("put:
"+n); notify();
}
}
class Producer implements Runnable
{
Q q;
Producer(
Q q)
{
this.q=q;
new Thread(this,"Producer").start();
}
public void run()
{
int
i=0;
while(t
rue)
{
q.put(i++);
}
}
}
class Consumer implements Runnable
{
Q q;
Consumer(Q
q)
{
this.q=q;
new Thread(this,"Consumer").start();
}
public void run()
{
while(true)
{
q.get();
}
}
}
class PCFixed
{
public static void main(String args[])
{
Q q=new  Q();
new
Producer(q);
new
Consumer(q);
System.out.println("Press ctrl+c to stop");
}
```

```
}
```

11.4 INPUT AND OUTPUTPut 1

Get1Put 2

Get 2

Put 3

Get 3

.

.

.

.

.

.

.

11.5 PRE-LAB QUESTIONS

1. What is the need for process synchronization?2.Define a semaphore?

3.Define producer-consumer problem?

11.6 LAB ASSIGNMENT

1. Write a C program to simulate producer-consumer problem using message-passing system.

11.7 POST-LAB QUESTIONS

1. Discuss the consequences of considering bounded and unbounded buffers in producer-consumer problem?
2. Can producer and consumer processes access the shared memory concurrently? If not which techniqueprovides such abenefit?

EXPERIMENT-12

12.1 OBJECTIVE

Develop a code to detect a cycle in wait-for graph

12.2 PROGRAM LOGIC

Step 1: Read the vertices and edges of a graph
Step 2: Define a method addEdge to connect the vertices
Step 3: Define a method isCyclicUtil to visit each edge and vertex of a graph **Step 4:** Define a method isCyclic to check the graph contains cycle or not
Step 5: Display whether graph forms a cycle or not

12.3 PROGRAM

```
import
java.io.*;
import
java.util.*;
class Graph
{
  private int V;
  private
  LinkedList<Integer>
  adj[]; Graph(int v) {
    V = v;
    adj = new
    LinkedList[v];
    for(int i=0;
    i<v; ++i)
      adj[i] = new LinkedList();
  }
 void addEdge(int
    v,int w) {
    adj[v].add(w);
    adj[w].add(v);
  }

 Boolean isCyclicUtil(int v, Boolean visited[], int parent)
  {
    visited[v]
    = true;
    Integer i;
    Iterator<Integer> it =
    adj[v].iterator(); while
    (it.hasNext())
    {
      i = it.next();
      if (!visited[i])
      {
        if (isCyclicUtil(i,
          visited, v))
          return true;
      }
      else if (i
        != parent)
```

```
        return
        true;
    }
    return false;
}
Boolean isCyclic()
{
    Boolean visited[] = new
    Boolean[V]; for (int i =
    0; i < V; i++)
      visited[i] = false;
    for (int u = 0; u
      < V; u++) if
      (!visited[u])
```

```
            if (isCyclicUtil(u,
              visited, -1))
              return true;
        return false;
      }
      public static void main(String args[])
      {
        Graph g1 = new
        Graph(5);
        g1.addEdge(1, 0);
        g1.addEdge(0, 2);
        g1.addEdge(2, 0);
        g1.addEdge(0, 3);
        g1.addEdge(3
        , 4); if
        (g1.isCyclic
        ())
          System.out.println("Graph
        contains cycle"); else
          System.out.println("Graph doesn't
         contains cycle"); Graph g2 = new
         Graph(3);
        g2.addEdge(0, 1);
        g2.addEdge(1
        , 2); if
        (g2.isCyclic
        ())
          System.out.println("Graph
        contains cycle"); else
          System.out.println("Graph doesn't contains cycle");
      }
    }
```

12.3 INPUT AND OUTPUT

