Shram Sadhana Bombay Trust's
COLLEGE OF ENGINEERING AND TECHNOLOGY,
BAMBHORI POST BOX NO. 94, JALGAON – 425001.(M.S.)
Included under section 2 (f) & 12 (B) of the UGC Act, 1956
ISO 9001: 2015 certified
Phone No. (0257) 2258393, Fax No. (0257) 2258392
Website- www.sscoetjalgaon.ac.in
Email: sscoetjal@gmail.com

ISO 9001:2015

DEPARTMENT OF COMPUTER ENGINEERING

Laboratory Manual

Class: T.E. Computer / IT

Subject: Operating Systems Lab

DEPARTMENT OF COMPUTER ENGINEERING

Vision of the Department

To emerge as the leading Computer Engineering department for inclusive development of students.

Mission of the Department

To provide student-centered conducive environment for preparing knowledgeable, competent and value-added computer engineers.

DEPARTMENT OF COMPUTER ENGINEERING

Programme Educational Objectives

PEO 1. Core Knowledge -Computer engineering graduates will have the knowledge of basic science and Engineering skills, Humanities, social science, management and conceptual and practical understanding of core computer engineering area with project development.

PEO 2. Employment/ Continuing Education - Computer engineering graduates will have the knowledge of Industry-based technical skills to succeed in entry level engineering position at various industries as well as in academics.

PEO 3. Professional Competency - Computer engineering graduates will have the ability to communicate effectively in English, to accumulate and disseminate the knowledge and to work effectively in a team with a sense of social awareness.

\

DEPARTMENT OF COMPUTER ENGINEERING

Programme Outcomes

Engineering Graduates will be able to:

- **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
- **Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
- **Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
- **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.
- **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
- **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
- **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

- **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
- **Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
- **Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
- **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.
- **Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Programme Specific Outcomes

Computer Engineering Graduates will be able to::

- **Software Systems Development:** Apply the theoretical concepts of computer engineering and practical knowledge in analysis, design and development of software systems.
- **Open Source Software:** Demonstrate familiarity and practical competence with a broad range of programming languages and open source platforms
- **Computer Proficiency:** Exhibit proficiency through latest technologies in demonstrating the ability for work efficacy to the industry & society

Operating Systems Lab

Course Outcomes

Upon successful completion of lab Course, student will be able to:

1. Apply concept of file handling and process scheduling.
2. Identify problems of deadlock and semaphore.
3. Apply concept of memory management.
4. Design a file allocation and organization techniques.
5. Solve disk scheduling algorithm.

Mapping between CO's of O.S.L. and PO's and PSO's

CO	PO 1	PO 2	PO 3	PO 4	PO 5	PO 6	PO 7	PO 8	PO 9	PO1 0	PO11 1	PO11 2	PSO 1	PSO 2	PSO 3
Apply concept of file handling and process scheduling	3	3	3	3	2	2	1	1	2	1	2	2	3	3	2
Identify problems of deadlock and semaphore	2	2	2	2	3	1	1	1	3	1	2	3	3	3	3
Apply concept of memory manageme nt	3	3	3	3	2	3	2	1	3	2	3	2	3	3	2
Design a file allocation and organizatio n techniques	3	3	3	3	2	2	1	1	2	1	2	2	3	3	2
Solve disk scheduling algorithm	2	2	2	2	3	1	1	1	3	1	3	2	3	3	3

SSBT's College of Engineering & Technology, Bambhori, Jalgaon

Computer Engineering Department

Name of student:

Date of Performance: Date of Completion:

EXPERIMENT NO. 1

TITLE: - File and directory management.

AIM: - To Perform various file and directory operations.

HARDWARE / SOFTWARE REQUIREMENTS: - Turbo C, PC, Mouse.

THEORY: -

In case C or C++ there are some functions which are used for calling an interrupt service.

General DOS interrupt interfaces -

Declaration:

int intdos(union REGS *inregs, union REGS *outregs);

Remarks:

intdos execute DOS interrupt 0x21 to invoke a specified DOS function. The value of inregs -> h.ah specifies the DOS function to be invoked.

Interrupts which are used in program –

1) Function 3CH (INT 21H) – create a file.
2) Function 41H (INT 21H) – delete a file.
3) Function 56H (INT 21H) – rename a file.
4) Function 3FH (INT 21H) – read a file.
5) Function 40H (INT 21H) – write a file.
6) Function 39H (INT 21H) – create a directory.
7) Function 3AH (INT 21H) – delete a directory.
8) Function 3BH (INT 21H) – set working directory.

File Management operations -

- Create File
- Delete File
- Rename File
- Read File
- Write File
- Exit File

Directory Management operations-

- Create Directory
- Remove Directory
- Set Working Directory
- Exit

ALGORITHM: -

For creating a file -

1) declare file pointer.
2) store 0x3C in ah register.
3) get offset of file pointer in dx register.
4) if carry is generated
   then error
   else file is created successfully.

For deleting a file –

1) declare file pointer.
2) store 0x41 in ah register.
3) get offset of file pointer in dx register.
4) if carry is generated
   then error
   else file is deleted successfully.

For rename a file –

1) open old file.
2) open new file.
3) store 0x56 in ah register.
4) get offset of file pointer in dx register of old file.
5) get offset of file pointer in di register of new file.
6) if carry is generated
   then error
   else file is renamed successfully.

For read a file –

1) declare file pointer.
2) store 0x3F in ah register.
3) get offset of file pointer in dx register.
4) if carry is generated
   then error
   else shows number of bytes actually read.

For write a file –

1) declare file pointer.
2) store 0x40 in ah register.
3) get offset of file pointer in dx register.
4) if carry is generated
   then error
   else shows number of bytes actually written.

For creating a directory -

1) declare file pointer.
2) store 0x39 in ah register.
3) get offset of file pointer in dx register.
4) if carry is generated
   then error
   else directory is created successfully.

For deleting a directory

1) declare file pointer.
2) store 0x3A in ah register.
3) get offset of file pointer in dx register.
4) if carry is generated
   then error
   else directory is deleted successfully.

For set working directory

1) declare file pointer.
2) store 0x3B in ah register.
3) get offset of file pointer in dx register.
4) if carry is generated
   then error
   else directory path is changed.

RESULT: -

REFERENCES: -

1. Advanced MSDOS Programming, 2nd edition By Ray Duncan.
2. Operating System Concepts, 6th edition By Abraham Silberschatz, Peter Baer Galvin.

QUESTIONS FOR VIVA: -

1) Define Operating System. Explain different Operating System services.

2) What is system call? Explain with example.

3) What is Command interpreter?

4) What is file? Explain various file attributes.

5) Explain any three DOS interrupts.

