Guru Nanak Institute of Technology

Department of Computer Science &Engineering

OPERATING SYSTEMS

Lab Manual [Subject Code:]
For the Academic year 2015-16

III B Tech Semester I– [CSE]

Guru Nanak Institute of Technology
Ibrahimpatnam, R R District – 501 506 (Telangana State)

Department of Computer Science &Engineering

LAB MANUAL FOR THE ACADEMIC YEAR 2015-16

SUB	**: OPERATING SYSTEMS LAB**
SUB CODE	**:**
SEMESTER	**:** I
STREAM	**:** CSE
INSTRUCTORES	**: Mr. Devi Prasad Mishra**
PROGRAMMER	**:**
VENUE	**:**
BLOCK	**: CSE**

INDEX

Result: _________________ Signature of the faculty incharge

Contents of Lab Manual

Contents of Lab manual(student):

1. Index
2. List of experiments as per the university.
3. List of experiments to be conducted for this semester. Add the additional experiments list separately.
4. Cycle to indicate schedule and the batch size.
5. Content of the experiment. (for CSE/Computer labs only flow charts/ algorithm should be given with input and output).

Contents of Lab manual(Teacher):

1. Index
2. List of experiments as per the university.
3. List of experiments to be conducted for this semester. Add the additional experiments list separately.
4. Cycle to indicate schedule and the batch size.
5. Content of the experiment. (for CSE/Computer labs flow charts/ algorithm should be given with input and output. Program should also include. For other than computer labs master reading to be taken and noted).

INDEX

Operating Systems LAB

1.LAB OBJECTIVE

- To Understand the Operating System functionalities

Operating Systems LAB

2.LAB OUTCOME

Upon successful completion of this Lab the student will be able to:

- To write software systems based on multiple cooperating processes or threads
- To implement process scheduling & synchronization algorithms
- To implement algorithms considering time and space complexity

3. INTRODUCTION ABOUT LAB

There are 60 systems (Compaq Presario) installed in this Lab. Their configurations are as follows:

Processor	:	Intel(R) Pentium(R) Dual CPU 2.0GH$_z$
RAM	:	2 GB
Hard Disk	:	160 GB
Mouse	:	Optical Mouse

Software

- All systems are configured in **DUAL BOOT** mode i.e., Students can boot from Windows XP or Linux as per their lab requirement. This is very useful for students because they are familiar with different Operating Systems so that they can execute their programs in different programming environments.
- Each student has a separate login for database access **Oracle 9i client** version is installed in all systems. On the server, account for each student has been created. This is very useful because students can save their work (scenarios', PL / SQL programs, data related projects, etc) in their own accounts. Each student work is safe and secure from other students.

- ETL TOOL: Informatica 7.1.1 is installed in all the systems.

  Students can execute ETL process using this tool. They can import the Meta data from different source definition into the data warehouse by applying the business logics.

- Reporting TOOL: COGONS is installed in all the systems.

  Students can generate reports using this tool from different Business Objects in different formats.

- **MASM (Macro Assembler)** is installed in all the systems

- Students can execute their assembly language programs using MASM. MASM is very useful students because when they execute their programs they can see contents of **Processor Registers** and how **each instruction** is being executed in the **CPU**.
- Rational Rose Software is installed in some systems
- Using this software, students can depict UML diagrams of their projects.
- **Softwares installed:** C, C++, JDK1.5, MASM, OFFICE-XP, J2EE and DOT NET, Rational Rose.
- Systems are provided for students in the 1:1 ratio.
- Systems are assigned numbers and same system is allotted for students when they do the lab.

A. STANDARD OPERATING PROCEDURE – SOP

a) Explanation on today's experiment by the concerned faculty using OHP/PPT covering the following aspects: 25 mins.

1) Name of the experiment/Aim
2) Software/Hardware required
3) Commands with suitable Options
4) Test Data
    1) Valid data sets
    2) Limiting value sets
    3) Invalid data sets

b) Writing of shell programs by the students 25 min.

c) Compiling and execution of the program

<u>Writing of the experiment in the Observation Book</u>:

The students will write the today's experiment in the Observation book as per the following format:

a) Name of the experiment/Aim
b) Software/Hardware required
c) Commands with suitable Options
d) Shell Programs/System call using C-Programs
e) Test Data
    a. Valid data sets
    b. Limiting value sets
    c. Invalid data sets
f) Results for different data sets
g) Viva-Voce Questions and Answers
h) Errors observed (if any) during compilation/execution
i) Signature of the Faculty

B. Guide Lines to Students in Lab

Disciplinary to be maintained by the students in the Lab:

- Students are required to carry their lab observation book and record book with completed experiments while entering the lab.
- Students must use the equipment with care. Any damage is caused student is punishable
- Students are not allowed to use their cell phones/pen drives/ CDs in labs.
- Students need to be maintain proper dress code along with ID Card
- Students are supposed to occupy the computers allotted to them and are not supposed to talk or make noise in the lab.
- Students, after completion of each experiment they need to be updated in observation notes and same to be updated in the record.
- Lab records need to be submitted after completion of experiment and get it corrected with the concerned lab faculty.
- If a student is absent for any lab, they need to be completed the same experiment in the free time before attending next lab.

Steps to perform experiments in the lab by the student

Step1: Students have to write the date, aim, S/W & H/W requirements for that experiment in the observation book.
Step2: Students have to listen and understand the experiment explained by the faculty and note down the important points in the observation book.
Step3: Students need to write procedure/algorithm in the observation book.
Step4: Analyze and Develop/implement the logic of the program by the student in respective platform
Step5: after approval of logic of the experiment by the faculty then the experiment has to be executed on the system.
Step6: After successful execution the results are to be shown to the faculty and noted the same in the observation book.
Step7: Students need to attend the Viva-Voce on that experiment and write the same in the observation book.
Step8: Update the completed experiment in the record and submit to the concerned faculty in-charge.

Instructions to maintain the record

- Before start of the first lab they have to buy the record and bring the record to the lab.
- Regularly (Weekly) update the record after completion of the experiment and get it corrected with concerned lab in-charge for continuous evaluation.

- In case the record is lost inform the same day to the faculty in charge and get the new record within 2 days the record has to be submitted and get it corrected by the faculty.
- If record is not submitted in time or record is not written properly, the evaluation marks (5M) will be deducted.

<u>Awarding the marks for day to day evaluation:</u>

Total marks for day to day evaluation is 15 Marks as per JNTUH.
These 15 Marks are distributed as:

Record	5 Marks
Exp setup/program written and execution	5 Marks
Result and Viva-Voce	5 Marks

<u>Allocation of Marks for Lab Internal</u>

Total marks for lab internal is 25 Marks as per JNTUH.
These 25 Marks are distributed as:
Average of day to day evaluation marks : 15 Marks
Lab Mid exam: 10 Marks

<u>Allocation of Marks for Lab External</u>

Total marks for lab internal is 50 Marks as per JNTUH.
These 50 Marks are distributed as:
Program Written: 20 Marks
Program Execution and Result: 15 Marks
Viva-Voce: 10 Marks
Record: 5 Marks

Guidelines to Students

- Equipment in the lab for the use of student community. Students need to maintain a proper decorum in the computer lab. Students must use the equipment with care. Any damage is caused is punishable.
- Students are required to carry their observation / programs book with completed exercises while entering the lab.
- Students are supposed to occupy the machines allotted to them and are not supposed to talk or make noise in the lab. The allocation is put up on the lab notice board.
- Lab can be used in free time / lunch hours by the students who need to use the systems should take prior permission from the lab in-charge.
- Lab records need to be submitted on or before date of submission.
- Students are not supposed to use floppy disks
- Use of computer network is encouraged.

List of Lab Experiments as per JNTU Syllabus

1) Simulate the following CPU Schedulling 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 Organisation Techniques

a)Single level directory b)Two level directory c)Hierarchial Directory d)DAG

5)Simulate Bankers Algorithm for Deadlock Avoidance

6)Simulate Bankers Algorithm for DeadLock Prevention

7)Simulate all page replacement algorithms

a) FIFO b) LRU c)LFU

8)Simulate paging Technique of Memory Management

List of Additional Experiments for the Semester

1) Write a program to implement Consumer-Producer Problem

Refernce Books

1)Operating Systems –Internals and Design Principles, Stallings,Sixth Edition-2009 ,Pearson education

2)Operting Systems., S.Haldar A.A.Aravind.,Pearson Education

8.Contents of Lab Experiments

Sub Task-1

a) **AIM: To Simulate the Round Robin CPU scheduling algorithm**

. **Recommended Hardware / Software Requirements:**

- Hardware Requirements: Intel Based desktop PC LANS Connected with minimum of 166 MHZ or faster processor with at least 64 MB RAM and 100 MB free disk space.
- TC

Prerequisite

Theory:

Round Robin: It is a primitive scheduling algorithm it is designed especially for time sharing systems. In this, the CPU switches between the processes. When the time quantum expired, the CPU switches to another job. A small unit of time called a quantum or time slice. A time quantum is generally is a circular queue new processes are added to the tail of the ready queue.
If the process may have a CPU burst of less than one time slice then the
process release the CPU voluntarily. The scheduler will then process to next process ready queue otherwise; the process will be put at the tail of the ready queue.

ALGO RITHM

```
Step1:      Start
Step2:      Define a structure process with elements pid,at,bt,st,ft,wt,tat.Create an
            Array of  Structure say p[100] ,Temp,rbt& fturn
Step3:      Read I,j,n
Step4:      Read the Processes details pid,at & bt
Step5:      for i=0 to i<n do till step7
Step6:      for j=0 to j<n-1 do till step 6
Step7:      check p[j]-at>p[j+1]
            If true then
            Temp=p[j]
            P[j]=p[j+1]
            P[j+1]=temp
            I← 0,k← n,time=p[0].at
Step8:      while n>0 then do
            If i=k then do
            If  p[i].rbt!=0 then
            If p[i].rbt<tim_sli then
            P[i].st← time
            Time← time+p[i].rbt
            P[i].rbt← 0
            P[i].ft← time
            N=n-1
```

```
Step9:      else if p[i].fturn=0 then
            P[i].fturn←1
            P[i].st← time
            p[i].rbt← p[i].rbt-time_sli
            time← time+time_sli
Step10:     if p[i].rbt=0 then
            P[i].ft← time
            N=n-1
Step11:     if time<p[i+1].at then
            For j=0 to I do
            If p[j].rbt!=0 then
            If p[j].rbt <time_sli then
            Time=time+p[j].rbt
            P[j].rbt←0
            P[j].ft← time
            N=n-1
Step12:     else
            P[i].rbt← p[i].rbt-time_sli
            If p[i].rbt=0 then
            P[j].ft← time
            N=n-1
Step13:     Print the output with the RoundRobin Fashion and Calculating
            St,ft,wt,&tat
Step14:     End
```