```
javac
CheckCycle.java
javaCheckCycle

Graph contains cycle

Graph doesn't contains cycle
```

12.4 PRE-LAB QUESTIONS

1. Define resource. Give examples.
2. What is deadlock?
3. What are the conditions to be satisfied for the deadlock to occur?

12.6 LAB ASSIGNMENT

1. Implementation of resource allocation graph (RAG).

12.7 POST LAB QUESTIONS

1. What is a Safe State and what is its use in deadlock avoidance?
2. What is deadlock?
3. What are read write locks?

4. What are the necessary conditions for deadlock to occur?

EXPERIMENT-13

13.1 OBJECTIVE

Develop a code to convert virtual address to physical address

13.2 PROGRAM LOGIC

Step 1: Read all the necessary input from the keyboard.
Step 2: Pages - Logical memory is broken into fixed - sized blocks. Step 3: Frames – Physical memory is broken into fixed – sized blocks.
Step 4: Calculate the physical address using the following
Physical address = (Frame number * Frame size) + offset Step 5: Display the physical address.
Step 6: Stop the process.

13.3 PROGRAM

```
#include
<stdio.h>
#include
<conio.h>
struct pstruct
{
       int
       fno;
       int
       pbit;
}ptable[10];

int pmsize,lmsize,psize,frame,page,ftable[20],frameno;

      void info()
{
       printf("\n\nMEMORY MANAGEMENT USING
       PAGING\n\n"); printf("\n\nEnter the Size of
       Physical memory: "); scanf("%d",&pmsize);
       printf("\n\nEnter the size of
       Logical memory: ");
       scanf("%d",&lmsize);
       printf("\n\nEnter the
       partition size: ");
       scanf("%d",&psize);
       frame = (int)
       pmsize/psize; page
       = (int)
       lmsize/psize;
       printf("\nThe physical memory is divided into %d
       no.of frames\n",frame); printf("\nThe Logical
       memory is divided into %d no.of pages",page);
}
void assign()
{
       int i;
       for (i=0;i<page;i++)
```

```
{
ptable[i].fno = -1;
ptable[i].pbit= -1;
}
for(i=0; i<frame;i++)
      ftable[i] =
32555; for
(i=0;i<page;i++)
{
printf("\n\nEnter the Frame number where page %d
      must be placed: ",i);
      scanf("%d",&frameno);
      ftable[frameno
      ] = i;
      if(ptable[i].p
      bit == -1)
```

```
            {
                  ptable[i].fno =
                  frameno;
            }     ptable[i].pbit =
      }           1;
      getch(
      );
      printf("\n\nPAGE TABLE\n\n");
      printf("PageAddress FrameNo.
      PresenceBit\n\n"); for
      (i=0;i<page;i++)
            printf("%d\t\t%d\t\t%d\n",i,ptable[i].fno,ptab
      le[i].pbit); printf("\n\n\n\tFRAME TABLE\n\n");
      printf("FrameAddress PageNo\n\n");
      for(i=0;i<frame;i++)
            printf("%d\t\t%d\n",i,ftable[i]);
}
void cphyaddr()
{
      int
      laddr,paddr,disp,phyadd
      r,baddr; getch();
      printf("\n\n\n\tProcess to create the
      Physical Address\n\n"); printf("\nEnter
      the Base Address: ");
      scanf("%d",&baddr);
      printf("\nEnter
      theLogical Address: ");
      scanf("%d",&laddr);
      paddr = laddr
      / psize; disp
      = laddr %
      psize;
       if(ptable[paddr].pbit == 1 )
            phyaddr = baddr + (ptable[paddr].fno*psize) + disp;
      printf("\nThe Physical Address where the instruction present: %d",phyaddr);
}
void main()
{
      clrscr();
      info();
      assign(
      );
      cphyadd
      r();
      getch()
      ;
}
```

13.4 INPUT AND OUTPUT

```
MEMORY MANAGEMENT USING PAGING
 Enter the Size of Physical
memory: 16 Enter the size of
Logical memory: 8 Enter the
partition size: 2
The physical memory is divided into
```

```
8 no.of frames The Logical memory is
divided into 4 no.of pages
Enter the Frame number where page 0
must be placed: 5 Enter the Frame number
where page 1 must be placed: 6 Enter the
Frame number where page 2 must be
placed: 7 Enter the Frame number where
page 3 must be placed: 2 PAGE TABLE
```

PageAddress	FrameNo.	PresenceBit
0	5	1
1	6	1
2	7	1
3	2	1