Name & Sign of Teacher

```
#include<stdio.h>
#include<stdlib.h>
void main()
{
 FILE *fp1,*fp2;
 char ch,data;
 int choice, status;
 do{
  printf("\nFile Handling Operations");
  printf("\nPress 1 to create a file");
  printf("\nPress 2 to open a file");
  printf("\nPress 3 to write data into a file");
  printf("\nPress 4 to read a file");
  printf("\nPress 5 to append data to a file");
  printf("\nPress 6 to copy a file");
  printf("\nPress 7 to rename a file");
  printf("\nPress 9 to exit");
  printf("\nEnter your choice");
  scanf("%d",&choice);
  switch(choice)
  {
   case 1: fp1 = fopen("testfile.txt", "w");
        if (fp1 == NULL)
        {
            printf("Error creating file\n");
        }
        printf("File created successfully\n");
        fclose(fp1);
        break;

   case 2: fp1 = fopen("testfile.txt", "r");
        if (fp1 == NULL)
        {
            printf("Error opening file\n");
          }
        printf("File opened successfully\n");
        break;

   case 3: fp1 = fopen("testfile.txt", "w"); // opening a file named test.txt
        if (fp1 == NULL)
        {
            printf("Error!");
        }
      fprintf(fp1, "Welcome to C programming\n");
        fprintf(fp1, "This is a sample program\n");
      printf("Data written successfully\n");
        fclose(fp1);
        break;

      case 4: fp1 = fopen("testfile.txt", "r"); // opening a file named test.txt
        if (fp1 == NULL)
        {
            printf("Error!");
        }
      printf("The contents of file are :\n");
```

```
      while ((ch = fgetc(fp1)) != EOF)
      {
         printf("%c", ch);
      }
     fclose(fp1);
     break;

   case 5:fp1=fopen("testfile.txt","a"); //opening file test.txt
     if(fp1==NULL)
      {
         printf("Error!");
      }
      fputs("This data is appended to existing data\n",fp1);
      printf("Data appended successfully\n");
      fclose(fp1);
      break;

     case 6: fp1 = fopen("testfile.txt", "r"); //opening file test.txt
      if (fp1 == NULL)
       {
         printf("Error!");
      }
      fp2 = fopen("test_copy.txt", "w"); //creating file test_copy.txt
      if (fp2 == NULL)
      {
         printf("Error!");
      }
     while((ch = fgetc(fp1)) != EOF)
      {
         fputc(ch, fp2);
      }
      printf("File copied successfully\n");
      fclose(fp1);
      fclose(fp2);
      break;

     case 7: status = rename("testfile.txt", "test_renamed.txt"); //renaming file test.txt to test_renamed.txt
      if (status == 0)
      {
         printf("File renamed successfully\n");
      }
      else
      {
         printf("Unable to rename the file\n");
      }
      break;
     case 9: exit(0);
     break;
     default: printf("Enter valid choice");
    }
 }while(choice<10);
}

//Output
File Handling Operations
```

Press 1 to create a file
Press 2 to open a file
Press 3 to write data into a file
Press 4 to read a file
Press 5 to append data to a file
Press 6 to copy a file
Press 7 to rename a file
Press 9 to exit
Enter your choice1
File created successfully

File Handling Operations
Press 1 to create a file
Press 2 to open a file
Press 3 to write data into a file
Press 4 to read a file
Press 5 to append data to a file
Press 6 to copy a file
Press 7 to rename a file
Press 9 to exit
Enter your choice2
File opened successfully

File Handling Operations
Press 1 to create a file
Press 2 to open a file
Press 3 to write data into a file
Press 4 to read a file
Press 5 to append data to a file
Press 6 to copy a file
Press 7 to rename a file
Press 9 to exit
Enter your choice3
Data written successfully

File Handling Operations
Press 1 to create a file
Press 2 to open a file
Press 3 to write data into a file
Press 4 to read a file
Press 5 to append data to a file
Press 6 to copy a file
Press 7 to rename a file
Press 9 to exit
Enter your choice4
The contents of file are :
Welcome to C programming
This is a sample program

File Handling Operations
Press 1 to create a file
Press 2 to open a file
Press 3 to write data into a file
Press 4 to read a file
Press 5 to append data to a file

Press 6 to copy a file
Press 7 to rename a file
Press 9 to exit
Enter your choice6
File copied successfully

File Handling Operations
Press 1 to create a file
Press 2 to open a file
Press 3 to write data into a file
Press 4 to read a file
Press 5 to append data to a file
Press 6 to copy a file
Press 7 to rename a file
Press 9 to exit
Enter your choice7
Unable to rename the file

File Handling Operations
Press 1 to create a file
Press 2 to open a file
Press 3 to write data into a file
Press 4 to read a file
Press 5 to append data to a file
Press 6 to copy a file
Press 7 to rename a file
Press 9 to exit
Enter your choice9

SSBT's College of Engineering & Technology, Bambhori, Jalgaon

Computer Engineering Department

Name of student:

Date of Performance: Date of Completion:

EXPERIMENT NO. 2

TITLE: - CPU Scheduling Algorithm.

AIM: - Implementation of CPU Scheduling Algorithms - FCFS, SJF, Priority based, RR scheduling algorithms.

HARDWARE / SOFTWARE REQUIREMENTS: - Turbo C, PC, Mouse.

THEORY: -

Scheduling –

Scheduling refers to the way processes are assigned to run on the available CPU's. This assignment is carried out by software's known as scheduler next process to run. A scheduler is an OS module that selects the next job to be admitted into the system and next process to run.

When more than one process is run able, the operating system must decide which one first. The part of the operating system concerned with this decision is called the scheduler, and algorithm it uses is called the scheduling algorithm.

Scheduling criteria -

1) CPU Utilization –keep the CPU as busy as possible.

2) Throughput – processes that complete their execution per time unit.

3) Turnaround time –amount of time to execute a particular process.

4) Waiting time –amount of time a process has been waiting in the ready queue.

Response time – amount of time it takes from when a request was submitted until the first response is produced, not output (for time-sharing environment).

The Scheduling algorithms can be divided into two categories with respect to how they deal with clock interrupts.

1) Nonpreemptive Scheduling

A non-preemptive means that a running process retains the control of the CPU and all the allocated resources until it surrenders controls to the OS on its own. This means that even if a higher priority process enters the system, the running process cannot be enforced to give up the control. This philosophy is better suited for real time system, where higher priority events need an immediate attention and therefore need to interrupt the currently running process.

2) Preemptive Scheduling

A preemptive means on the other hand allows a higher priority process to replace a currently running process even if its time slice is not over it has not requested for any I/O. this requires context switching more frequently, thus reducing the throughput, but then it is better suited for online, time processing where interactive users and high priority processes require immediate attention.

Scheduling Algorithms -

1] First-Come-First-Served (FCFS) Scheduling –

First-Come-First-Served algorithm is the simplest scheduling algorithm is the simplest scheduling algorithm. Processes are dispatched according to their arrival time on the ready queue. Being a non preemptive discipline, once a process has a CPU, it runs to completion.

FCFS is more predictable than most of other schemes since it offers time. FCFS scheme is not useful in scheduling interactive users because it cannot guarantee good response time. The code for FCFS scheduling is simple to write and understand.

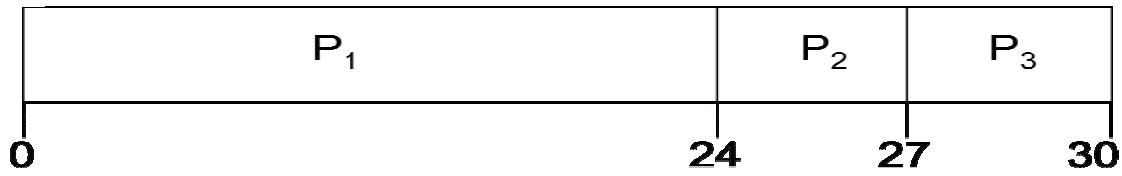
The First-Come-First-Served algorithm is rarely used as a master scheme in modern operating systems but it is often embedded within other schemes.

