COMPUTER NETWORKS AND OPERATING SYSTEM LABORATORY MANUAL

B.TECH (R17)
(III YEAR – I SEM)

(2019-2020)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY

(Autonomous Institution – UGC, Govt. of India)

Recognized under 2(f) and 12 (B) of UGC ACT 1956
Affiliated to JNTUH, Hyderabad, Approved by AICTE - Accredited by NBA & NAAC – 'A' Grade - ISO 9001:2015 Certified)
Maisammaguda, Dhulapally (Post Via. Hakimpet), Secunderabad – 500100, Telangana State, India

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

VISION

- To improve the quality of technical education that provides efficient software engineers with an attitude to adapt challenging IT needs of local, national and international arena, through teaching and interaction with alumni and industry.

MISSION

- Department intends to meet the contemporary challenges in the field of IT and is playing a vital role in shaping the education of the 21st century by providing unique educational and research opportunities.

PROGRAMME EDUCATIONAL OBJECTIVES (PEOs)

PEO1 – ANALYTICAL SKILLS

To facilitate the graduates with the ability to visualize, gather information, articulate, analyze, solve complex problems, and make decisions. These are essential to address the challenges of complex and computation intensive problems increasing their productivity.

PEO2 – TECHNICAL SKILLS

To facilitate the graduates with the technical skills that prepare them for immediate employment and pursue certification providing a deeper understanding of the technology in advanced areas of computer science and related fields, thus encouraging to pursue higher education and research based on their interest.

PEO3 – SOFT SKILLS

To facilitate the graduates with the soft skills that include fulfilling the mission, setting goals, showing self-confidence by communicating effectively, having a positive attitude, get involved in team-work, being a leader, managing their career and their life.

PEO4 – PROFESSIONAL ETHICS

To facilitate the graduates with the knowledge of professional and ethical responsibilities by paying attention to grooming, being conservative with style, following dress codes, safety codes, and adapting themselves to technological advancements.

PROGRAM SPECIFIC OUTCOMES (PSOs)

After the completion of the course, B. Tech Information Technology, the graduates will have the following Program Specific Outcomes:

1. **Fundamentals and critical knowledge of the Computer System:-** Able to Understand the working principles of the computer System and its components , Apply the knowledge to build, asses, and analyze the software and hardware aspects of it .

2. **The comprehensive and Applicative knowledge of Software Development:** Comprehensive skills of Programming Languages, Software process models, methodologies, and able to plan, develop, test, analyze, and manage the software and hardware intensive systems in heterogeneous platforms individually or working in teams.

3. **Applications of Computing Domain & Research:** Able to use the professional, managerial, interdisciplinary skill set, and domain specific tools in development processes, identify the research gaps, and provide innovative solutions to them.

PROGRAMOUTCOMES (POs)

Engineering Graduates should possess the following:

1. **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **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.
3. **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.
4. **Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **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.
6. **The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **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.
8. **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **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.
11. **Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multi-disciplinary environments.
12. **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.

MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY

(UGC-Autonomous Institution , Govt. of India)

(Permanently Affiliated to JNTUH, Approved by AICTE-Accredited by NBA & NAAC- A-Grade; ISO 9001:2008 Certified)
Maisammaguda, Dhulapally Post, Via Hakimpet, Secunderabad – 500100

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Computer Networks & Operating System Lab Manual (R17A0587)

TABLE OF CONTENTS

S.No	**List of Experiments in CN**	Page.no
1.	Implement the data link layer framing methods such as character count, character stuffing and bit stuffing	1-11
2	Implement on a data set of characters the three CRC polynomials CRC 12, CRC 16 and CRC CCIP	12-15
3	Implement Dijkstra's algorithm to compute the shortest path thru a graph	16-18
4	Take an example subnet graph with weights indicating delay between nodes	19-20
5	Now obtain Routing table art each node using distance vector routing algorithm	21-23
6	Take an example subnet of hosts. Obtain broadcast tree for it	24-25
7	Take a 64 bit playing text and encrypt the same using DES algorithm.	26-33
8	Write a program to break the above DES coding	34-38
9	Using RSA algorithm Encrypt a text data and Decrypt the same.	39-43
	List of Experiments in OS	
1	Simulate the following CPU scheduling algorithm a) FCFS b) SJF c) Round Robin d) Priority	45-55
2	Simulate MVT & MFT	56-61
3	Simulate all page replacement algorithms a) FIFO b) LRU c)OPTIMAL	62-71
4	Simulate all file organization techniques a) Single level b)Two level	72-78
5	Simulate all File Allocation Strategies **a)** Sequential B)Indexed C)Linked	79-86
6	Simulate Bankers Algorithm for Deadlock Avoidance	87-91

MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY
Maisammaguda, Dhulapally Post, Via Hakimpet, Secunderabad – 500100