PROGRAM

```
#include<stdio.h>
#include<conio.h>
struct process
{
int id,at,bt,st,ft,wt,tat,fturn,rbt;
}p[10],temp;
void main()
{
int i,j,k,n,time=0,count=1,c=1,time_sli;
clrscr();
printf("Nter the process");
scanf("%d",&n);
printf("Nter the process id,at,bt \n");
for(i=0;i<n;i++)
{
scanf("%d %d %d",&p[i].id,&p[i].at,&p[i].bt);
p[i].fturn=0;
p[i].rbt=p[i].bt;
}
printf("Nter the time slice");
scanf("%d",&time_sli);
for(i=0;i<n;i++)
for(j=i;j<n-1;j++)
if(p[j].at>p[j+1].at)
```

```
{
temp=p[j];
p[j]=p[j+1];
p[j+1]=temp;
}
printf("\n Process id \t time\n");
i=0;
k=n;
time=p[0].at;
while(n>0){
if(i==k)
i=0;
if(p[i].rbt!=0)
{
if(p[i].rbt<time_sli)
{
if(p[i].fturn==0)
p[i].st=time;
time=time+p[i].rbt;
p[i].rbt=0;
p[i].ft=time;
n--;
printf("%d \t \t %d\n",p[i].id,time);
}
else
{
if(p[i].fturn==0)
{
p[i].fturn=1;
p[i].st=time;
}
p[i].rbt=p[i].rbt-time_sli;
time=time+time_sli;
if(p[i].rbt==0)
{
p[i].ft=time;
n--;
}
printf("%d \t \t %d\n ",p[i].id,time);
if(time<p[i+1].at)
{
for(j=0;j<=i;j++)
{
if(p[i].rbt!=0)
{
if(p[j].rbt<time_sli)
{
time=time+p[j].rbt;
p[j].rbt=0;
p[j].ft=time;
n--;
printf("%d \t \t%d\n",p[j].id,time);
}
else
{
p[i].rbt=p[i].rbt-time_sli;
if(p[j].rbt==0)
{
p[j].ft=time;
n--;
}
printf("%d \t\t%d\n",p[j].id,time);
}}}
}}}
i++;
}
printf("\n id\t at \tbt \tst\tft\twt\ttat\n");
printf("...........................\n");
for(i=0;i<k;i++)
{
p[i].tat=p[i].ft-p[i].at;
p[i].wt=p[i].ft-p[i].bt-p[i].at;
printf("%d
\t%d\t%d\t%d\t%d\t%d\t%d\n",p[i].id,p[i].at,p[i].bt,p[i].st,p[i].ft,p[i].wt,p[i].tat);}
getch();
}
```

OUTPUT:

```
Nter the process3
Nter the process id,at,bt
1 0 3
2 1 4
3 2 3
Nter the time slice2

 Process id      time
1               2
 2              4
 3              6
 1              7
2               9
 3              10

 id     at     bt     st     ft     wt     tat
...........................
1       0      3      0      7      4      7
2       1      4      2      9      4      8
3       2      3      4      10     5      8
```

SHORTEST JOB FIRSTSCHEDULINGALGORITHM

(b) AIM: To Simulate the Shortest Job First Schedulling Algorithm

. **Recommended Hardware / Software Requirements:**

- Hardware Requirements: Intel Based desktop PC LANS Connected with minimum of 166 MHZ or faster processor with at least 64 MB RAM and 100 MB free disk space.
- TC

Prerequisite

Theory:

Shortest job First: The criteria of this algorithm are which process having the smallest CPU burst, CPU is assigned to that next process. If two process having the same CPU burst time FCFS is used to break the tie.

ALGORITHM

```
Step1:      Start
Step2:      Define a structure process with elements pid,at,bt,st,ft,wt,tat.Create an
            Array of  Structure say p[100] & Temp
Step3:      Read I,j,k,n,b,c,r,q,count,min,pos
Step4:      Read id,at,and bt of the process
Step5:      for i=0 to n do
                For j=0 to n-1 do
                if (p[j.at>p[j+1].at) then
                Temp=p[j]
                P[j]=p[j+1]
                P[j+1]=temp
Step6:      Initialize min=p[0].bt
Step7:      for i=1 till p[i].at=p[0].at do
            if(p[i].bt<min) then
            min=p[i].bt
            pos=i
Step8:      temp=p[0]
Step9:      P[0]=p[pos]
Step10      P[pos]=temp
Step11:     i=0
Step12:     p[i].st←p[i].at
Step13:     p[i].ft←p[i].st+p[i].bt
Step14:     for i=1 to n do till Step18
            Count=0
Step15:     for r=I to n do
            If(p[r].at<=p[i-1].ft) then
            Count=count+1
Step16:     if count=1 then
```

```
                If p[i].at>p[i-1].ft then
                P[i].st←p[i].at
                Else
                P[i].st← p[i-1].ft
                P[i].ft← p[i].st+p[i].bt
Step17:         else
                For b=I to i+count do
                For c=I to i+count-1 do
                If p[c].bt>p[c+1].bt then
                Temp=p[c]
                P[c]=p[c+1]
                P[c+1]=temp
Step18:         if p[i].at>p[i-1].ft then
                P[i].st=p[i].at
                Else
                P[i].st← p[i-1].ft
                P[i].ft← p[i].st+p[1].bt
Step19:         Display the Output
Step20:         for i=0 to n do
                P[i].wt← p[i].st-p[i].at
                P[i].tat← p[i].ft-p[i].at
Step21:         Print all the details id,at,bt,st,ft,wt,tat
Step22:         End
```

PROGRAM

```
#include<stdio.h>
#include<conio.h>
struct process
{
int id,at,st,bt,ft,tat,wt;
}
p[10],temp;
void main()
{
int i,j,k,n,b,c,r,q,count=0,min=0,pos=0;
clrscr();
printf("Enter the No.of Processes");
scanf("%d",&n);
printf("Nter the id ,at & bt");
for(i=0;i<n;i++)
{
scanf("%d %d %d",&p[i].id,&p[i].at,&p[i].bt);
}
for(i=0;i<n;i++)
{
for(j=0;j<n-1;j++)
{
if(p[i].at>p[j+1].at)
{
```

```
temp=p[j];
p[j]=p[j+1];
p[j+1]=temp;
}
}
}
min=p[0].bt;
for(i=1;p[i].at==p[0].at;i++)
{
if(p[i].bt<min)
{
min=p[i].bt;
pos=i;
}
}
temp=p[0];
p[0]=p[pos];
p[pos]=temp;
i=0;
p[i].st=p[i].at;
p[i].ft=p[i].st+p[i].bt;
for(i=1;i<n;i++)
{
count=0;
for(r=i;r<n;r++)
{
if(p[r].at<=p[i-1].ft)
count=count+1;
}
if(count==1)
{
if(p[i].at>p[i-1].ft)
p[i].st=p[i].at;
else
p[i].st=p[i-1].ft;
p[i].ft=p[i].st+p[i].bt;
}
else
{
for(b=i;b<i+count;b++)
{
for(c=i;c<i+count-1;c++)
{
if(p[c].bt>p[c+1].bt)
{
temp=p[c];
p[c]=p[c+1];
p[c+1]=temp;
}}}
if(p[i].at>p[i-1].ft)
p[i].st=p[i].at;
```

```
else
p[i].st=p[i-1].ft;
p[i].ft=p[i].st+p[i].bt;
}}
printf("id \t at\t bt\t st\tft\twt\ttat\n");
printf("\n.......................................\n");
for(i=0;i<n;i++)
{
p[i].wt=p[i].st-p[i].at;
p[i].tat=p[i].ft-p[i].at;
printf("\n%d\t%d\t%d\t%d\t%d\t%d\t%d",p[i].id,p[i].at,p[i].bt,p[i].st,p[i].ft,p[i].wt,p[i].tat);
}
getch();
}
```

OUTPUT:

```
Enter the No.of Processes2
Nter the id ,at & bt1 0 3
2 1 4
id      at      bt      st     ft      wt      tat

.......................................

1       0       3       0       3       0       3
2        1       4        3       7        2       6
```

FIRST COME FIRST SERVE SCHEDULING ALGORITHM

c)AIM: To simulate the First Come First Serve Schedulling Algorithm

. **Recommended Hardware / Software Requirements:**

- Hardware Requirements: Intel Based desktop PC LANS Connected with minimum of 166 MHZ or faster processor with at least 64 MB RAM and 100 MB free disk space.
- TC

Prerequisite
Theory:

First-come, first-serve scheduling(FCFS): In this, which process enter the ready queue first is served first. The OS maintains DS that is ready queue. It is the simplest CPU scheduling algorithm. If a process request the CPU then it is loaded into the ready queue, which process is the head of the ready queue, connect the CPU to that process.

ALGORITHM

```
Step1:      Start
Step2:      Define a structure process with elements pid,at,bt,st,ft,wt,tat.Create an
            Array of  Structure say p[100] & Temp
Step3:      Read I,j,n
Step4:      Read the Processes details pid,at & bt
Step5:      for i=0 to i<n do till step7
Step6:      for j=0 to j<n-1 do till step 6
Step7:      check p[j]-at>p[j+1]
            If true then
            Temp=p[j]
            P[j]=p[j+1]
            P[j+1]=temp
            I=i+1,  j=j+1
Step8:      for i=0 to n do
Step9:      if i=0 is true then
            P[i].st=p[i].at endif
Step10:     else if p[i].at>p[i-1].ft then
            P[i].st=p[i].at
            P[i].ft=p[i].st+p[i].bt
            End if
Step11:     else
            P[i].st=p[i-1].st+p[i-1].bt
Step12:     end for
Step13:     for i=0 to n do
            P[i].ft=p[i].bt+p[i].st
            P[i].wt=p[i].st-p[i].at
            P[i].tat=p[i].ft-p[i].at
Step14:     Display Output
Step15:     End
```

PROGRAM

```
#include<stdio.h>
struct process
{
int id,at,st,wt,dt,bt,tot;
} p[20],t;
void main()
{
int n,i,j,awt=0,pd=0;
clrscr();
printf("Enter the process");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("Enter id,at,bt");
scanf("%d %d %d ",&p[i].id,&p[i].at,&p[i].bt);
}
for(i=0;i<n;i++)
{
for(j=0;j<n-1;j++)
{
if(p[i].at<p[i].id)
{
t=p[i];
p[i]=p[j];
p[j]=t;
}
}
}
for(i=0;i<n;i++)
{
if(i==0)
{
p[i].st=p[i].at;
p[i].dt=p[i].st+p[i].bt;
}
else if(p[i].at>p[i-1].dt)
{
p[i].st=p[i].at;
p[i].dt=p[i].st+p[i].bt;
}
else
p[i].st=p[i-1].st+p[i-1].bt;
}
for(i=0;i<n;i++)
{
p[i].dt=p[i].st+p[i].bt;
p[i].wt=p[i].st-p[i].at;
p[i].tot=p[i].dt-p[i].at;
awt=awt+p[i].wt;
pd=pd+p[i].tot;
}
printf("id\tat\tbt\tst\tdt\twt\ttot");
for(i=0;i<n;i++)
printf("\n %d \t %d \t %d \t %d \t %d \t %d \t %d
\t%d",p[i].id,p[i].at,p[i].bt,p[i].st,p[i].dt,p[i].wt,p[i].tot);
printf("Avg waiting time is %d",(awt/n));
printf("Avg executing time is %d ",(pd/n));
getch();
}
```

OUTPUT:

```
Enter the No of Process:        5
Enter process id arrival time & burst time:
1 0 3
2 2 6
3 4 4
4 6 5
5 8 2

Pid     at      bt      st      ft      wt      tat
1       0       3       0       3       0       3
2       2       6       3       9       1       7
3       4       4       9       13      5       9
4       6       5       13      18      7       12
5       8       2       18      20      10      12
```

PRIORITY SCHEDULING

d) AIM: To simulate Priority Schedulling Algorithm

. **Recommended Hardware / Software Requirements:**

- Hardware Requirements: Intel Based desktop PC LANS Connected with minimum of 166 MHZ or faster processor with at least 64 MB RAM and 100 MB free disk space.
- TC

Prerequisite

Theory:

Priority Scheduling: The cpu is allocated to the process with the highest priority. Equal priority processes are scheduled in the FCFS order. Priorities aregenerally some fixed range of numbers such as 0 to 409. The low numbers represent high priority