Example:

Process	Burst Time
P_1	24
P_2	3
P_3	3

Assume processes arrive as: P_1 , P_2 , P_3

The Gantt Chart for the schedule is:



Waiting time for $P_1 = 0$; $P_2 = 24$; $P_3 = 27$
Average waiting time: $(0 + 24 + 27)/3 = 17$

2] Shortest-Job-First (SJF) Scheduling –

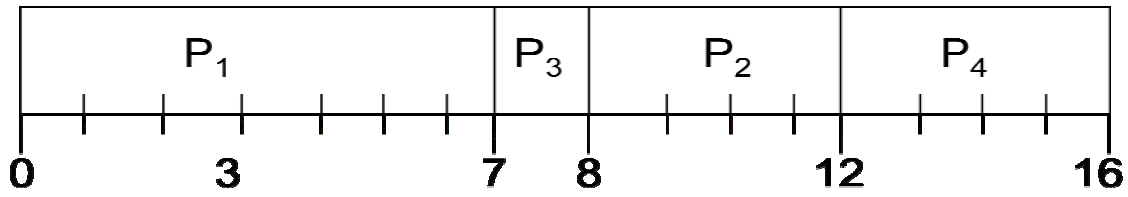
Shortest-Job-First (SJF) is a non-preemptive discipline in which waiting job (or process) with the smallest estimated run-time-to-completion is run next. In other words, when CPU is available, it is assigned to the process that has smallest next CPU burst.

The SJF scheduling is especially appropriate for batch jobs for which the run times are known in advance. Since the SJF scheduling algorithm gives the minimum average time for a given set of processes, it is probably optimal.

Example of Non-Preemptive SJF -

Process	Arrival Time	Burst Time
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4

SJF (non-preemptive)

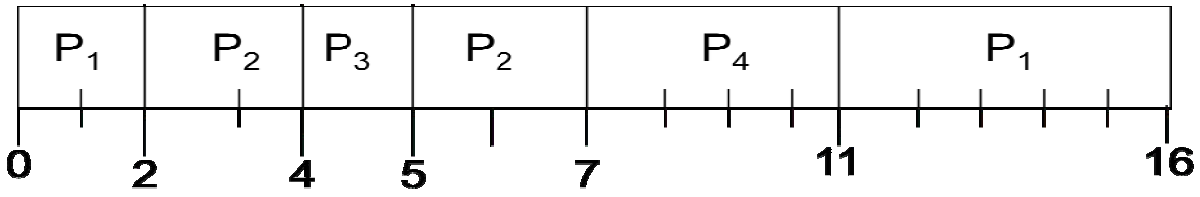


Average waiting time = $(0 + 6 + 3 + 7)/4 = 4$

Example of Preemptive SJF –

Process	Arrival Time	Burst Time
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4

SJF (preemptive)



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

3] Priority Scheduling -

The basic idea is straightforward: each process is assigned a priority, and priority is allowed to run. Equal-Priority processes are scheduled in FCFS order. The shortest-Job-First (SJF) algorithm is a special case of general priority scheduling algorithm.

An SJF algorithm is simply a priority algorithm where the priority is the inverse of the (predicted) next CPU burst. That is, the longer the CPU burst, the lower the priority and vice versa.

4] Round Robin Scheduling –

Round-robin (RR) is one of the simplest scheduling algorithms for processes in an operating system. As the term is generally used, time slices are assigned to each process in equal portions and in circular order, handling all processes without priority (also known as cyclic executive). Round-robin scheduling is simple, easy to implement, and starvation-free. Round-robin scheduling can also be applied to other scheduling problems, such as data packet scheduling in computer networks. The name of the algorithm comes from the round-robin principle known from other fields, where each person takes an equal share of something in turn.

Example:

Time Quantum = 20

Process	Burst Time
P1	53
P2	17
P3	68
P4	24

Gantt Chart for Schedule

Typically, higher average turnaround time than SRTF, but better response.

ALGORITHM: -

1) include required header files.
2) define structure of process and create the array of structure variable.
3) write a main( ) function
   I. declaration of variable
   II. create a menu as
      1. FCFS
      2. SJF
      3. Priority
      4. Round Robin
      5. Exit
   III. if choice = 1
   then call FCFS( ) function
      i. read the number of processes and burst time of each process.
      ii. calculate waiting time, Waiting Time = Waiting Time – Arrival Time
         and Turnaround time = Waiting time + Burst Time
      iii. calculate average waiting time and average turnaround time.
      iv. print the Waiting time and Turnaround time of each process and also print average waiting time and average turnaround time.

   IV. if Choice = 2
      then call SJF( ) function
      i. read the number of processes and burst time of each process.
      ii. if P[i].Bt > P[i+1].Bt
         then swap P[i]& P[i+1]
      iii. calculate waiting time, Waiting Time = Waiting Time – Arrival Time
         and Turnaround time = Waiting time + Burst Time
      iv. calculate average waiting time and average turnaround time.
      v. print the Waiting time and Turnaround time of each process and also print average waiting time and average turnaround time.

   V. If Choice = 3
      then call Priority( ) function

i. read the number of process and burst time for process.
ii. if P[i].pri > p[i+1].pri
then swap P[i] & P[i+1]
iii. calculate waiting time, Waiting Time = Waiting Time – Arrival Time
and Turnaround time = Waiting time + Burst Time
iv. calculate average waiting time and average turnaround time.
v. print the Waiting time and Turnaround time of each process and also print average waiting time and average turnaround time.

VI. If Choice =4
then call RoundRobin( ) function
i. read the number of processes and burst time of each process.
ii. read time quantum.
iii. if p[i].Bt > 0
then If p[i].Bt > quantum
allocate the process for execution as per round robin scheduling algorithm.
iv. calculate waiting time Wt Time = [Wt Time – Arrival Time]
and Turnaround time Tat = [Wt time + Burst Time]
iii. calculate average waiting time and average turnaround time
iv. print the Waiting time and Turnaround time of each process and also print average waiting time and turnaround time.

VII. if choice = 5
then terminate the program

RESULT: -

REFERENCES: -

1. Operating System Concepts, 6th edition By Abraham Silberschatz, Peter Baer Galvin.

QUESTIONS FOR VIVA: -

1) What is process? Explain process management.

2) What is scheduling? Explain the types of Schedulers.

3) What is multitasking?

4) What is Throughput, Turnaround time, Waiting time and Response time?

5) What is meant by CPU burst and I/O burst?