```
  FRAME TABLE
   FrameAddress      PageNo
      0             32555
      1             32555
      2                3
      3             32555
      4             32555
      5              0
      6              1
      7              2
  Process to create the
   Physical Address Enter
   the Base Address: 1000
   Enter theLogical Address: 3
The Physical Address where the instruction present: 1013
```

13.5 PRE LAB QUESTION

1. Define Memory Partitioning, Paging, Segmentation.
2. In loading programs into memory, what is the difference between load-time dynamic linking and run-time dynamiclinking?
3. Give the functionalities of OS in memory management
4. Why might the direct blocks be stored in the inode itself?
5. List out memory allocation algorithms which was commonly used

13.6 LAB ASSIGNMENT

1. Implement memory allocation algorithms first fit and best fit.
2. Implement different paging techniques

13.7 POST LAB QUESTIONS

1. Define the concept of virtual memory?
2. What is the purpose of page replacement?
3. Define the general process of page replacement? 4. List out the various page replacement techniques? 5. What is page fault?

EXPERIMENT-14

14.1 OBJECTIVE

Simulate how operating system allocates frame to process.

14.2 PROGRAM LOGIC

Step 1: Read no of process,process names,size of each process,Total no of frames
Step 2: Read choice 1.Fixed Allocation 2.Proportional Allocation
Step 3: If choice is Fixed
Allocation Calculate
Allocation of frames of
each process
= Total no of process /
No of process **Step 4:** If choice is
Proportional Allocation Calculate
Allocation of frames to each process =
(size of each process/total size of processes)*total no of frames
Step 5: Display the output

14.3 PROGRAM

```
#include<std
io.h>
#include<con
io.h>    Void
main()
{
int
i,s[10],S,m,a[10],n,ch
,total=0; char p[10];
printf("enter total no
of process");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("enter the process
name %d",i)
scanf("%s",&p[i])
printf("enter the size of
each process %d",i)
scanf("%d",&s[i])
}
printf("enter total no
of frames");
scanf("%d",&m);
printf("enter ur
choice");
scanf("%d",&ch);
switch(ch)
{
case 1: printf("Fixed
Allocation");
for(i=0;i<n;i++)
{
a[i]=m/n;
printf("Allocation of frames to process %c is %d",p[i],a[i]);
}
```

```
Case 2: printf("Proportional
Allocation");
for(i=0;i<n;i++)
{
total+=s[i];
a[i]=(s[i]/t
otal)*m;
Printf("Allocation of frames for process %c is %d",p[i],a[i]);
}
default: printf("invalid choice");
}
}
```

14.4 INPUT & OUTPUT: Enter

total no of process: 4Enter

process Names: a

```
Enter the size of
each process: 5 Enter
process Names: b
Enter the size of
each process: 6 Enter
process Names: c
Enter the size of
each process: 7 Enter
process Names: d
Enter the size of
each process: 8 Enter
total No of frames:
40 Enter ur choice: 1
Fixed Allocation
Allocation of frames to
process a is 10 Allocation
of frames to process b is
10 Allocation of frames to
process c is 10 Allocation
of frames to process d is
10 Enter ur choice 2
Proportional Allocation
Allocation of frames for
process a is 7 Allocation
of frames for process b is
9 Allocation of frames for
process c is 10 Allocation
```

```
of frames for process d is
12
```

14.5 PRE-LAB QUESTIONS

1. What are the advantages of noncontiguous memory allocation schemes?

2. What is the process of mapping a logical address to physical address with respect to the paging memorymanagement technique?

3. Define the terms – base address, offset?

14.6 ASSIGNMENT QUESTIONS

1.Write a C program to simulate two-level paging technique.

2.Write a C program to simulate segmentation memory management technique

14.7 POST-LAB QUESTIONS

1.Differentiate between paging and segmentation memory allocation techniques? 2.What is the purpose of page table?

3. Whether the paging memory management technique suffers with internal or external fragmentationproblem.Why?

4. What is the effect of paging on the overall context-switching time?

.

EXPERIMENT-15

15.1 OBJECTIVE

Simulate the prediction of deadlock in operating system when all the processes announce their resource requirement in advance.

15.2 PROGRAM LOGIC

Step 1: Read values for No of processes, No of Resources , Claim Matrix,Available Matrix, Resources vector
Step 2: Calculate no of resources allocated to each process
Step 3: Check whether the claim matrix is greater than availability matrix
Step 4: if it is true NO
Allocation will be done
Else Resources will be
allocated
Step 5: Check each process which is
causing deadlock If it is found
Display deadlock causing
process Else
Requested Resources are allocated to process

15.3 PROGRAM