ALGORITHM

```
Step1:      Start

Step2:      Define a structure process with elements pid,at,bt,st,ft,wt,tat.Create an

            Array of  Structure say p[100] & Temp

Step3:      Read I,j,n

Step4:      Read the Processes details pid,at & bt
Step5:      for i=0 to i<n do till step7

Step6:      for j=0 to j<n-1 do till step 6

Step7:      check p[j]-at>p[j+1]

            If true then

            Temp=p[j]

            P[j]=p[j+1]

            P[j+1]=temp

Step8:      for i=1 till p[i].at do

            If p[i].pri<min then

            Min=p[i].bt
```

```
              Pos=i
Step9:        temp=0
Step10:       p0]← p[pos]
              P[]pos]← temp
              I← 0
Step11:       p[i].st← p[i].at,p[i].ft← p[i].st+p[i].bt
Step12:       for i=1 to n do
              Count← 0
              For r=I to n do
              If p[r].at<=p[i-1].ft then
              Count← count+1
Step13:       if count =1 then
              If p[i].at>p[i-1].ft then
              P[i].st← p[i].at
              Else
              P[i].st← p[i-1].ft
              P[i].ft← p[i].st+p[i].bt
Step14:       else do
              For b=I to i+count do
              For c=I to i+count-1 do
              If p[c].pri>p[c+1].pri then
              Temp← p[c]
              P[c]← p[c+1]
              P[c+1]← temp
              If p[i].at>p[i-1].ft
              P[i].st← p[i].at
```

```
            Else do
            P[i].st← p[i-1].ft
            P[i].ft← p[i].st+p[i].bt
Step15:     Print id,at,bt,pri,st,ft after calculating twt,tat accordingly

Step16:     End
```

PROGRAM

```
#include<stdio.h>
#include<conio.h>
struct process
{
int id,at,st,bt,ft,tat,wt,pri;
}p[10],temp;
void main()
{
int i,j,k,n,b,c,r,q,count=0,min=0,pos=0;
clrscr();
printf("Nter the no.of processes");
scanf("%d",&n);
printf("Nter the id,at,bt and Priority");
for(i=0;i<n;i++)
{
scanf("%d %d %d %d",&p[i].id,&p[i].at,&p[i].bt,&p[i].pri);
}
for(i=0;i<n;i++)
for(j=0;j<n-1;j++)
if(p[j].at>p[j+1].at)
{temp=p[j];
p[j]=p[j+1];
p[j+1]=temp;
}
min=p[0].pri;
for(i=1;p[i].at==p[0].at;i++)
if(p[i].pri<min)
{
min=p[i].bt;
pos=i;
}
temp=p[0];
p[0]=p[pos];
p[pos]=temp;
i=0;
p[i].st=p[i].at;
p[i].ft=p[i].st+p[i].bt;
for(i=1;i<n;i++)
{
count=0;
for(r=i;r<n;r++)
{
if(p[r].at<=p[i-1].ft)
{
```

```
count=count+1;
}}
if(count==1)
{
if(p[i].at>p[i-1].ft)
p[i].st=p[i].at;
else
p[i].st=p[i-1].ft;
p[i].ft=p[i].st+p[i].bt;
}
else{
/*for(r=i;r<n;r++)
{
if(p[r].at<=p[i-1].ft)
{
count=count++;
}
}
if(count==1)
{
if(p[i].at>p[i-1].ft)
p[i].st=p[i].at;
else
p[i].st=p[i-1].ft;
p[i].ft=p[i].st+p[i].bt;
}
```

```
else

{ */

for(b=i;b<i+count;b++)

for(c=i;c<i+count-1;c++)

if(p[c].pri>p[c+1].pri)

{

temp=p[c];

p[c]=p[c+1];

p[c+1]=temp;

}

if(p[i].at>p[i-1].ft)

p[i].st=p[i].at;

else

p[i].st=p[i-1].ft;

p[i].ft=p[i].st+p[i].bt;

}}

printf("id \t at\t bt\tpri\tst\tft\twt\ttat\n");

printf("\n--------------------------------\n");

for(i=0;i<n;i++)

{

p[i].wt=p[i].st-p[i].at;

p[i].tat=p[i].ft-p[i].at;

printf("%d\t%d\t %d\t %d\t
%d\t%d\t%d\t%d\n",p[i].id,p[i].at,p[i].bt,p[i].pri,p[i].st,p[i].ft,p[i].wt,p[i].tat);

}

getch();
```

```
}
```

OUTPUT:

```
Nter the no.of processes5

Nter the id,at,bt and Priority1 0 3 1

2 1 4 3

3 2 3 2

4 3 1 4

5 2 2 2

id    at    bt   pri   st    ft    wt    tat

-------------------------------

1     0     3     1     0    3     0     3

3     2     3     2     3    6     1     4

5     2     2     2     6    8     4     6

2     1     4     3     8    12    7     11

4     3     1     4     12   13    9     10
```

Sub Task-2

a)AIM: **To Simulate Sequential file allocation strategies**

. **Recommended Hardware / Software Requirements:**

- Hardware Requirements: Intel Based desktop PC LANS Connected with minimum of 166 MHZ or faster processor with at least 64 MB RAM and 100 MB free disk space.
- TC

Prerequisite

Theory:

a) Sequential allocation :
In this allocation strategy, each file occupies a set of contiguously blocks on the disk. This strategy is best suited. For sequential files. The file allocation table consists of a single entry for each file. It shows the filenames, staring block of the file and size of the file. The main problem of this strategy is, it is difficult to find the contiguous free blocks in the disk and some free blocks could happen between two files.

ALGORITHM:

```
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.
```

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
main()
{
 int n,i,j,b[20],sb[20],t[20],x,c[20][20];
 clrscr();
 printf("Enter no.of files:");
 scanf("%d",&n);
 for(i=0;i<n;i++)
 {
```

```
            printf("Enter no. of blocks occupied by file%d",i+1);
            scanf("%d",&b[i]);
            printf("Enter the starting block of file%d",i+1);
            scanf("%d",&sb[i]);
            t[i]=sb[i];
            for(j=0;j<b[i];j++)
                        c[i][j]=sb[i]++;
            }
 printf("Filename\tStart block\tlength\n");
 for(i=0;i<n;i++)
            printf("%d\t  %d \t%d\n",i+1,t[i],b[i]);
 printf("Enter file name:");
 scanf("%d",&x);
 printf("File name is:%d",x);
 printf("length is:%d",b[x-1]);
 printf("blocks occupied:");
 for(i=0;i<b[x-1];i++)
            printf("%4d",c[x-1][i]);
 getch();
}
```

OUTPUT:

```
Enter no.of files: 2
Enter no. of blocks occupied by file1 4
Enter the starting block of file1 2
 Enter no. of blocks occupied by file2 10
Enter the starting block of file2 5
Filename        Start block     length
 1                    2               4
2                     5               10
Enter file name: rajesh
 File name is:12803 length is:0blocks occupied
```

Indexed File Allocation

(b) AIM: Write a C Program to implement Indexed File Allocation method.

. **Recommended Hardware / Software Requirements:**

- Hardware Requirements: Intel Based desktop PC LANS Connected with minimum of 166 MHZ or faster processor with at least 64 MB RAM and 100 MB free disk space.
- TC

Prerequisite

Theory:

Indexed allocation :
Indexed allocation supports both sequential and direct access files. Te file indexes are not physically stored as a part of the file allocation table. Whenever the file size increases, we can easily add some more blocks to the index. In this strategy, the file allocation table contains a single entry for each file. The entry consisting of one index block, the index blocks having the pointers to the other blocks. No external fragmentation.

Algorithm:

```
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 cosumer.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.
```

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
main()
{
 int n,m[20],i,j,sb[20],s[20],b[20][20],x;
 clrscr();
 printf("Enter no. of files:");
 scanf("%d",&n);
 for(i=0;i<n;i++)
 {       printf("Enter starting block and size of file%d:",i+1);
         scanf("%d%d",&sb[i],&s[i]);
         printf("Enter blocks occupied by file%d:",i+1);
```

```
          scanf("%d",&m[i]);
          printf("enter blocks of file%d:",i+1);
          for(j=0;j<m[i];j++)
                    scanf("%d",&b[i][j]);
 } printf("\nFile\t index\tlength\n");
 for(i=0;i<n;i++)
 {
          printf("%d\t%d\t%d\n",i+1,sb[i],m[i]);
 }printf("\nEnter file name:");
 scanf("%d",&x);
 printf("file name is:%d\n",x);
 i=x-1;
 printf("Index is:%d",sb[i]);
 printf("Block occupied are:");
 for(j=0;j<m[i];j++)
          printf("%3d",b[i][j]);
 getch();
}
```

OUTPUT:

```
Enter no. of files:2
 Enter starting block and size of file1: 2   5
Enter blocks occupied by file1:10
enter blocks of file1:3
2  5   4  6  7   2   6  4  7
Enter starting block and size of file2: 3   4
Enter blocks occupied by file2:5
enter blocks of file2: 2  3   4  5   6
File     index  length
 1       2       10
2       3       5
Enter file name: venkat
file name is:12803
Index is:0Block occupied are:
```

Linked File Allocation

c) AIM: Write a C Program to implement Linked File Allocation method.

. **Recommended Hardware / Software Requirements:**

- Hardware Requirements: Intel Based desktop PC LANS Connected with minimum of 166 MHZ or faster processor with at least 64 MB RAM and 100 MB free disk space.
- TC

Prerequisite

Theory:

Linked allocation :
It is easy to allocate the files, because allocation is on an individual block basis. Each block contains a pointer to the next free block in the chain. Here also the file allocation table consisting of a single entry for each file. Using this strategy any free block can be added to a chain very easily. There is a link between one block to another block, that's why it is said to be linked allocation. We can avoid the external fragmentation.

ALGORITHM:

```
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.
```

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
struct file
{
 char fname[10];
 int start,size,block[10];
}f[10];
main()
{
 int i,j,n;
 clrscr();
 printf("Enter no. of files:");
 scanf("%d",&n);
 for(i=0;i<n;i++)
 {
 printf("Enter file name:");
 scanf("%s",&f[i].fname);
 printf("Enter starting block:");
 scanf("%d",&f[i].start);
 f[i].block[0]=f[i].start;
 printf("Enter no.of blocks:");
 scanf("%d",&f[i].size);
 printf("Enter block numbers:");
 for(j=1;j<=f[i].size;j++)
 {
         scanf("%d",&f[i].block[j]);
 }
 }
 printf("File\tstart\tsize\tblock\n");
 for(i=0;i<n;i++)
 {
         printf("%s\t%d\t%d\t",f[i].fname,f[i].start,f[i].size);
         for(j=1;j<=f[i].size-1;j++)
                 printf("%d--->",f[i].block[j]);
         printf("%d",f[i].block[j]);
         printf("\n");
 }
 getch();
}
```

OUTPUT:

```
Enter no. of files:2
Enter file name:venkat
Enter starting block:20
Enter no.of blocks:6
Enter block numbers: 4
12
15
45
32
25
Enter file name:rajesh
Enter starting block:12
Enter no.of blocks:5
Enter block numbers:6
5
4
3
2
File    start   size    block
venkat  20      6       4--->12--->15--->45--->32--->25
rajesh  12      5       6--->5--->4--->3--->2
```

Sub Task 3:

AIM: A program to simulate the MVT(Multiprogram Variable Task)

. **Recommended Hardware / Software Requirements:**

- Hardware Requirements: Intel Based desktop PC LANS Connected with minimum of 166 MHZ or faster processor with at least 64 MB RAM and 100 MB free disk space.
- TC

Prerequisite

Theory:

MVT :
MVT stands for multiprogramming with variable number of tasks. Multiprogramming is a technique to execute number of programs simultaneously by a single processor. This is one of the memory management techniques. To eliminate the same of the problems with fixed partitions, an approach known as dynamic partitioning developed. In this technique, partitions are created dynamically, so that each process is loaded into partition of exactly the same size at that process. This scheme suffering from external fragmentation.

PROGRAM:
```
#include<stdio.h>
#include<conio.h>
void main()
{
int m=0,m1=0,m2=0,p,count=0,i;
clrscr();
printf("enter the memory capacity:");
scanf("%d",&m);
printf("enter the no of processes:");
scanf("%d",&p);
for(i=0;i<p;i++)
{
printf("\nenter memory req for process%d: ",i+1);
scanf("%d",&m1);
count=count+m1;
if(m1<=m)
{
if(count==m)
printf("there is no further memory remaining:");
printf("the memory allocated for process%d is: %d ",i+1,m);
m2=m-m1;
printf("\nremaining memory is: %d",m2);
m=m2;
}
else
{
printf("memory is not allocated for process%d",i+1);
}
printf("\nexternal fragmentation for this process is:%d",m2);
}
getch();
}
```

Input:

```
Input:
Enter the memory capacity: 80
Enter no of processes: 2
Enter memory req for process1: 23
```

Output:

```
The memory allocated for process1 is: 80
Remaining memory is: 57
External fragmentation for this process is: 57
Enter memory req for process2: 52
The memory allocated for process2 is: 57
Remaining memory is: 5
External fragmentation for this process is: 5
```

MULTIPROGRAM FIXED TASK

AIM: A Program to simulate the MFT

MFT:
MFT stands for multiprogramming with fixed no of tasks. MFT is the one of the memory management technique. In this technique, main memory is divided into no of static partitions at the system generated time. A process may be loaded into a partition of equal or greater size. The partition sizes are depending on o.s. in this memory management scheme the o.s occupies the low memory, and the rest of the main memory is available for user space. This scheme suffers from internal as well as external fragmentation

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
int main()
{
int m,p,s,p1;
int m1[4],i,f,f1=0,f2=0,fra1,fra2,s1,pos;
clrscr();
printf("Enter the memory size:");
scanf("%d",&m);
printf("Enter the no of partitions:");
scanf("%d",&p);
s=m/p;
printf("Each partn size is:%d",s);
printf("\nEnter the no of processes:");
scanf("%d",&p1);
pos=m;
for(i=0;i<p1;i++)
{
if(pos<s)
{
printf("\nThere is no further memory for process%d",i+1);
m1[i]=0;
break;
}
else
{
printf("\nEnter the memory req for process%d:",i+1);
scanf("%d",&m1[i]);
if(m1[i]<=s)
{
printf("\nProcess is allocated in partition%d",i+1);
fra1=s-m1[i];
printf("\nInternal fragmentation for process is:%d",fra1);
f1=f1+fra1;
pos=pos-s;
14OS Lab Manual By K. Ravi Chythanya
}
else
{
printf("\nProcess not allocated in partition%d",i+1);
s1=m1[i];
while(s1>s)
{
s1=s1-s;
pos=pos-s;
}
pos=pos-s;
fra2=s-s1;
f2=f2+fra2;
printf("\nExternal Fragmentation for this process is:%d",fra2);
}
}
}
printf("\nProcess\tallocatedmemory");
for(i=0;i<p1;i++)
printf("\n%5d\t%5d",i+1,m1[i]);
f=f1+f2;
printf("\nThe tot no of fragmentation is:%d",f);
getch();
return 0;
}
```

Input:

```
Enter the memory size: 80
Enter the no of partitions: 4
Each partition size: 20
Enter the number of processes: 2
Enter the memory req for process1: 18
```

Output:

```
Process1 is allocated in partn1
Internal fragmentation for process1 is: 2
Enter the memory req for process2: 22
Process2 is not allocated in partn2
External fragmentation for process2 is: 18
Process memory allocated
 1 20 18
 2 20 22
The tot no of fragmentation is: 20
```

SINGLE LEVEL ROOT DIRECTORY

a)AIM: To SIMULATION OF SINGLE LEVEL ROOT DIRECTORY

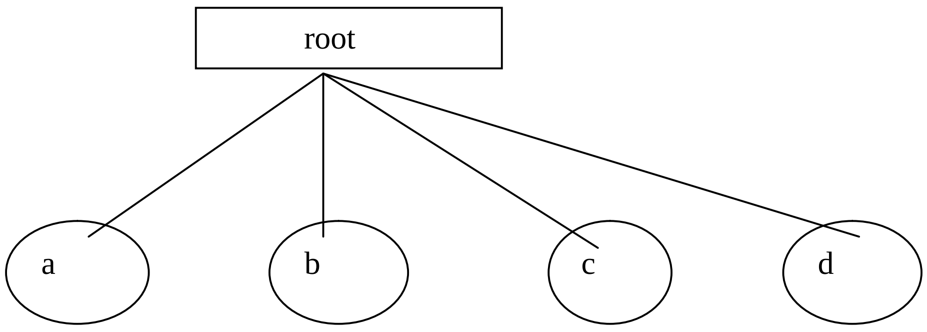
. **Recommended Hardware / Software Requirements:**

- Hardware Requirements: Intel Based desktop PC LANS Connected with minimum of 166 MHZ or faster processor with at least 64 MB RAM and 100 MB free disk space.
- TC

Prerequisite

Theory:

a)Single Level Directories: It is the simplest of all directory structures, in this the directory system having only one directory, it consisting of the all files. Sometimes it is said to be root directory. The following dig. Shows single level directory that contains four files (A, B, C, D).



Program:

```
#include<stdio.h>
#include<conio.h>
main()
{
int master,s[20];
char f[20][20][20];
char d[20][20];
int i,j;
clrscr();
printf("enter number of directorios:");
scanf("%d",&master);
printf("enter names of directories:");
for(i=0;i<master;i++)
scanf("%s",&d[i]);
printf("enter size of directories:");
for(i=0;i<master;i++)
scanf("%d",&s[i]);
printf("enter the file names :");
for(i=0;i<master;i++)
for(j=0;j<s[i];j++)
scanf("%s",&f[i][j]);
printf("\n");
printf(" directory\tsize\tfilenames\n");
printf("*************************************************\n");
for(i=0;i<master;i++)
{
printf("%s\t\t%2d\t",d[i],s[i]);
for(j=0;j<s[i];j++)
printf("%s\n\t\t\t",f[i][j]);
printf("\n");
}
printf("\t\n");
getch();
}
```

Output:

```
Enter number of directories:2
Enter names of directories: abc
Def
Enter size of directories:2
2
Enter file names:
Asd
Fgh
Jkl
Qwe
Directory           Size       Filename
*******************************
Abc                   2          asd
                                  Fgh
Def                   2          jkl
                                  Qwe
```

TWO LEVEL ROOT DIRECTORY

b) AIM: SIMULATION OF TWO LEVEL ROOT DIRECTORY

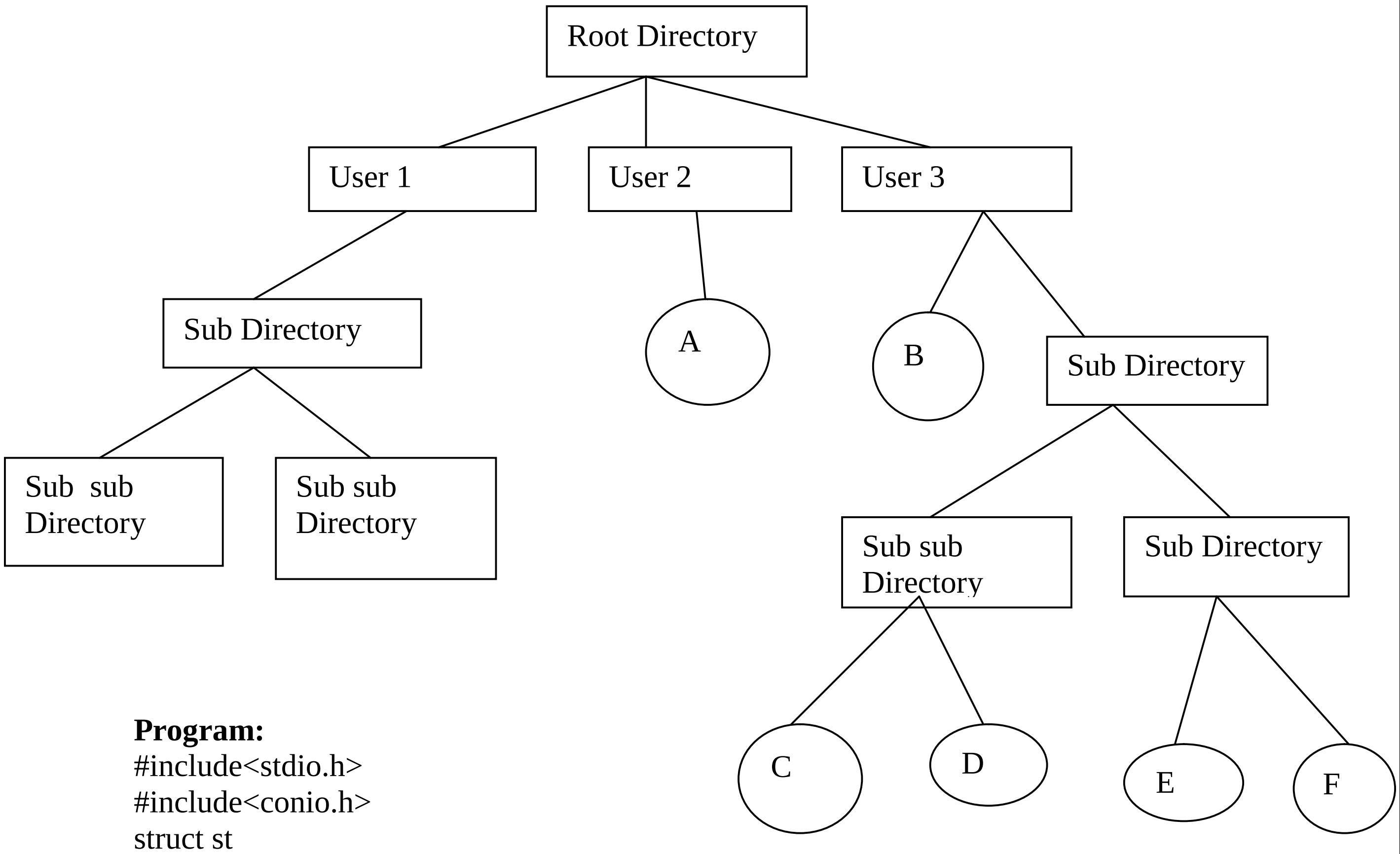
. **Recommended Hardware / Software Requirements:**

- Hardware Requirements: Intel Based desktop PC LANS Connected with minimum of 166 MHZ or faster processor with at least 64 MB RAM and 100 MB free disk space.
- TC

Prerequisite

Theory:

Two Level Directory: The problem in single level directory is different users may be accidentally using the same names for their files. To avoid this problem, each user need a private directory. In this way names chosen by one user don't interface with names chosen by a different user. The following dig 2-level directory Here root directory is the first level directory it consisting of entries of user directory.