Name & Sign of Teacher

```
//PROGRAM FOR 'CPU SCHELUDING' IN 'OS'
#include<stdio.h>

typedef struct process
{
  int bt,wt,tt,no,prio;
}prcs;
int main()
{ prcs p[10],t;
  float avg_w,avg_t;
  int i,j,k,ch,n,qntm,c;
//   //clrscr();();
  do
  {  printf("\n---------- MENU ----------\n");
     printf("\n\n1-FCFS");
     printf("\n2-SJF");
     printf("\n3-PRIORITY SCHEDULING");
     printf("\n4-ROUND ROBIN");
     printf("\n5-EXIT");
     printf("\nEnter ur choice:\t");
     scanf("%d",&ch);
     switch(ch)
     {   case 1:
    printf("\nEnter no. of processes:\t");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {  printf("Enter burst time for P%d:  ",i+1);
       scanf("%d",&p[i].bt);
       p[i].prio=0;
       p[i].no=i+1;
    }
    p[0].wt=0;
    for(i=1;i<n;i++)
    p[i].wt=p[i-1].wt+p[i-1].bt;
    for(i=0;i<n;i++)
      p[i].tt=p[i].wt+p[i].bt;
    ////clrscr();();
    printf("\nGantt Chart is:");
    printf("\n0 ");
    for(i=0;i<n;i++)
      printf("P%d %d ",p[i].no,p[i].tt);
    printf("\n\nProcess   BT WT TT");
    for(i=0;i<n;i++)
     printf("\n%d   %d %d    %d",p[i].no,p[i].bt,p[i].wt,p[i].tt);
    avg_w=0;
    avg_t=0;
    for(i=0;i<n;i++)
    {  avg_w=avg_w+p[i].wt;
       avg_t=avg_t+p[i].tt;
    }
    avg_w=(float)avg_w/n;
    avg_t=(float)avg_t/n;
    printf("\n\nAverage waiting time is: %.2f ",avg_w);
    printf("\nAverage turn around time is: %.2f ",avg_t);
  break;
```

```
case 2:
 printf("\nEnter no. of processes:\t");
 scanf("%d",&n);
 for(i=0;i<n;i++)
 {  printf("Enter execution time for p%d:  ",i+1);
    scanf("%d",&p[i].bt);
    p[i].prio=0;
    p[i].no=i+1;
 }
 for(i=0;i<n;i++)
 {  for(j=i+1;j<n;j++)
    {  if(p[i].bt>p[j].bt)
{  t=p[i];
   p[i]=p[j];
   p[j]=t;
}
    }
 }
 p[0].wt=0;
 for(i=1;i<n;i++)
   p[i].wt=p[i-1].wt+p[i-1].bt;
 for(i=0;i<n;i++)
   p[i].tt=p[i].wt+p[i].bt;
 //clrscr();();
 printf("\nGantt Chart is:");
 printf("\n0 ");
 for(i=0;i<n;i++)
  printf("P%d %d ",p[i].no,p[i].tt);
 for(i=0;i<n;i++)
 {  for(j=i+1;j<n;j++)
    {  if(p[i].no>p[j].no)
{  t=p[i];
   p[i]=p[j];
   p[j]=t;
}
    }
 }
 printf("\n\nProcess   BT WT TT");
 for(i=0;i<n;i++)
   printf("\n%d   %d  %d   %d",p[i].no,p[i].bt,p[i].wt,p[i].tt);
 avg_w=0;
 avg_t=0;
 for(i=0;i<n;i++)
 {  avg_w=avg_w+p[i].wt;
    avg_t=avg_t+p[i].tt;
 }
 avg_w=(float)avg_w/n;
 avg_t=(float)avg_t/n;
 printf("\n\nAverage waiting time is: %.2f ",avg_w);
 printf("\nAverage turn around time is: %.2f ",avg_t);
 break;
case 3:
  printf("\nEnter no. of processes:\t");
  scanf("%d",&n);
  for(i=0;i<n;i++)
```

```
  {  printf("Enter burst time for p%d: ",i+1);
     scanf("%d",&p[i].bt);
     printf("Enter priority for p%d: ",i+1);
     scanf("%d",&p[i].prio);
     p[i].no=i+1;
  }
  for(i=0;i<n;i++)
  {  for(j=i+1;j<n;j++)
     {  if(p[i].prio>p[j].prio)
 {  t=p[i];
    p[i]=p[j];
    p[j]=t;
 }
     }
  }
  p[0].wt=0;
  for(i=1;i<n;i++)
    p[i].wt=p[i-1].wt+p[i-1].bt;
  for(i=0;i<n;i++)
    p[i].tt=p[i].wt+p[i].bt;
  ////clrscr();();
  printf("\nGantt Chart is:");
  printf("\n0 ");
  for(i=0;i<n;i++)
   printf("P%d %d ",p[i].no,p[i].tt);
  for(i=0;i<n;i++)
  {  for(j=i+1;j<n;j++)
     {  if(p[i].no>p[j].no)
 {  t=p[i];
    p[i]=p[j];
    p[j]=t;
 }
     }
  }
  printf("\n\nProcess   BT WT TT Priority");
  for(i=0;i<n;i++)
  printf("\n%d   %d   %d  %d  %d",p[i].no,p[i].bt,p[i].wt,p[i].tt,p[i].prio);
  avg_w=0;
  avg_t=0;
  for(i=0;i<n;i++)
  {  avg_w=avg_w+p[i].wt;
     avg_t=avg_t+p[i].tt;
  }
  avg_w=(float)avg_w/n;
  avg_t=(float)avg_t/n;
  printf("\n\nAverage waiting time is: %.2f ",avg_w);
  printf("\nAverage turn around time is: %.2f ",avg_t);
break;
case 4:
   printf("\nEnter no. of processes:\t");
   scanf("%d",&n);
   printf("\nEnter Time Quantum:\t");
   scanf("%d",&qntm);
   for(i=0;i<n;i++)
   {  printf("Enter burst time for p%d:  ",i+1);
```

```
        scanf("%d",&p[i].bt);
        p[i].prio=0;
        p[i].no=p[i].bt;
        p[i].tt=0;
        p[i].wt=0;
     }
     //clrscr();();
     printf("\nGantt Chart is:");
     c=0;i=0;j=0;
     printf("\n0");
     while(c<n)
     {   if(p[i].bt>0)
  {  if(p[i].bt<qntm)
     {  p[i].wt=p[i].wt+(j-p[i].tt);
        c++;
        j+=p[i].bt;
        p[i].tt=j;
        p[i].bt=0;
        printf(" p%d %d",i+1,p[i].tt);
     }
     else
     {  p[i].bt-=qntm;
        p[i].wt=p[i].wt+(j-p[i].tt);
        j+=qntm;
        p[i].tt=j;
        printf(" p%d %d",i+1,p[i].tt);
        if(p[i].bt==0)
        c++;
     }
  }
  if(i==n-1)
    i=0;
  else
    i++;
    }
    for(i=0;i<n;i++)
    {  p[i].bt=p[i].no;
       p[i].no=i+1;
    }
    printf("\n\nProcess   BT WT TT");
    for(i=0;i<n;i++)
  printf("\n%d     %d %d    %d",p[i].no,p[i].bt,p[i].wt,p[i].tt);
    avg_w=0;
    avg_t=0;
    for(i=0;i<n;i++)
    {  avg_w=avg_w+p[i].wt;
       avg_t=avg_t+p[i].tt;
    }
    avg_w=(float)avg_w/n;
    avg_t=(float)avg_t/n;
    printf("\n\nAverage waiting time is: %.2f ",avg_w);
    printf("\nAverage turn around time is: %.2f ",avg_t);
 break;
 }
    }while(ch!=5);
```

```
   return 0;
}

//Output
---------- MENU ----------


1-FCFS
2-SJF
3-PRIORITY SCHEDULING
4-ROUND ROBIN
5-EXIT
Enter ur choice: 1
Enter no. of processes: 5
Enter burst time for P1:  12
Enter burst time for P2:  13
1Enter burst time for P3:  14
Enter burst time for P4:  15
Enter burst time for P5:  16
Gantt Chart is:
0 P1 12 P2 25 P3 39 P4 54 P5 70

Process   BT WT TT
1   12 0    12
2   13 12    25
3   14 25    39
4   15 39    54
5   16 54    70

Average waiting time is: 26.00
Average turn around time is: 40.00
---------- MENU ----------


1-FCFS
2-SJF
3-PRIORITY SCHEDULING
4-ROUND ROBIN
5-EXIT
Enter ur choice: 2
Enter no. of processes: 1
Enter execution time for p1:  0
1Gantt Chart is:
0 P1 0

Process   BT WT TT
1   0  0   0

Average waiting time is: 0.00
Average turn around time is: 0.00
---------- MENU ----------


1-FCFS
2-SJF
```