```
#include<std
io.h>
#include<con
io.h>   void
main()
{
int
found,flag,l,p[4][5],tp,e[4][5],i,j,k=1,m[5],r[5],
a[5],temp[5],sum=0,tr; clrscr();
pritnf("enter total no
of process");
scanf("%d",&tp);
pritnf("enter total no
of resources");
scanf("%d",&tr);
printf("enter claim
matrix");
for(i=0;i<tp;i++)
for(j=0;j<tr;j++)
{
scanf("%d",&c[i][j]);
}
printf("enter
allocation matrix");
for(i=0;i<tp;i++)
for(j=0;j<tr;j++)
{
scanf("%d",&p[i][j]);
}
printf("enter the
resource vectors");
for(i=0;i<tr;i++)
scanf("%d",&r[i]);
```

```
printf("enter the
availability matrix"));
for(i=0;i<tr;i++)
{
scanf("%d",&
r[i]);
temp[i]=a[i]
;
}
for(i=0;i<tp;i++)
{
sum=0;
for(j=0;j<t
r;j++)
{
sum+=p[i][j];
```

```
}
if(sum==0)
{
m[k]=
i;
k++;
}
for(i=1;i<=tp;i++);
{
for(l=1;l<
k;l++)
if(i!=m[l]
)
{
flag=1;
for(j=i;j<=5
;j++)
if(c[i][j]>t
emp[j])
{
flag
=0;
brea
k;
}
}
if(flag==1)
{
m[k]=i;
++;
for(j=1;j<=tr;j++
)
temp[j]+=p[i][j]
;
}
}
printf("\n deadlock causing
process are");
for(j=1;j<=tp;j++)
{
found=0;
for(i=1;i<
k;i++)
{
if(j==m[i])
found=1;
}
if(found=
=0)
printf("%
d",j);
}
}
```

15.4 INPUT AND OUTPUT:

Enter total no of process 4

Enter total no of resources 5

Enter claim matrix

```
0 1 0 0 1

0 0 1 0 1

0 0 0 0 1

1  0  1  0  1

Enter resources vector:
```

```
2 1 1 2 1

Enter

availability

vector 0 0 0 0 1

Deadlock causing process will be 2
```

15.5 PRE-LAB QUESTIONS 1.Define resource. Give examples.2.What is deadlock?

3.What are the conditions to be satisfied for the deadlock to occur?

15.6 LAB ASSIGNMENT

1. Write a C program to implement deadlock detection technique for the following scenarios?a.Single instance of each resource type.

   b.Multiple instances of each resource type.

15.7 POST-LAB QUESTIONS

1. How can be the resource allocation graph used to identify a deadlock situation?2.How is Banker‟s algorithm useful over resource allocation graph technique?

3.Differentiate between deadlock avoidance and deadlock prevention?

ADDITIONAL PROGRAMS

A. Optimal Page Replacement Algorithm :

1. OBJECTIVE

Program to perform Optimal page replacement algorithm.

2. PROGRAM

```
#include<con
io.h>
#include<std
io.h>
int optimalframe(int
w[],int m); void
main()
{
int i,j,n,m,r,pagef,l,h,p;
int a[100],b[100],w[10];
pagef=
0;
clrscr
();
printf("enter the number
of pages\n");
scanf("%d",&n);
printf("enter the number of frames in
main memory\n"); scanf("%d",&m);
for(i=0;i<m;i++)
   b[i]=-1;
for(i=0;i<n;i++)
     {
          printf("enter the
          element %d\n",i);
          scanf("%d",&a[i]);
     }
     for(i=0;i<m;i++)
     {
          b[i]=a[
      i];
      for(r=0;r<m     printf("%d ",b[r]);
      ;r++)
                printf("
                \n");
     }          pagef++;
     for(i=m;i<n
     ;i++)
     {
           for(j=0;j<m;j++)
           {
                if(b[j]==a[i])
                {
                printf("the page %d is already present in
                      main memory\n",a[i]); break;
```