Program:

```
#include<stdio.h>
#include<conio.h>
struct st
{
char dname[10];
char sdname[10][10];
char fname[10][10][10];
int ds,sds[10];
}dir[10];
```

```
void main()
{
int i,j,k,n;
clrscr();
printf("enter number of directories:");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("enter directory %d names:",i+1);
scanf("%s",&dir[i].dname);
printf("enter size of directories:");
scanf("%d",&dir[i].ds);
for(j=0;j<dir[i].ds;j++)
{
printf("enter subdirectory name and size:");
scanf("%s",&dir[i].sdname[j]);
scanf("%d",&dir[i].sds[j]);
for(k=0;k<dir[i].sds[j];k++)
{
printf("enter file name:");
scanf("%s",&dir[i].fname[j][k]);
}
}
}
printf("\ndirname\t\tsize\tsubdirname\tsize\tfiles");
printf("\n******************************************************\n");
for(i=0;i<n;i++)
{
printf("%s\t\t%d",dir[i].dname,dir[i].ds);
for(j=0;j<dir[i].ds;j++)
{
printf("\t%s\t\t%d\t",dir[i].sdname[j],dir[i].sds[j]);
for(k=0;k<dir[i].sds[j];k++)
printf("%s\t",dir[i].fname[j][k]);
printf("\n\t\t");
}
printf("\n");
   }
getch();
    }
```

OUTPUT:

```
Enter number of Directories: 2
Enter directory 1 name:abc
Enter size of directory:2
Enter subdirectory name and size:asd 2
Enter file name:qqq
Enter file name:www
Enter subdirectory name and size:sdf 2
Enter file name:eee
Enter file name:rrr
Enter directory 2 name: def
Enter size of directory:1
Enter sub directory name and size:wer 1
Enter file name :yui
```

Hierarchial Level Directory

c) AIM: To simulate Hierarchial Level Directory

. **Recommended Hardware / Software Requirements:**

- Hardware Requirements: Intel Based desktop PC LANS Connected with minimum of 166 MHZ or faster processor with at least 64 MB RAM and 100 MB free disk space.
- TC

Prerequisite

Theory:

Hierarchical Directory: The two level directories eliminate name conflicts among users but it is not satisfactory for users but it is not satisfactory for users with a large no of files. To avoid this, create the subdirectory and load the same type of the files into the subdirectory. So, in this method each can have as many directories are needed.

Source Code:

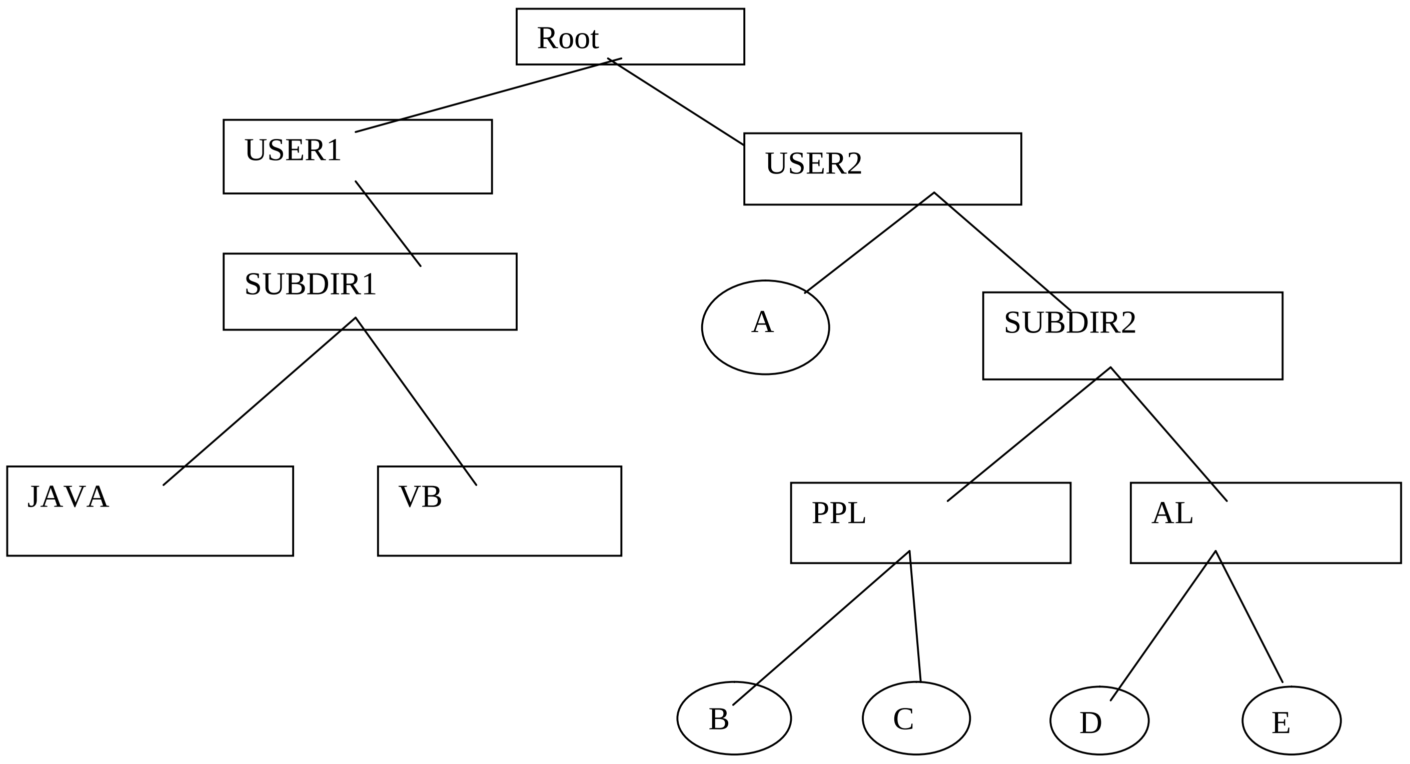
```
#include<stdio.h>
#include<graphics.h>
struct tree_element
{
char name[20];
int x,y,ftype,lx,rx,nc,level;
struct tree_element *link[5];
};
typedef struct tree_element node;
void main()
{
int gd=DETECT,gm;
node *root;
root=NULL;
clrscr();
```

```
create(&root,0,"root",0,639,320);
clrscr();
initgraph(&gd,&gm,"c:\\tc\\BGI");
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,BLUE);
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]);
}
}
}
```

OUTPUT:

```
Enter Name of dir/file (under root): ROOT
Enter 1 for Dir / 2 For File : 1
No of subdirectories / files (for ROOT) :2
Enter Name of dir/file (under ROOT): USER 1
Enter 1 for Dir /2 for file:1
No of subdirectories /files (for USER 1): 1
Enter Name of dir/file (under USER 1):SUBDIR
Enter 1 for Dir /2 for file:1
No of subdirectories /files (for SUBDIR): 2
Enter Name of dir/file (under USER 1):JAVA
Enter 1 for Dir /2 for file:1
No of subdirectories /files (for JAVA): 0
Enter Name of dir/file (under SUBDIR):VB
Enter 1 for Dir /2 for file:1
No of subdirectories /files (for VB): 0
Enter Name of dir/file (under ROOT):USER2
Enter 1 for Dir /2 for file:1
No of subdirectories /files (for USER2): 2
Enter Name of dir/file (under ROOT):A
Enter 1 for Dir /2 for file:2
Enter Name of dir/file (under USER2):SUBDIR 2
Enter 1 for Dir /2 for file:1
No of subdirectories /files (for SUBDIR 2): 2
Enter Name of dir/file (under SUBDIR2):PPL
Enter 1 for Dir /2 for file:1
No of subdirectories /files (for PPL): 2
Enter Name of dir/file (under PPL):B
Enter 1 for Dir /2 for file:2
Enter Name of dir/file (under PPL):C
Enter 1 for Dir /2 for file:2
Enter Name of dir/file (under SUBDIR):AI
Enter 1 for Dir /2 for file:1
No of subdirectories /files (for AI): 2
Enter Name of dir/file (under AI):D
Enter 1 for Dir /2 for file:2
Enter Name of dir/file (under AI):E
```

Enter 1 for Dir /2 for file:2



DAG(Directed Acyclic Graph)

d) AIM: To simulate DAG

. **Recommended Hardware / Software Requirements:**

- Hardware Requirements: Intel Based desktop PC LANS Connected with minimum of 166 MHZ or faster processor with at least 64 MB RAM and 100 MB free disk space.
- TC

Prerequisite

Theory:

General graph Directory: When we add links to an existing tree structured directory, the tree structure is destroyed, resulting in a simple graph structure. This structure is used to traversing is easy and file sharing also possible.

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,level;
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\\BGI");
draw_link_lines();
display(root);
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(stdin);
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;
}
}
/* displays the constructed tree in graphics mode */
display(node *root)
{
int i;
settextstyle(2,0,4);
settextjustify(1,1);
setfillstyle(1,BLUE);
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]);
}}}
```

OUTPUT:

```
Enter Name of dir/file (under root): ROOT
Enter 1 for Dir / 2 For File : 1
No of subdirectories / files (for ROOT) :2
Enter Name of dir/file (under ROOT): USER 1
Enter 1 for Dir /2 for file:1
No of subdirectories /files (for USER 1): 2
Enter Name of dir/file (under USER1): VB
Enter 1 for Dir /2 for file:1
No of subdirectories /files (for VB): 2
Enter Name of dir/file (under VB): A
Enter 1 for Dir /2 for file:2
Enter Name of dir/file (under VB): B
Enter 1 for Dir /2 for file:2
Enter Name of dir/file (under USER1): C
Enter 1 for Dir /2 for file:2
Enter Name of dir/file (under ROOT): USER2
Enter 1 for Dir /2 for file:1
No of subdirectories /files (for USER2): 1
Enter Name of dir/file (under USER2):JAVA
Enter 1 for Dir /2 for file:1
No of subdirectories /files (for JAVA):2
Enter Name of dir/file (under JAVA):D
Enter 1 for Dir /2 for file:2
Enter Name of dir/file (under JAVA):HTML
Enter 1 for Dir /2 for file:1
No of subdirectories /files (for HTML):0
How many links:2
File/Dir: B
User Name: USER 2
File/Dir: HTML
User Name: USER1
```

Sub Task-5

AIM: To Simulate BANKER'S ALGORITHM FOR DEADLOCK AVOIDANCE

. **Recommended Hardware / Software Requirements:**

- Hardware Requirements: Intel Based desktop PC LANS Connected with minimum of 166 MHZ or faster processor with at least 64 MB RAM and 100 MB free disk space.
- TC

Prerequisite

Theory:
Banker's Algorithm:

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

Data structures

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

Safety Algorithm

1. Work and Finish be the vector of length m and n respectively, Work=Available and Finish[i] =False.
2. Find an i such that both
   - Finish[i] =False
   - Need<=Work

   If no such I exists go to step 4.
3. work=work+Allocation, Finish[i] =True;
4. if Finish[1]=True for all I, then the system is in safe state.

Resource request algorithm

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

1. if Request<=Need I go to step 2. Otherwise raise an error condition.

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

If the resulting resource allocation state is safe, the transaction is completed and process Pi is allocated its resources. However if the state is unsafe, the Pi must wait for Request i and the old resource-allocation state is restored.

PROGRAM

```
#include<stdio.h>
#include<conio.h>
void main()
{
int req[10][10],r[10],av[10][10],a,max[10][10],curr[10][10],no[10],need[10][10],x=0;
int pno,rno,i,j,k,op,p,re,value,h;
clrscr();
printf("Nter the no.of processes");
scanf("%d",&pno);
printf("Nter the no of resourses");
scanf("%d",&rno);
printf("Nter the no.of resourses available");
for(i=0;i<rno;i++)
{
scanf("%d",&r[i]);
av[i]=r[i];
}
printf("Nter the %d * %dmaximum allocation matrix",pno,rno);
for(i=0;i<pno;i++)
for(j=0;j<rno;j++)
scanf("%d",&curr[i][j]);
printf("Nter need matrix is \n");
for(i=0;i<pno;i++)
{
for(j=0;j<rno;j++)
{
req[i][j]=max[i][j]-curr[i][j];
printf("%d",req[i][j]);
}
printf("\n");
}
value=safe(pno,rno);
if(value==0)
printf("\n\n it is not safe in the beginning ");
if(value==1)
printf("It is safe in the beginning");
do
```