3-PRIORITY SCHEDULING
4-ROUND ROBIN
5-EXIT
Enter ur choice:3
3
Enter no. of processes: 1
Enter burst time for p1: 2
Enter priority for p1: 3
Gantt Chart is:
0 P1 2

Process BT WT TT Priority
1 2 0 2 3

Average waiting time is: 0.00
Average turn around time is: 2.00
---------- MENU ----------

1-FCFS
2-SJF
3-PRIORITY SCHEDULING
4-ROUND ROBIN
5-EXIT
Enter ur choice: 4
Enter no. of processes: 3
Enter Time Quantum: 1
2Enter burst time for p1:
3
Enter burst time for p2: 2
Enter burst time for p3: 4
Gantt Chart is:
0 p1 1 p2 2 p3 3 p1 4 p2 5 p3 6 p1 7 p3 8 p3 9

Process BT WT TT
1 3 4 7
2 2 3 5
3 4 5 9

Average waiting time is: 4.00
Average turn around time is: 7.00
---------- MENU ----------

1-FCFS
2-SJF
3-PRIORITY SCHEDULING
4-ROUND ROBIN
5-EXIT
Enter ur choice:5

SSBT's College of Engineering & Technology, Bambhori, Jalgaon

Computer Engineering Department

Name of student:

Date of Performance: Date of Completion:

EXPERIMENT NO. 3

TITLE: - Memory Management.

AIM: - Implementation of various Memory Management strategies - First Fit, Best Fit, Next Fit and Worst Fit.

HARDWARE / SOFTWARE REQUIREMENTS: - Turbo C, PC, Mouse.

THEORY: -

Memory management is the act of managing computer memory. The essential requirement of memory management is to provide ways to dynamically allocate portions of memory to programs at their request, and freeing it for reuse when no longer needed. This is critical to the computer system.

Several methods have been devised that increase the effectiveness of memory management. Virtual memory systems separate the memory addresses used by a process from actual physical addresses, allowing separation of processes and increasing the effectively available amount of RAM using paging or swapping to secondary storage. The quality of the virtual memory manager can have a big impact on overall system performance. Memory is usually divided into fast primary storage and slow secondary storage. Memory management in the operating system handles moving information between these two levels of memory.

The task of fulfilling an allocation request consists of finding a block of unused memory of sufficient size. Even though this task seems simple, several issues make the implementation complex. One of such problems is internal and external fragmentation, which arises when there are many small gaps between allocated memory blocks, which are insufficient to fulfill the request. Another is that allocator's metadata can inflate the size of (individually) small allocations; this effect can be reduced by chunking. Usually, memory is allocated from a large pool of unused memory area called the heap (also called the free store). Since the precise location of the allocation is not known in advance, the memory is accessed indirectly, usually via a pointer reference. The precise algorithm used to organize the memory area and allocate and deallocate chunks is hidden behind an abstract interface and may use any of the methods described below.

Memory Allocation Strategies –

1) First Fit – Allocate the first hole that is big enough. Searching can start at the beginning of the set of holes. We can stop searching as such as we find a free hole that is large enough.

2) Best Fit – Allocate the smallest hole that is big enough we must search entire list, unless the list is ordered by size. This strategy produces the smallest leftover hole.

3) Next Fit – For allocation searching can start from the previous first fit search ended or next the hole is allocated after the previously allocated hole.

4) Worst Fit – Allocate the largest hole. Again we must search the entire list, unless it is sorted by size, this strategy produces the largest leftover hole, which may be more useful than the smaller leftover hole from a best fit approach.

ALGORITHM: -

1) read the number of blocks.
2) read the size of each block.
3) read the requested blocks.
4) read the size of each requested block.
5) read the users choice
   if choice = 1
   then search for the free hole which will be available very firstly
   else if choice = 2
   then start to find out the highest fit hole
   else if choice = 3
   then from next position start to find out next free hole
   else if choice = 4
   then allocate the hole having the largest size
   else if choice = 5
   then terminate the program.

RESULT: -

REFERENCES: -

1. Operating System Concepts, 6th edition By Abraham Silberschatz, Peter Baer Galvin.

QUESTIONS FOR VIVA: -

1) What is Memory Management?

2) Explain Contiguous and Non – Contiguous memory allocation.

3) Explain segmentation and paging concepts.

4) Explain memory allocation policies.

5) What is Virtual Memory?