```
        }
}
if(j=
=m)
{               for(l=0;l<m;l++)
                {
                        for(p=i+1;p<n;p++)
                        {
```

```
                                    if(b[l]==a[p])
                                {
                                    w[l]=
                                    p;
                                }   break
                            }       ;
                            if(p==
                            n)
                            {
                            b[l]=a
                            [i];
                        }}  break;
                        if(l=
                        =m)
                        {
                            h=optimalframe(
                            w,m);
                        }   b[h]=a[i];
                for(r=0;r<m
                ;r++)
                    printf("%d ",b[r]);
            printf("\n");
                pagef++;
             }
        }
        printf("number of page
        faults: %d",pagef); getch();
        }
        int optimalframe(int w[],int m)
        {
            int
            max,k,h=0;
            max=w[0];
            for(k=1;k<m;    if(w[k]>max)
            k++)            {
                                max=w[k
                                ]; h=k;
                            }


             return h;
        }
```

3. INPUT AND OUTPUT

```
enter the number of
pages 20
enter the number of frames in
main memory 3
enter the
element 0 7
enter the
element 1 0
enter the
element 2 1
```

```
enter the
element 3 2
enter the
element 4 0
enter the element 5
```

```
3
enter the
element 6 0
enter the
element 7 4
enter the
element 8 2
enter the
element 9 3
enter the
element 10 0
enter the
element 11 3
enter the
element 12 2
enter the
element 13 1
enter the
element 14 2
enter the
element 15 0
enter the
element 16 1
enter the
element 17 7
enter the
element 18 0
enter the
element 19 1
7  -1  -1
7  0 -1
7  0  1
2  0  1
the page 0 is already present in
main memory 2  0  3
the page 0 is already present in
main memory 2  4  3
the page 2 is already present in
main  memory  the  page  3  is
already present in main memory 2
0  3
the page 3 is already present in
main  memory  the  page  2  is
already present in main memory 2
0  1
the page 2 is already present in
main  memory  the  page  0  is
already present in main memory
the page 1 is already present in
main memory 7  0  1
the page 0 is already present in
main  memory  the  page  1  is
already present in main memory
number of page faults: 9
```

B. Dekker's Algorithm :

1. OBJECTIVE

Program to perform Dekker"s algorithm.

2. PROGRAM

```
# include <stdio.h>
#
include<pthrea
d.h> #
include<cursor
s.h>
int turn=0,
flag[2]={0,0},bal=0;
void longdelay()
{
long int t1,t2;
for(t1=0;t1<64000;t1++);
for(t2=0;t2<64000;t2++);
}
void shortdelay()
{
int t3,t4;
for(t3=0;t3<32000
;t3++)
for(t4=0;t4<32000
;t4++)
}
void test(int i)
{
int
```

```
j,k,m,p,q,r
,c; j=1-i;
for(k=0;k<3
;k++)
{
flag[i]=1;
while(flag[j
])
```

```
{
if (turn==j)
{
flag[i]=0;
printf("Iam
waiting:%d",i);
while(turn==j);
flag[i]=1;
}
}
longdelay();
printf("Iam
entering %d",i);
c=bal;
/*critical
section*/
printf("\n
process: %d",l);
printf("flag[%d]=%d,
flag[%d]=%d",i,flag[i],j,flag[j]);
printf("turn=%d\n",turn);
bal=c+1000;
printf("the balance
is %d",bal); turn=j;
flag[i]=0;
printf("Iam
exiting: %d",i);
shortdelay();
}
}
int main()
```

```
{
pthread_t t1,t2;
pthread_creat(&t,NULL,(void*)&test,(voi
d*)0); printf("After creation of first
thread…..");
pthread_creat(&t2,NULL,(void*)&test,(void
*)1); pthread_join(t1,NULL);
```

```
pthread_join(t2,
NULL); sleep(s);
printf("Parent
terminated:%d",bal);
return(0); }
```