```
{
printf("Is there any request");
scanf("%d",&p);
printF("Nter the resourses need of each kind");
for(h=0;h<rno;h++)
{
scanf("%d",&no[h]);
if(curr[p][h]+no[h]>max[p][h])
{
++x;
break;
}
}
if(x>0)
printf("\n Demand is more than Claim");
if(x==0)
{
for(h=0;h<rno;h++)
{
if(no[h]>av[h])
++x;
}
if(x>0)
printf("\n Resourses %d not available",h);
else
{
for(h=0;h<rno;h++)
{
av[h]=(av[h]-no[h]);
curr[p][h]=curr[p][h]+no[h];
need[p][h]=need[p][h]-no[h];
}
value=safe(pno,rno);
if(value==0)
printf("It is not safe");
else
printf("Safe state");
}
}
}
 printf("Nter for mole request");
 scanf("%d",&op);
 }
 while(op==1);
 int safe(int n,int k)
 {
 int rest[10],currav[10],temp[10],pos=0;
 int i,j,p,count=0,possible=1;
 for(i=0;i<n;i++)
 rest[i]=1;
 for(i=0;i<k;i++)
```

```
currav[i]=av[i];
while(possible)
{
j=0;
for(i=0;i<n;i++)
{
if(rest[i]!=0)
{
for(p=0;p<k;p++)
{
if(req[i][p]<=currav[p])
currav[p]=currav[p]+max[i][p];
else
break;
}
if(p==k)
{
rest[i]=0;
{
j++;
}
if(rest[i]==0)
{
count++;
temp[pos++]=i;
}
}
}
if(count==n)
possible=0;
if(j==0)
possible=0;
}
for(i=0;i<n;i++)
{
if(rest[i]!=0)
return(0);
}
printf("\n Resourse are allocated in this order:\n\n");
for(i=0;i<n;i++)
printf("P %d \t",temp[i]);
return(1);
}
}
```

OUTPUT:

```
Enter the No of Process : 4
Enter the No. of Resourses:2
Enter the No.of Resourses Available:1 1
Enter the 4*2 maximum allocation matrix: 4 2
1 2
1 3
3 2
Enter the 4*2 Current Matrix 1 2
0 1
1 0
2 0
The Need Matrix is:
3 0
1 1
0 3
1 2
Is there any request: 1
Enter the Process: 2
Enter the recourses needed  for each kind: 1
Demand is more than Claim
Resourses are allocated in thjis order

P1      P3      P0      P2
It is safe in the beginning
```

Sub Task 6:

AIM: To simulate BANKER’S ALGORITHM FOR DEAD LOCK PREVENTION

. **Recommended Hardware / Software Requirements:**

- Hardware Requirements: Intel Based desktop PC LANS Connected with minimum of 166 MHZ or faster processor with at least 64 MB RAM and 100 MB free disk space.
- TC

PROGRAM

```
#include<stdio.h>
#include<conio.h>
void main()
{
int pno,rno,i,j,k,op,p,re,n,max[10][10],curr[10][10];
int req[10][10],av[10],r[10],a;
clrscr();
printf("ENter no.of processes");
scanf("%d",&pno);
printf("Enter no of resourses");
for(i=0;i<rno;i++)
{
scanf("%d",&n[i]);
av[i]=r[i];
}
printf("Enter the %d to %d moreallocation matrix",pno,rno);
for(i=0;i<pno;i++)
for(j=0;j<pno;j++)
scanf("%d",&max[i][j]);
printf("\n Nter the %d %d current matrix \n",pno,rno);
for(i=0;i<pno;i++)
for(j=0;j<rno;j++)
scanf("%d",&curr[i][j]);
printf("The need matrix is \n");
for(i=0;i<pno;i++)
{
for(j=0;j<rno;j++)
{
req[i][j]=max[i][j]-curr[i][j];
printf("%d",req[i][j]);
}
printf("\n");
}
value=safe(pno,rno);
if(value==0)
```

```
printf("\n\n it is not safe in beginning");
else
printf("\n\n it is safe in the beginning");
getch();
}
int safe(int n,int k)
{
int rest[10],currav[10],temp[10],pos=0;
int i,j,p,count=0,possible=1;
for(i=0;i<n;i++)
rest[i]=1;
for(i=0;i<k;i++)
currav[i]=av[i];
while(possible)
{
j=0;
for(i=0;i<n;i++) {
if(rest[i]!=0)
{
for(p=0;p<k;p++)
{
if(req[i][p]<=currav[p])
currav[p]=currav[p]+max[i][p];
else
break;
}
if(p==k)
{
rest[i]=0;
{
j++;
}
if(rest[i]==0)
{
count++;
temp[pos++]=i;
}}
}
if(count==n)
possible=0;
if(j==0)
possible=0;
}
for(i=0;i<n;i++)
{
if(rest[i]!=0)
return(0);
}
printf("\n Resourses r allocated in this Order:\n \n");
for(i=0;i<n;i++)
printf("P %d \t", temp[i]);
return(1);
}
}
```

OUTPUT:

```
ENter no.of processes2
Enter no of resourses2
Nter the no of resourses1 2
Enter the 2 to 2 moreallocation matrix1 2
2 1

 Nter the 2 2 current matrix
1 2
2 1
The need matrix is
00
00

 Resourses r allocated in this Order:

P 0    P 1

 it is safe in the beginning
```

Sub Task-7

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

FIRST IN FIRST OUT

a)AIM:To Simulate FIRST IN FIRST OUT PAGE REPLACEMENT ALGORITHM

. **Recommended Hardware / Software Requirements:**

- Hardware Requirements: Intel Based desktop PC LANS Connected with minimum of 166 MHZ or faster processor with at least 64 MB RAM and 100 MB free disk space.
- TC

Prerequisite

Theory:

a) FIFO (First in First Out) algorithm: FIFO is the simplest page replacement algorithm, the idea behind this is, “Replace a page that page is oldest page of main memory” or “Replace the page that has been in memory longest”. FIFO focuses on the length of time a page has been in the memory rather than how much the page is being used.

ALGORITHM

```
Step1:    Start
Step2:    Global Declaration fifo(),t[5],pgf,n,a[20],I,j,frm
Step3:    pgf←fifo()
Step4:    Read pos,flag,i←0
Step5:    while i<n do
Step6:    for pos=0 to pos<frm & i<n do
Step7:    for j=0 to j<pos or j<frm do
Step8:    if a[i]=t[j] then
          Flag← 1, break
Step9:    if flag=1 then continue
Step10:   t [pos]← a[i]
Step11:   for j← 0 to j<frm do
          Print t[j]
Step12:   pgf← pgf+1,pos←pos+1
Step13:   return pgf to Step3
Step14:   end
```

PROGRAM

```
#include<stdio.h>
#include<conio.h>
int t[3],pgf=0,n,a[20],i,j,frm;
void main()
{
clrscr();
printf("Nter the no.of elements in reference string");
scanf("%d",&n);
printf("Nter the reference String");
for(i=0;i<n;i++)
scanf("%d",&a[i]);
printf("Nter the No of frames");
scanf("%d",&frm);
pgf=fifo();
printf("No of page fault is %d",pgf);
getch();
}
int fifo()
{
int pos,flag;
i=0;
while(i<n)
{
for(pos=0;pos<frm && i<n;i++)
{
if(a[i]==t[j])
{
flag=1;
break;
}
if(flag==1)
continue;
t[pos]=a[i];
printf("\n");
for(j=0;j<frm;j++)
{
printf("%d \t",t[j]);
}
printf("\n");
pgf++;
pos++;
}
}
return(pgf);
}
```

OUTPUT:

```
Nter the no.of elements in reference string3
Nter the reference String1 2 4
Nter the No of frames3

1       0       0

1       2       0
```

```
1       2       4
No of page fault is 3
```

LEAST RECENTLY USED PAGE REPLACEMENT

b)AIM: To simulate LEAST RECENTLY USED PAGE REPLACEMENT

. **Recommended Hardware / Software Requirements:**

- Hardware Requirements: Intel Based desktop PC LANS Connected with minimum of 166 MHZ or faster processor with at least 64 MB RAM and 100 MB free disk space.
- TC

Prerequisite

Theory:

b) LRU (Least Recently Used): the criteria of this algorithm is “Replace a page that has been used for the longest period of time”. This strategy is the page replacement algorithm looking backward in time, rather than forward.

ALGORITHM

```
Step1:      start
Step2:      read t[3],pgf←0,n,a[20],I,j,frm[10],nfm,k,min[10]
Step3:      for i=0 to i<nfm do
            Min[i]=-1
Step4:      pgf←lru()
Step5:      pos← -1,pres←0,m,l,p
Step6:      for i←0,pos←0 to pos←nfm & i<n do
Step7:      if pos>=1 then
            For m←0 to m<pos do
            If a[i]=frm[m] then
            Pres←1,break
Step8:      if pres=0 then
            Frm[pos++]←a[i],pg++
Step9:      for k←0 to k<nfm do
            Print frm[k] pres←0
Step10:     for i<n do
            For m←0 to m<nfm do
Step11:     if a[i]=frm[m] then
            Pres←1 break
Step12:     if pres=0 then
```

```
            For j=0 to j<nfm do
            For k=0 to k<I do
            If frm[j]=a[k] then
            Min[j]=k
            Pos=0
            L=min[0]
Step13:     for p←1 to p<nfm do
            If(min<p3<1) then
            L=min[p],pos=p
Step14:     frm[pos]←a[i]
Step15:     for k← 0 to k<nfm do
            Min[k]← -1
Step16:     pgf←pgf+1
Step17:     pres←0
Step18:     return pgf
Step19:     end
```

PROGRAM

```
#include<stdio.h>
#include<conio.h>
int t[3],pgf=0,n,a[20],i,j,frm[10],nfm,k;
int min[10];
void main()
{
printf("Nter the no of elements in reference string");
scanf("%d",&n);
printf("enter the reference string");
for
scanf("%d",&a[i]);
printf("Nternthe no of frames");
scanf("%d",&nfm);
for(i=0;i<nfm;i++)
min[i]=-1;
pgf(lru();
printf("No og page faults is %d ",pgf);
getch();
}
int lru()
{
int po=-1,pres=0;m,l,p;
for(i=0,pos=0;pos<nfm ,i<n;i++)
{
if(pos>=1)
for(m=0;m<pos;m++)
{
if(a[i]==frm[m])
{
pres=1;
break;
}
}
if(pres==0)
{
frm[pos++]=a[i];
pgf+
+;
}
for"(k=0;k<nfm;k++)
printf("%d",frm[k]);
printf("\n");
pres=0;
}
for(i=0;i<n;i++)
{
for(m=0;m<nfm;m++)
if(a[i]==frm[m])
{
pres=1;
break;
}
if(pres==0)
{
for(j=0;j<nfm;j++)
for(k=0;k<1;k++)
if(frm[j]==a[k])
{
min[j]=k;
}
pos=0;
l=min[0];
for(p=1;p<nfm;p++)
if(mi9n[p]<l)
{
l=min[p];
pos=p;
}
frm[pos]=a[i];
for(k=0;k<nfm;k++)
{
printf("%d",frm[k]);
min[k]=-1;
}
pgf++;
printf?("\n");
}
pres=0;
}
return pgf;
}
```

OUTPUT:

```
Enter the no of Elements in reference string 10
Enter the reference string 1 2 3 4 1 2 5 6 4 5
Enter the No. of Frames 4
1     0     0     0
1     2     0     0
1     2     3     0
1     2     3     4
1     2     5     4
1     2     5     6
4     2     5     6
No of page faults is : 7
```

Least Frequently Used page replacement algorithm

(c) AIM: To implement page replacement algorithms

. **Recommended Hardware / Software Requirements:**

- Hardware Requirements: Intel Based desktop PC LANS Connected with minimum of 166 MHZ or faster processor with at least 64 MB RAM and 100 MB free disk space.
- TC

Prerequisite

Theory:

c) LFU (Least Frequently Used): The least frequently used algorithm “select a page for replacement, if the page has not been used for the often in the past” or “Replace page that page has smallest count” for this algorithm each page maintains as counter which counter value shows the least count, replace that page. The frequency counter is reset each time is page is loaded.

Program:

```
#include<stdio.h>
#include<conio.h>
int i,j,nof,nor,flag=0,ref[50],frm[50],pf=0,victim=-1;
int recent[10],optcal[50],count=0;
int optvictim();
void main()
{
  clrscr();
  printf("\n................................");
  printf("\nEnter the no.of frames");
  scanf("%d",&nof);
  printf("Enter the no.of reference string");
  scanf("%d",&nor);
  printf("Enter the reference string");
  for(i=0;i<nor;i++)
     scanf("%d",&ref[i]);
  clrscr();
  printf("\n................................");
  printf("\nThe given string");
  printf("\n....................\n");
  for(i=0;i<nor;i++)
     printf("%4d",ref[i]);
  for(i=0;i<nof;i++)
  {
     frm[i]=-1;
     optcal[i]=0;
  }
  for(i=0;i<10;i++)
     recent[i]=0;
  printf("\n");
  for(i=0;i<nor;i++)
  {
    flag=0;
    printf("\n\tref no %d ->\t",ref[i]);
    for(j=0;j<nof;j++)
    {
        if(frm[j]==ref[i])
        {
          flag=1;
          break;
        }
    }
    if(flag==0)
    {
        count++;
        if(count<=nof)
           victim++;
        else
           victim=optvictim(i);
        pf++;
        frm[victim]=ref[i];
        for(j=0;j<nof;j++)
          printf("%4d",frm[j]);
    }
  }
  printf("\n Number of page faults: %d",pf);
  getch();
}
int optvictim(int index)
{
  int i,j,temp,notfound;
  for(i=0;i<nof;i++)
  {
    notfound=1;
    for(j=index;j<nor;j++)
        if(frm[i]==ref[j])
        {
           notfound=0;
           optcal[i]=j;
           break;
        }
    if(notfound==1)
          return i;
  }
  temp=optcal[0];
  for(i=1;i<nof;i++)
     if(temp<optcal[i])
          temp=optcal[i];
  for(i=0;i<nof;i++)
     if(frm[temp]==frm[i])
          return i;
 return 0;
}
```

OUTPUT:

```
  Enter no.of Frames....3
 Enter no.of reference string..6

 Enter reference string..
6 5 4 2 3 1

The given reference string:
       …………………. 6  5  4  2  3  1

       Reference NO 6->      6 -1 -1
       Reference NO 5->      6  5 -1
       Reference NO 4->      6  5  4
       Reference NO 2->      2  5  4
       Reference NO 3->        2   3   4
       Reference NO 1->        2   3   1

       No.of page faults...6
```

Sub Task-8
AIM: Simulate Paging technique of Memory Management
. **Recommended Hardware / Software Requirements:**

- Hardware Requirements: Intel Based desktop PC LANS Connected with minimum of 166 MHZ or faster processor with at least 64 MB RAM and 100 MB free disk space.
- TC

Prerequisite

Theory:
Paging is an efficient memory management scheme because it is non-contiguous memory allocation method. The basic idea of paging is the physical memory (main memory)
is divided into fixed sized blocks called frames, the logical address space is divided into fixed
sized blocks, called pages, but page size and frame size should be equal. The size of the frame or a page is depending on operating system.
In this scheme the operating system maintains a data structure that is page table, it
is used for mapping purpose. The page table specifies the some useful information, it tells which frames are there and so on. The page table consisting of two fields, one is the page number and other one is frame number. Every address generated by the CPU divided into two parts, one is page number and second is page offset or displacement. The pages are loaded into available free frames in the physical memory.

Algorithm:

Step 1: enter number of pages
Step 2: enter page size
Step 3:allocate memeory for the pages assigned by using malloc function
Step 4: prit the page number and page offset

PROGRAM:
```
 #include<stdio.h>
#include<conio.h>
 main()
{
int np,ps,i;
int *sa;
 clrscr();
printf("Enter how many pages\n");
scanf("%d",&np);
printf("Enter the page size \n");
scanf("%d",&ps);
for(i=0;i<np;i++)
{
sa[i]=(int)malloc(ps);
 printf("Page%d\t Address %u\n",i+1,sa[i]);
}
getch();
}
```

OUTPUT:

Input:

```
Enter how many pages: 5
Enter the page size: 4
```

Output:

```
Page1 Address: 1894
Page2 Address: 1902
Page3 Address: 1910
Page4 Address: 1918
Page5 Address: 1926
```

Operating Systems Viva questions

1. **What are the basic functions of an operating system?**

Operating system controls and coordinates the use of the hardware among the various applications programs for various uses. Operating system acts as resource allocator and manager. Since there are many possibly conflicting requests for resources the operating system must decide which requests are allocated resources to operating the computer system efficiently and fairly. Also operating system is control program which controls the user programs to prevent errors and improper use of the computer. It is especially concerned with the operation and control of I/O devices.

2. **Why paging is used?**

Paging is solution to external fragmentation problem which is to permit the logical address space of a process to be noncontiguous, thus allowing a process to be allocating physical memory wherever the latter is available.

3. **While running DOS on a PC, which command would be used to duplicate the entire diskette?**

Diskcopy

4. **What resources are used when a thread created? How do they differ from those when a process is created?**

–When a thread is created the threads does not require any new resources to execute the thread shares the resources like memory of the process to which they belong to. The benefit of code sharing is that it allows an application to have several different threads of activity all within the same address space. Whereas if a new process creation is very heavyweight because it always requires new address space to be created and even if they share the memory then the inter process communication is expensive when compared to the communication between the threads.

5. **What is virtual memory?**

Virtual memory is hardware technique where the system appears to have more memory that it actually does. This is done by time-sharing, the physical memory and storage parts of the memory one disk when they are not actively being used.

6. **What is Throughput, Turnaround time, waiting time and Response time?**

Throughput – number of processes that complete their execution per time unit. Turnaround time – amount of time to execute a particular process. 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).

7. **What is the state of the processor, when a process is waiting for some event to occur?**

Waiting state

8. **What is the important aspect of a real-time system or Mission Critical Systems?**

- A real time operating system has well defined fixed time constraints. Process must be done within the defined constraints or the system will fail. An example is the operating system for a flight control computer or an advanced jet airplane. Often used as a control device in a dedicated application such as controlling scientific experiments, medical imaging systems, industrial control systems, and some display systems. Real-Time systems may be either hard or soft real-time. **Hard real-time**: Secondary storage limited or absent, data stored in short term memory, or read-only memory (ROM), Conflicts with time-sharing systems, not supported by general-purpose operating systems. **Soft real-time**: Limited utility in industrial control of robotics, Useful in applications (multimedia, virtual reality) requiring advanced operating-system features.

9. **What is the difference between Hard and Soft real-time systems?**

- A hard real-time system guarantees that critical tasks complete on time. This goal requires that all delays in the system be bounded from the retrieval of the stored data to the time that it takes the operating system to finish any request made of it. A soft real time system where a critical real-time task gets priority over other tasks and retains that priority until it completes. As in hard real time systems kernel delays need to be bounded

10. **What is the cause of thrashing? How does the system detect thrashing? Once it detects thrashing, what can the system do to eliminate this problem?**

- Thrashing is caused by under allocation of the minimum number of pages required by a process, forcing it to continuously page fault. The system can detect thrashing by evaluating the level of CPU utilization as compared to the level of multiprogramming. It can be eliminated by reducing the level of multiprogramming.

11. **What is hard disk and what is its purpose?**

- Hard disk is the secondary storage device, which holds the data in bulk, and it holds the data on the magnetic medium of the disk. Hard disks have a hard platter that holds the magnetic medium, the magnetic medium can be easily erased and rewritten, and a typical desktop machine will have a hard disk with a capacity of between 10 and 40 gigabytes. Data is stored onto the disk in the form of files.

12. **What is fragmentation? Different types of fragmentation?**

- Fragmentation occurs in a dynamic memory allocation system when many of the free blocks are too small to satisfy any request. **External Fragmentation**: External Fragmentation happens when a dynamic memory allocation algorithm allocates some memory and a small piece is left over that cannot be effectively used. If too much external fragmentation occurs, the amount of usable memory is drastically reduced. Total memory space exists to satisfy a request, but it is not contiguous. **Internal Fragmentation**: Internal fragmentation is the space wasted inside of allocated memory blocks because of restriction on the allowed sizes of allocated blocks. Allocated memory may be slightly larger than requested memory; this size difference is memory internal to a partition, but not being used

13. **What is DRAM? In which form does it store data?**

- DRAM is not the best, but it's cheap, does the job, and is available almost everywhere you look. DRAM data resides in a cell made of a capacitor and a transistor. The capacitor tends to lose data unless it's recharged every couple of milliseconds, and this recharging tends to slow down the performance of DRAM compared to speedier RAM types.

14. **What is Dispatcher?**

- Dispatcher module gives control of the CPU to the process selected by the short-term scheduler; this involves: Switching context, Switching to user mode, Jumping to the proper location in the user program to restart that program, dispatch latency – time it takes for the dispatcher to stop one process and start another running.

15. **What is CPU Scheduler?**

- Selects from among the processes in memory that are ready to execute, and allocates the CPU to one of them. CPU scheduling decisions may take place when a process: 1.Switches from running to waiting state. 2.Switches from running to ready state. 3.Switches from waiting to ready. 4.Terminates. Scheduling under 1 and 4 is non-preemptive. All other scheduling is preemptive.

16. **What is Context Switch?**

- Switching the CPU to another process requires saving the state of the old process and loading the saved state for the new process. This task is known as a context switch. Context-switch time is pure overhead, because the system does no useful work while switching. Its speed varies from machine to machine, depending on the memory speed, the number of registers which must be copied, the existed of special instructions(such as a single instruction to load or store all registers).

17. **What is cache memory?**

- Cache memory is random access memory (RAM) that a computer microprocessor can access more quickly than it can access regular RAM. As the microprocessor processes data, it looks first in the cache memory and if it finds the data there (from a previous reading of data), it does not have to do the more time-consuming reading of data from larger memory.

18. **What is a Safe State and what is its use in deadlock avoidance?**

- When a process requests an available resource, system must decide if immediate allocation leaves the system in a safe state. System is in safe state if there exists a safe sequence of all processes. Deadlock Avoidance: ensure that a system will never enter an unsafe state.

19. **What is a Real-Time System?**

- A real time process is a process that must respond to the events within a certain time period. A real time operating system is an operating system that can run real time processes successfully

20. What is multi tasking, multi programming, multi threading?

Multi programming: Multiprogramming is the technique of running several programs at a time using timesharing.
It allows a computer to do several things at the same time. Multiprogramming creates logical parallelism.

Multi tasking: Multitasking is the logical extension of multiprogramming .The concept of multitasking is quite similar to multiprogramming but difference is that the switching between jobs occurs so frequently that the users can interact with each program while it is running. This concept is also known as time-sharing systems. A time-shared operating system uses CPU scheduling and multiprogramming to provide each user with a small portion of time-shared **system.**

21. What is the difference between process and thread.

Process is some job or task which is running in background.
while a thread is a single line of execution in a programs , so many threads can be there in a program.

22. What is Dispatcher?

Dispatcher module gives control of the CPU to the process selected by the short-term scheduler; this involves: Switching context switching to user mode jumping to the proper location in the user program to restart that program

Dispatch latency – time it takes for the dispatcher to stop one process and start another running.

23. What are the basic functions of an operating system?

Operating system controls and coordinates the use of the hardware among the various applications programs for various uses. Operating system acts as resource allocator and manager.

24. Why thread is called as a lightweight process?

Thread is a lightweight process becoz it simply divides a process into various subprocesses....each subprocess completes specific task and these threads run symoultaneously (parallel) resulting in speedy process completion & reduced resource usage

25. What is the difference between process and thread?

Threads share the address space of the process that created it; processes have their own address. Threads have direct access to the data segment of its process; processes have their own copy of the data segment

26. Define task and TCB?

A Process Control Block (PCB, also called Task Control Block or Task Struct) is a data structure in the operating system kernel containing the information needed to manage a particular process. The PCB is "the manifestation of a process in an operating system.

27. Explain briefly about, processor, assembler, compiler, loader, linker and the functions executed by

Processor:--A processor is the part a computer system that executes instructions .It is also called a CPU Assembler: -- An assembler is a program that takes basic computer instructions and converts

28. What is the state of the processor, when a process is waiting for some event to occur?

Waiting state

29. Why do multi-tasking operating systems in a single user machines today?

You can run muntiple application for a same time through multi-tasking operating system. For Example:- At a same time you can access MS Outlook to check the your intranet mail and same time you access your internet mails,For Ex- yahoo, hotmail

30. What is the difference between blocking and waiting state of process?

A process is said to be in waiting state when it is queuing in the main memory for its turn to be executed. Whereas, the process enters a blocked state in case of any interrupt or due to unavailability of resources.

31. Give an example of microkernel?

Amoeba, WinNT, Minix

32. Describe the relationship between a page file, and virtual memory on a Windows operating systems.

Virtual memory is the memory taken form hard disk.Swap: When the programme excuting in the main memory is completly shifted to Virtual Memory for the execution. Then it is called the Swaping.

33. What do you mean by deadlock?

Deadlock is a situation where a group of processes are all blocked and none of them can become unblocked until one of the other becomes unblocked.

34. What resources are used when a thread created? How do they differ from those when a process is created?

When a thread is created the threads does not require any new resources to execute the thread shares the resources like memory of the process to which they belong to.

35. What is the main component of operating system?

not a DOS (disk oprating system)only shell and kernal.

36. What are the latest versions of MSDOS?

Dos 7.0.

37. What is Throughput, Turnaround time, waiting time and Response time?

Throughput – number of processes that complete their execution per time unit

Turnaround time – amount of time to execute a particular process

Waiting time – amount of time a process has

38. Give a non-computer example of preemptive scheduling?

Consider any system where people use some kind of resources and compete for them. The non-computer examples for preemptive scheduling the traffic on the single lane road if there is emergency.

39. Different types of Real-Time Scheduling?

Hard real-time systems – required to complete a critical task within a guaranteed amount of time.

Soft real-time computing – requires that critical processes receive priority over less fortunate

40. Describe different job scheduling in operating systems?

a. FCFS b. SJF preemptive c. SJF non-preemptive d. Round Robin e. priority

41. Define Demand Paging?

Demand Paging: Demand paging is the paging policy that a page is not read into memory until it is requested, that is, until there is a page fault on the page.

42. What is page fault?

Page fault is the situation when the CPU want to access a page from the cache but it doesn't find there. So it has to go to the main memory for the particular page.

43. List of CPU Scheduling algorithms implemented in Windows Operating System

1) Solaris 2 Uses priority-based process scheduling

2) Windows 2000 uses a priority-based preemptive scheduling algorithm

44. Differentiate between Complier and Interpreter?

An interpreter reads one instruction at a time and carries out the actions implied by that instruction. It does not perform any translation. But a compiler translates the entire instructions.

45. Why paging is used?

Paging is solution to external fragmentation problem which is to permit the logical address space of a process to be noncontiguous, thus allowing a process to be allocating physical memory.

46. Common Functions of Interrupts?

->Interrupt transfers control to the interrupt service routine generally, through the interrupt vector, which contains the addresses of all the service routines

47. Difference between Primary storage and secondary storage?

Main memory: – only large storage media that the CPU can access directly.

Secondary storage: – extension of main memory that provides large nonvolatile storage capacity.

48. What are the Methods for Handling Deadlocks?

->Ensure that the system will never enter a deadlock state.

->Allow the system to enter a deadlock state and then recover.

->Ignore the problem and pretend that deadlocks never occur in the system;

49. What is Semaphore?

Locking Mechanism used inside resource mangers and resource dispensers.

50. What is the difference between preemptive scheduling and time slicing?

Under preemptive scheduling, the highest priority task executes until it enters the waiting or dead states or a higher priority task comes into existence. Under time slicing, a task executes for a predefined slice of time and then reenters the pool of ready tasks. The scheduler then determines which task should execute next, based on priority and other factors.

51. What is Mutex?

Mutex is a program object that allows multiple program threads to share the same resource, such as file access, but not simultaneously. When a program is started a mutex is created with a unique name. After this stage, any thread that needs the resource must lock the mutex from other threads while it is using the resource. the mutex is set to unlock when the data is no longer needed or the routine is finished

52. What are the Reasons for Process Termination?

Normal completion, Time limit exceeded ,Memory unavailable, Bounds violation Protection error, Arithmetic error, Time overrun I/O failure ,Invalid instruction Privileged instruction ,Data misuse Operator or OS intervention ,Parent termination

53. What is busy waiting?

The repeated execution of a loop of code while waiting for an event to occur is called busy-waiting. The CPU is not engaged in any real productive activity during this period, and the process does not progress toward completion.

54. What is a Phantom deadlocks?

In distributed deadlock detection, the delay in propagating local information might cause the deadlock detection algorithms to identify deadlocks that do not really exist. Such situations are called phantom deadlocks and they lead to unnecessary aborts

55. Difference between a process and a program?

Process is program under execution. Program is only a set of instructions and process is a program in memory. Thread is a basic unit of execution within a process. This means program is something which is not loaded in memory yet.

56. Which is the first operating system?

The GM-NAA I/O input/output system of General Motors and North American Aviation was the first operating system in the history of computer science.It was created in 1956 by Bob Patrick of General Motors and Owen Mock of North American Aviation.

57. What is the shortest definition of os?

OS is a Interface between a user and the hardware of a computer.

58. What is the difference between UNIX and windows operating systems?

In terms of file systems UNIX is more secure than windows. UNIX is multiuser, where as windows is single user.

59. Difference between time sharing and multitasking systems?

Time Sharing: programs are executed in an interleaved manner that is 1st the programs are merged together to form a single stream of instruction then given to the processor for execution.

Multitasking: programs are kept separate and cup switches in between.

60. What is the difference between multi programming operating system and multiprocessing operating system?

A multi processing os supports multiple processes to run with the help of some scheduling algorithm(round robin) its mean u can run multiple process at same time, there are two type of multi processing os 1) multiprocessing :- single Processor and multiple process 2) symmetric multiprocessing :- multiple Processor and single processor

A multiprogramming OS supports all features like 1) Multi tasking 2) Multi processing 3} Multi threading (thread can also be considered as a task) but if any of these feature is not supported by the os then that os can’t be multiprogramming os

61. What is GUI?

- Short for Graphical User Interface, a GUI Operating System contains graphics and icons and is commonly navigated by using a computer mouse. See our GUI dictionary definition for a complete definition. Below are some examples of GUI Operating Systems.

System 7.x, Windows 98, Windows CE

62. What is Multi-user OS?

- A multi-user Operating System allows for multiple users to use the same computer at the same time and/or different times. See our multi-user dictionary definition for a complete definition for a complete definition. Below are some examples of multi-user Operating Systems.

Linux, UNIX, Windows 2000, Windows XP, Mac OS X

63. What is Multiprocessing OS?

- An Operating System capable of supporting and utilizing more than one computer processor. Below are some examples of multiprocessing Operating Systems.

Linux, UNIX, Windows 2000, Windows XP, Mac OS X

64. What is Multitasking OS?

- An Operating system that is capable of allowing multiple software processes to run at the same time. Below are some examples of multitasking Operating Systems.

UNIX, Windows 2000, Windows XP, Mac OS X

65. What is Multithreading OS?

- Operating systems that allow different parts of software program to run concurrently. Operating systems that would fall into this category are:

Linux, UNIX, Windows 2000, and Windows XPMac OS X

66. What information is stored in process control block?

- Process control block contains process identifier, state, priority, program counter, memory pointer, I/O status information etc.

67. What are the various process states?
- Process states are running, ready, blocked, new, exit

68. What are the UNIX process states?
- UNIX process states are user running, kernel running, ready to run(in memory), asleep in memory, ready to run(swapped), sleeping, preempted , created , zombie.

69. What are four basic thread operations?
- spawn, block, unblock, finish.

70. What are the two categories of thread implementation?
-User-level threads, kernel-level threads

71. When race condition occurs?
- A race condition occurs when multiple processes or threads read and write data items so that the final result depends on the order of execution of instructions in the multiple processes.

72. What is critical resource and critical section?
- during the course of execution each process is sending commands to the i/o device, receiving status information, sending data , and/or receiving data, such resources are referred as critical resource and the portion of the program that uses it is called as critical section of the program.

73. What are the two type of semaphore?
- Binary semaphore, non-binary semaphore

74. What is another name for non-binary semaphore?
- Counting or General semaphore

75. What is strong semaphore?
- The process that has been blocked the longest is released from the queue first; a semaphore whose definition includes this policy is called a strong semaphore.

76. What is weak semaphore?
- A semaphore that does not specify the order in which processes are removed from the queue is a weak semaphore.

77) What is Monitor?

- A Monitor is a software module consisting of one or more procedures, an initialization sequence, and local data.

78) Monitor supports synchronization by what?
-Condition variables.

79) Condition variables are operated by which functions?
-cwait (), csignal ()

80) What are the primitives used for message passing?
-Send & receive

81) What is resource allocation graph?
-A useful tool in characterizing the allocation of resources to processes is called as resource allocation graph.

82) What are the conditions that must be present for a deadlock to happen?
-Mutual exclusion, Hold & wait, No preemption

83) What is safe state?
-A safe state is one in which there is at least one sequence of resource allocation to processes that does not result in deadlock.

84) What is unsafe state?
- An unsafe state is a state that is not safe.

85) What are the mechanisms used for IPC in UNIX?
- Pipes, messages, shared memory, semaphores, signals.

86) What are the memory management requirements?
- Relocation, protection, sharing, logical organization, physical organization

87) What is technique used to overcome fragmentation?
- Compaction

88) What is external fragmentation?
- As time goes on, memory becomes more & more fragmented, and memory utilization declines, this is called as external fragmentation.

89) What are the three placement algorithms?
- Best-fit, first-fit, and next-fit.

90) What is logical address?
- A logical address is a reference to memory location independent of the current assignment of data to memory.

91) What is physical address?
- A physical address is an actual location in main memory.

92) What is translation lookaside buffer?
- TLB is a special high-speed cache for page table entries.

93) Which are the fetch policies?
- Demand paging, prepaging.

94) What are the various replacement policies?
- Optimal, least recently used, first–in-first-out, clock

95) What is optimal policy?
- Optimal policy selects for replacement that page for which the time to the next reference is the longest.

96) What is LRU policy?
- LRU policy replaces the page in memory that has not been referenced for the longest time

97) What is first-in-first-out policy?
- FIFO policy treats the page frames allocated to a process as a circular buffer, and pages are removed in round robin style.

98) What is a local replacement policy?
- A local replacement policy chooses only among the resident pages of the process that generated the page fault in selecting a page to replace.

99) What is a global replacement policy?
- A global replacement policy considers all unlocked pages in main memory as candidates for replacement, regardless of which process owns a particular page.

100) What is fixed-allocation policy?
- A fixed-allocation policy gives a process a fixed no. of frames in main memory within which to execute.

101) What is variable-allocation policy?
- A variable-allocation policy allows the no. of page frames allocated to a process to be varied over the lifetime of the process.

102) What are various types of scheduling?
- Long-term scheduling, medium-term scheduling, short-term scheduling, i/o scheduling.

103) What is non preemptive scheduling?
- in this once a process is in the running state, it continues to execute until it terminates or it blocks itself to wait for i/o or to request some operating system service.

104) What is preemptive scheduling?
- In this the currently running process may be interrupted and moved to the ready state by the operating system.