Name & Sign of Teacher

```
//Program for memory  management methods ,Experiment No. 3.
#include<conio.h>
#include<stdio.h>
void main()
{
int i,j,n,t,size[10],req[10],nr,min,f,w=1,l;
int ch,fp[10],fm[10],fn[10],intfrg,k[10];
clrscr();
printf("\nEnter no. of block of memory:-\t");
scanf("%d",&n);
printf("\nEnter size of memory:-\n");
for(i=1;i<=n;i++)
{
 scanf("%d",&size[i]);
}
printf("Enter number of memory requested:-\t");
scanf("%d",&nr);
printf("\nEnter request size of memory:-\n");
for(i=1;i<=nr;i++)
{
 scanf("%d",&req[i]);
}
if(nr<=n)
{
do
{
 for(i=1;i<=n;i++)
 {
 fm[i]=0;
 fp[i]=0;
 }

 for(i=1;i<=nr;i++)
 {
 fn[i]=0;
 }
printf("\n\n\n1.>first fit");
printf("\n2.>next fit");
printf("\n3.>best fit");
printf("\n4.>worst fit");
printf("\n5.>exit");
printf("\n\tEnter your choice:-\t");
scanf("%d",&ch);
switch(ch)
  {
  case 1:
  printf("\n\tALLOCATION BY FIRST FIT:-");
  for(j=1;j<=nr;j++)
  {
  for(i=1;i<=n;i++)
  {
   if(size[i]>=req[j] && fm[i]==0 && fn[j]==0)
   {
   fm[i]=1;
   fn[j]=1;
```

```
 intfrg=size[i]-req[j];
 printf("\n\t%d memory size is allocated to %d and internal fragmentation is %d.",req[j],size[i],intfrg);
 }
}
if(fn[j]==0)
{
printf("\n\t%d is not allocated.",req[j]);
}
}
break;
case 2:
printf("\n\tALLOCATION BY NEXT FIT:-");
 t=1;
 for(j=1;j<=nr;j++)
{
for(i=t;i<=n;i++)
{
 if(size[i]>=req[j] && fm[i]==0 && fn[j]==0)
 {
 fm[i]=1;
 fn[j]=1;
 intfrg=size[i]-req[j];
 printf("\n\t%d memory size is allocated to %d and internal fragmentation is %d.",req[j],size[i],intfrg);
 t=i;
 break;
 }
 if(i==n)
 {
 t=1;
 }
}
if(fn[j]==0)
{
printf("\n\t%d is not allocated.",req[j]);
}
}
break;
case 3:
printf("ALLOCATION BY BEST FIT:-");
for(j=1;j<=nr;j++)
{
l=1;
f=0;
for(i=1;i<=n;i++)
{
if(size[i]>=req[j] && fm[i]==0 && fn[j]==0)
 {
 k[i]=size[i]-req[j];
 f=1;
 }
 else
  k[i]=999;
}
if(f==0)
{
```

```
printf("\n\t%d is not allocated.",req[j]);
}
else
{
min=k[1];
for(i=1;i<=n;i++)
{
 if(min>k[i])
 {
   min=k[i];
   l=i;
 }
}
printf("\n\t%d is allocated by %d internal fragmentation is %d",req[j],size[l],min);
fm[l]=1;
}
     }
     break;
     case 4:
printf("ALLOCATION BY WORST FIT:-");
for(j=1;j<=nr;j++)
 {
l=1;
f=0;
for(i=1;i<=n;i++)
{
if(size[i]>=req[j] && fm[i]==0)
 {
 fp[j]=size[i];
 k[i]=size[i]-req[j];
 fm[i]=1;
 }
}
if(f==0)
{
printf("\n\t%d is not allocated.",req[j]);
}
else
{
min=size[1];
for(i=1;i<=n;i++)
{
 if(min<size[i] && fp[w]==0)
 {
   min=size[i];
   l=i;
   w=i;
 }
}
printf("\n\t%d is allocated by %d internal fragmentation is %d",req[j],min,k[l]);
}
     }
break;
     case 5:
exit(0);
```

```
  break;
}
}while(ch!=5);
}
else
 printf("\nYou have requested more holes than available.");
getch();
}


//output
Enter no. of block of memory:- 7
Enter size of memory:-
5
1
0
1
2
3
4
5
1
5

1
Enter number of memory requested:-
Enter request size of memory:-
3
1.>first fit
2.>next fit
3.>best fit
4.>worst fit
5.>exit
 Enter your choice:- 1
 ALLOCATION BY FIRST FIT:-
 3 memory size is allocated to 5 and internal fragmentation is 2.


1.>first fit
2.>next fit
3.>best fit
4.>worst fit
5.>exit
 Enter your choice:- 2
 ALLOCATION BY NEXT FIT:-
 3 memory size is allocated to 5 and internal fragmentation is 2.


1.>first fit
2.>next fit
3.>best fit
4.>worst fit
5.>exit
 Enter your choice:- 3
 ALLOCATION BY BEST FIT:-
```

3 is allocated by 5 internal fragmentation is 2

1.>first fit
2.>next fit
3.>best fit
4.>worst fit
5.>exit
Enter your choice:- 4
ALLOCATION BY WORST FIT:-
3 is not allocated.

SSBT's College of Engineering & Technology, Bambhori, Jalgaon

Computer Engineering Department

Name of student:

Date of Performance: Date of Completion:

EXPERIMENT NO. 4

TITLE: - Page Replacement Algorithm.

AIM: - Implementation of Page Replacement Algorithms - FIFO, LRU and Optimal.

HARDWARE / SOFTWARE REQUIREMENTS: - Turbo C, PC, Mouse.

THEORY: -

Page Replacement -

Initially execution of a process starts with none of its pages in memory. Each of its pages page fault at least once when it is first referenced. But it may so happen that some of its pages are never used. In such case those pages which are not referenced even once will never be brought into memory. This saves load time and memory space. If this is so the degree of multiprogramming increased so that more ready processes can be loaded and executed. Now we may come across a situation where in all of a sudden a process hit to not accessing certain pages start accessing those pages. The degree of multiprogramming has been raised without looking into this aspect and the memory is over allocated. Over allocation of memory shows up when there is a page fault for want of page in memory and the operating system finds the required page in backing store but cannot bring in the page into memory for want of free frames.

There are three page replacement policies to be followed in demand paging -

1] FIFO Page Replacement –

The first in first out policy simply removes pages in the order they arrived in the main memory. Using this policy we simply remove a page based on the time of its arrival in the memory. Clearly, use of this policy would suggest that we swap page located at there position as it arrived in the memory earliest.

The first in first out page replacement algorithm is the simplest page replacement algorithm. When a page replacement is required the oldest page in memory victim. The performance of the FIFO algorithm is not always good. The replaced page may have an initialization module that needs to be executed only once and therefore no longer needed. On the other hand the page may have a heavily used variable in constant use. Such page swapped out will cause a page fault almost immediately to be brought in. Thus the number of page faults increases and results in slower process execution.

2] LRU Page Replacement –

The main distinction between FIFO and optimal algorithm is that the FIFO algorithm uses the time when a page was brought into memory(looks back) where as the optimal algorithm uses the time when a page is to be used in future (looks ahead). If the recent page is used as an approximation of the near future, then replace the page that has not been used for the longest period of time. This is the last recently used (LRU) algorithm.

LRU expands to least recently used. This policy suggests that we remove a page whose last usage is farthest from current time.

3] Optimal Page Replacement –

One result of discovery of algorithm for an optimal page replacement algorithm. It has lower page fault rate of all algorithm. Anomaly of such algorithm is minimum replace the page that will not be used for longest period of time.

Implementation of the optimal page replacement algorithm is difficult since it requires future a priori knowledge of the reference string. Hence the optimal page replacement algorithm is more a benchmark for comparison.

ALGORITHM: -

1) read the size of page frame.
2) read the length of string.
3) read the reference string.
4) read the number of choice as follows
    1. FIFO
    2. LRU
    3. Optimal

if choice =1
then select first page to replace.
If choice =2
then select past or least recently used page for replacement.
If choice =3
then select page which is having longest future reference for replacement.

RESULT: -

REFERENCES: -

1. Operating System Concepts, 6th edition By Abraham Silberschatz, Peter Baer Galvin.
2. Modern Operating System, 2nd edition By Andrew Tanenbaum.

QUESTIONS FOR VIVA: -

1) Why paging is used?

2) What is fragmentation? Explain different types of fragmentation?

3) What are demand- and pre-paging?

4) What is page fault?

5) Explain page replacement policies.

Name & Sign of Teacher

```
// program for page rep,Experiment No.4
#include<stdio.h>
void main()
{
 int i,size,n,a[20],ch;
 char c;
 clrscr();
 printf("\nENTER THE FRAME SIZE");
 scanf("%d",&size);
 printf("\nHOW MANY PAGES DO YOU WANT TO ENTER");
 scanf("%d",&n);
 printf("\nENTER THE SEQUENCE OF PAGES");
 for(i=0;i<n;i++)
 {
  scanf("%d",&a[i]);
 }
 do
 {
  printf("\n1.FIFO ALGORITHM");
  printf("\n2.OPTIMAL ALGORITHM");
  printf("\n3.LRU ALGORITHM");
  printf("\n4.EXIT");
  printf("\nENTER YOUR CHOICE");
  scanf("%d",&ch);
  switch(ch)
  {
   case 1:
    fifo(a,size,n);
    break;
   case 2:
    optimal(a,size,n);
    break;
   case 3:
    lru(a,size,n);
    break;
   case 4:
    exit(0);
  }
 }while(ch!=4);
 getch();
}
int check(int d,int x[10],int size)
{
 int flag=0,i;
 for(i=0;i<size;i++)
 {
  if(x[i]==d)
   flag=1;
 }
 return flag;
}
int print(int x[20],int size)
{
 int i;
 for(i=0;i<size;i++)
```

```
 {
  if(x[i]==-1)
   printf("\t%c",'-');
  else
   printf("\t%d",x[i]);
 }
 return 0;
}
int fifo(int a[20],int size,int n)
{
 int i,x[5],j,k=1,flag,fault=0;
 float s;
 for(i=0;i<size;i++)
  x[i]=-1;
 j=0;
 for(i=0;i<n;i++)
 {
  flag=check(a[i],x,size);
  if(!flag)
  {
   x[j]=a[i];
   fault++;
   printf("\nPAGE FRAME %d:",k++);
   print(x,size);
   j=(j+1)%3 ;
  }
     // if(j==0)
/*  {
   getch();
  }   */
 }
// getch();
 s=(float)fault/(float)n;
 printf("\nTOTAL NO FAULTS ARE:%d",fault);
 printf("\nFAULT RATE IS:%f",s);
  // getch();
 return 0;
}
int find0(int a[20],int n,int x[5],int size,int p)
{
 int k=0,t[5],m=-1,j,i,q;
 j=0;
 do
 {
  for(i=p;i<n;i++)
  {
   if(x[j]==a[i])
    break;
  }
  t[++k]=i;
  j++;
 }while(j<size);
 for(i=0;i<size;i++)
 {
  if(t[i]>n)
```

```
   return i;
  }
  i=0;
  while(i<size)
  {
   if(m<t[i])
   {
    m=t[i];
    q=i;
   }
   i++;
  }
  return q;
}
int find1(int a[20],int p,int x[5],int size)
{
  int j,i,t[5],q,m=999,k=0;
  j=0;
  do
  {
   for(i=p;i>=0;i--)
   {
    if(x[j]==a[i])
     break;
   }
   t[k++]=i;
   j++;
  }while(j<size);
  i=0;
  while(i<size)
  {
   if(m>t[i])
   {
    m=t[i];
    q=i;
   }
   i++;
  }
  return q;
}
int optimal(int a[20],int n,int size)
{
  int i,x[5],j,flag,fault=0,k=1;
  float s;
  for(i=0;i<size;i++)
   x[i]=-1;
  j=-1;
  for(i=0;i<n;i++)
  {
   flag=check(a[i],x,size);
   if(!flag)
   {
    if(fault>=3)
     j=find0(a,n,x,size,i+1);
    else
```

```
    j++;
   x[j]=a[i];
   fault++;
   printf("\nPAGE FRAME:%d",k++);
   print(x,size);
  }}
 printf("\nTOTAL NO OF FAULTS ARE:%d",fault);
 s=(float)fault/(float)n;
 printf("\nFAULT RATE:%f",s);
 return 0;
}
int lru(int a[20],int size,int n)
{
 int j,i,x[5],flag,fault=0,k=1;
 float s;
 for(i=0;i<size;i++)
 {
  x[i]=-1;}
 j=-1;
 for(i=0;i<n;i++)
 {
  flag=check(a[i],x,size);
  if(!flag)
  {
   if(fault>=3)
    j=find1(a,i-1,x,size);
   else
    j++;
   x[j]=a[i];
   fault++;
   printf("\nPAGE FRAME:%d",k++);
   print(x,size);
  }}
 printf("\nTOTAL NO FAULTS ARE:%d",fault);
 s=(float)fault/(float)n;
 printf("\nFAULT RATE:%f",s);
 return 0;
}

//OUTPUT
Enter frame size:       3
Enter length of string: 20
enter the string7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
 *******menu********
1.FIFO
2.OPTIMAL
3.LRU
4.EXIT
 enter ur choice1

  7       1000    1000
  7       0       1000
  7       0       1
  2       0       1
  2       3       1
```

```
 2      3      0
 4      3      0
 4      2      0
 4      2      3
 0      2      3
 0      1      3
 0      1      2
 7      1      2
 7      0      2
 7      0      1
Num of page faults are: 15
 *******menu********
1.FIFO
2.OPTIMAL
3.LRU
4.EXIT
 enter ur choice2

7      0      0
7      0      0
7      0      1
7      2      1
7      0      1
7      3      1
7      0      1
7      4      1
7      2      1
7      3      1
7      0      1
7      3      1
7      2      1
7      0      1
 no of pages fault      9
 *******menu********
1.FIFO
2.OPTIMAL
3.LRU
4.EXIT
 enter ur choice3

7      1000   1000
7      0      1000
7      0      1
2      0      1
2      0      3
4      0      3
4      0      2
4      3      2
0      3      2
1      3      2
1      0      2
1      0      7
Num of page faults are: 12
 *******menu********
1.FIFO
```

2.OPTIMAL
3.LRU
4.EXIT
 enter ur choice4

SSBT's College of Engineering & Technology, Bambhori, Jalgaon

Computer Engineering Department

Name of student:

Date of Performance: Date of Completion:

EXPERIMENT NO. 5

TITLE: - Banker's Deadlock Avoidance Algorithm.

AIM: - Implementation of Banker's deadlock avoidance algorithm.

HARDWARE / SOFTWARE REQUIREMENTS: - Turbo C, PC, Mouse.

THEORY: -

The Banker's algorithm is a resource allocation & deadlock avoidance algorithm developed by Edsger Dijkstra that tests for safety by simulating the allocation of pre-determined maximum possible amounts of all resources, and then makes a "safe-state" check to test for possible deadlock conditions for all other pending activities, before deciding whether allocation should be allowed to continue. The algorithm was developed in the design process for the operating system and originally described (in Dutch) in EWD108. The name is by analogy with the way that bankers account for liquidity constraints.

The Banker's algorithm is run by the operating system whenever a process requests resources. The algorithm avoids deadlock by denying or postponing the request if it determines that accepting the request could put the system in an unsafe state (one where deadlock could occur). When a new process enters a system, it must declare the maximum number of instances of each resource type that may not exceed the total number of resources in the system. Also, when a process gets all its requested resources it must return them in a finite amount of time.

Assuming that the system distinguishes between four types of resources, (A, B, C and D), the following is an example of how those resources could be distributed. Note that this example shows the system at an instant before a new request for resources arrives. Also, the types and number of resources are abstracted. Real systems, for example, would deal with much larger quantities of each resource.

Safe and Unsafe States -

A state (as in the above example) is considered safe if it is possible for all processes to finish executing (terminate). Since the system cannot know when a process will terminate, or how many resources it will have requested by then, the system assumes that all processes will

eventually attempt to acquire their stated maximum resources and terminate soon afterward. This is a reasonable assumption in most cases since the system is not particularly concerned with how long each process runs (at least not from a deadlock avoidance perspective). Also, if a process terminates without acquiring its maximum resources, it only makes it easier on the system. Given that assumption, the algorithm determines if a state is safe by trying to find a hypothetical set of requests by the processes that would allow each to acquire its maximum resources and then terminate (returning its resources to the system). Any state where no such set exists is an unsafe state.

ALGORITHM: -

1] Safety Algorithm

The algorithm for finding out whether or not a system is in a safe state can be described as follows -

1) Let Work and Finish be vectors of length m & n respectively. Initialize
   Work = Available and Finish[i] = false for i = 0, 1, 2,…..,n-1.
2) Find an index i such that both
   a. Finish[i] == false
   b. Need i <= Work

   If no such i exists, go to step 4.

3) Work = Work + allocation i
   Finish[i] = true
   Go to step 2
4) if Finish[i] == true for all i, then the system is in safe state.

2] Resource Request Algorithm

Let request i be the request vector for processes Pi. if request i [j] = k, then process Pi wants k instances of resource type Rj. When a request for resources is made by process Pi, the following actions are taken -

1) if request i <= Need I, go to step 2. Otherwise raise an error condition, since the process has executed its maximum claim.
2) If request i <= Available, go to step 3. Otherwise, Pi must wait, since the resources are not available.
3) Have the system pretend to have allocate the requested resources to process Pi modifying the state as follows:

   Available = Available – Request i

   Allocation i = Allocation i + Request i

   Need i := Need i – Request i

If the resulting resource allocation state is safe, the transaction is completed and process Pi is allocated its resources.

RESULT: -

REFERENCES: -

1. Operating System Concepts, 6th edition By Abraham Silberschatz, Peter Baer Galvin.

QUESTIONS FOR VIVA: -

1) What is the meaning of deadlock?

2) When is a system in safe state?

3) What is deadlock avoidance?

4) How hold and wait can be prevented in a deadlock prevention approach.

5) Explain Bankers algorithm.

Name & Sign of Teacher

```
//#include<stdio.h>
//#include<conio.h>
void main()
{
 int a[10][10],m[10][10],ne[10][10],av[10],pri[10],flag,j,k,n,i,c[10],count=0,fla,f,fl[10];
 clrscr();
 for(i=1;i<=10;i++)
 c[i]=0;
 printf("\nENTER THE NO OF PROCESSES :\t");
 scanf("%d",&n);
 for(i=1;i<=n;i++)
 {
  printf("\nENTER THE MAX NEED FOR PROCESS P%d :\t",i);
  for(j=1;j<=3;j++)
  {
   scanf("%d",&m[i][j]);
  }
 }
 for(i=1;i<=n;i++)
 {
  printf("\nENTER THE RESOURSES ALLOCATED TO PROCESS P%d :\t",i);
  for(j=1;j<=3;j++)
  {
   scanf("%d",&a[i][j]);
  }
 }
  for(i=1;i<=n;i++)
 {
  for(j=1;j<=3;j++)
  {
   ne[i][j]=m[i][j]-a[i][j];
  }
 }
 printf("\nENTER THE NO OF AVAILABLE RESOURSES :\t");
 for(i=1;i<=3;i++)
 {
  scanf("%d",&av[i]);
 }
 printf("\n\t\tTABLE\n");
 printf("\nPROCESSES\t\tMAX NEED\t\tALLOCATED\t\tNEED\n");
 printf("\n\t\t\tA  B  C \t\tA  B  C\t\t\tA  B  C\n");
 for(i=1;i<=n;i++)
 {
  printf("\nP%d",i);
  printf("\t\t\t%d  %d  %d \t\t%d",m[i][1],m[i][2],m[i][3],a[i][1],a[i][2]);
  printf("  %d  %d\t\t\t%d  %d  %d",a[i][2],a[i][3],ne[i][1],ne[i][2],ne[i][3]);
 }
 printf("\nAVAILABLE RESOURSES\nA\tB\tC\n");
 for(i=1;i<=3;i++)
 {
  printf("%d\t",av[i]);
 }
 label:
 for(i=1;i<=n;i++)
 {
```

```
 for(j=1;j<=3;j++)
 {
  if((av[j]>=ne[i][j])&&c[i]==0)
  flag=1;
  else
  {
   flag=0;
   break;
  }
 }
 if(flag==1)
 {
  for(k=1;k<=3;k++)
  {
   av[k]=av[k]+a[i][k];
  }
  c[i]=1;
  count++;
  pri[count]=i;
 }
}
for(i=1;i<=n;i++)
{
 for(j=1;j<=3;j++)
 {
  if((av[j]>=ne[i][j])&&c[i]==0)
  fla=1;
  else
  {
  fla=0;
  break;
  }
 }
 if(fla==1)
 fl[i]=1;
}
for(i=1;i<=n;i++)
{
 if(fl[i]==0)
 f=0;
 else
 {
 f=1;
 break;
 }
}
if(f==1)
goto labe;
else
{
printf("\nTHE DEADLOCK IS PRESENT IN SYSTEM");
goto lab;
}
labe:
for(i=1;i<=n;i++)
```

```
 {
  if(c[i]==1)
  flag=1;
  else
  {
  flag=0;
  break;
  }
 }
 if(flag==0)
 goto label;
 printf("\nTHE SYSTEM IS IN SAFE MODE \n THE SEQUENCE IS:");
 for(i=1;i<=n;i++)
 printf("\t ==> P%d",pri[i]);
 lab:
 getch();
}
  //OUTPUT//
ENTER THE NO OF PROCESSES :   4
ENTER THE MAX NEED FOR PROCESS P1 :     8
7
9

ENTER THE MAX NEED FOR PROCESS P2 :     6
5
7

ENTER THE MAX NEED FOR PROCESS P3 :     6
8
7

ENTER THE MAX NEED FOR PROCESS P4 :     4
6
8

ENTER THE RESOURSES ALLOCATED TO PROCESS P1 :   8
7
9

ENTER THE RESOURSES ALLOCATED TO PROCESS P2 :   5
4
6

ENTER THE RESOURSES ALLOCATED TO PROCESS P3 :   5
3
6

ENTER THE RESOURSES ALLOCATED TO PROCESS P4 :   2
4
7

ENTER THE NO OF AVAILABLE RESOURSES :   11 7 8

               TABLE
```

PROCESSES	MAX NEED	ALLOCATED	NEED
	A B C	A B C	A B C
P1	8 7 9	8 7 9	0 0 0
P2	6 5 7	5 4 6	1 1 1
P3	6 8 7	5 3 6	1 5 1
P4	4 6 8	2 4 7	2 2 1

AVAILABLE RESOURSES

A	B	C
11	7	8

THE SYSTEM IS IN SAFE MODE

THE SEQUENCE IS: ==> P1 ==> P2 ==> P3 ==> P4