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

GENERAL LABORATORY INSTRUCTIONS

1. Students are advised to come to the laboratory at least 5 minutes before (to starting time), those who come after 5 minutes will not be allowed into the lab.
2. Plan your task properly much before to the commencement, come prepared to the lab with the synopsis / program / experiment details.
3. Student should enter into the laboratory with:
   a. Laboratory observation notes with all the details (Problem statement, Aim, Algorithm, Procedure, Program, Expected Output, etc.,) filled in for the lab session.
   b. Laboratory Record updated up to the last session experiments and other utensils (if any) needed in the lab.
   c. Proper Dress code and Identity card.
4. Sign in the laboratory login register, write the TIME-IN, and occupy the computer system allotted to you by the faculty.
5. Execute your task in the laboratory, and record the results / output in the lab observation note book, and get certified by the concerned faculty.
6. All the students should be polite and cooperative with the laboratory staff, must maintain the discipline and decency in the laboratory.
7. Computer labs are established with sophisticated and high end branded systems, which should be utilized properly.
8. Students / Faculty must keep their mobile phones in SWITCHED OFF mode during the lab sessions. Misuse of the equipment, misbehaviors with the staff and systems etc., will attract severe punishment.
9. Students must take the permission of the faculty in case of any urgency to go out ; if anybody found loitering outside the lab / class without permission during working hours will be treated seriously and punished appropriately.
10. Students should LOG OFF/ SHUT DOWN the computer system before he/she leaves the lab after completing the task (experiment) in all aspects. He/she must ensure the system / seat is kept properly.

Head of the Department **Principal**

MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY
Maisammaguda, Dhulapally Post, Via Hakimpet, Secunderabad – 500100

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

OBJECTIVES AND OUTCOMES

OBJECTIVES:

- To understand the functionalities of various layers of OSI model
- To explain the difference between hardware, software; operating systems, programs and files.
- Identify the purpose of different software applications.

OUTCOMES: By the end of the semester, students will be able

- Understand fundamental underlying principles of computer networking.
- Understand details and functionality of layered network architecture.
- Apply mathematical foundations to solve computational problems in computer networking.
- Describe and demonstrate the functions and features of current operating systems
- Demonstrate proficiency in common industry software applications (word processing, spreadsheet, presentation, and database) to effectively communicate in a professional business setting
- Demonstrate skills that meet industry standards and certification requirements in the use of system hardware, operating systems technologies, and application systems.

RECOMMENDED SYSTEM / SOFTWARE REQUIREMENTS:

1. Intel based desktop PC with minimum of 166MHz or faster processor with at least 64 MB RAM and 100 MB free disk space.

2. C ++ Compiler or Unix/Linux

USEFUL TEXT BOOKS / REFEREECES / WEBSITES :

1. An Introduction to Operating Systems, P.C.P Bhatt, 2nd edition, PHI.
2. Modern Operating Systems, Andrew S Tanenbaum, 3rd Edition, PHI

1. a) Implement character stuffing on given data

Algorithm:

Step 1: Initially give the user 2 choices, whether to character stuff or to directly exit, if wrong choice is entered then prompt an invalid choice message.
Step 2: Intake from the user the number of characters which are to be character stuffed. Step 3: Then the characters which are to be stuffed are to be taken inside the for loop.
Step 4: Original data is displayed and the characters to be stuffed at the start and end of the frame are uploaded in the program.
Step 5: If DLE character is present then stuff DLE character before it.
Step 6: The characters DLESTX are inserted at the start and end of the data. Step 7: The data along with the stuffed characters are displayed
Step 8: The original data is recovered and displayed on the receiving side Step 9: Stop

Program :

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
void charc(void);
void main()
{
int
choice;
while(1
)
{
printf("\n\n\n1.character stuffing");
printf("\n\n2.exit");
printf("\n\n\nenterchoice");
scanf("%d",&choice);
printf("%d",choice);
if(choice>2)
printf("\n\n invalid option ...please renter");
switch(choice)
{
case
1:charc();
break;
case 2:_exit(0);
}
}
}
void charc(void)
{
clrscr();
char
c[50],d[50],t[50];
int i,m,j;
printf("enter the number of characters\n");
scanf("%d",&m);printf("\n enter the
characters\n"); for(i=0;i<m+1;i++)
{
scanf("%c",&c[i]);
}
printf("\n original data\n");
for(i=0;i<m+1;i++)
printf("%c",c[i]);d[0]='d';
d[1]='l';d[2]='e';
d[3]='s';d[4]='t';
d[5]='x';
for(i=0,j=6;i<m+1;i++,j++)
{
if((c[i]=='d'&&c[i+1]=='l'&& c[i+2]=='e'))
{
d[j]='d'; j++;
d[j]='l'; j++;
d[j]='e'; j++;
m=m+3;
}
d[j]=c[i];
}
m=m+6; m++;
d[m]='d'; m++;
d[m]='l'; m++;
d[m]='e'; m++;
d[m]='s'; m++;
d[m]='t'; m++;
d[m]='x';
m++;
printf("\n\n transmitted data: \n"); for(i=0;i<m;i++)
{
printf("%c",d[i]);
}
for(i=6,j=0;i<m-6;i++,j++)
{
if(d[i]=='d'&&d[i+1]=='l'&&d[i+2]=='e'&&d[i+3]=='d'&&d[i+4]=='l'&&d[i+5]=='e')
i=i+3;
t[j]=d[i];
}
printf("\n\nreceived data:"); for(i=0;i<j;i++)
```

```
{
printf("%c",t[i]);
}
}
```

o/p:

1 b) Implement character count on given data

Algorithm for Character count

1. Start
2. Append DLE STX at the beginning of the string
3. Check the data if character is present; if character DLE is present in the string (example DOODLE) insert another DLE in the string (ex: DOODLEDLE)
4. Transmit DLE ETXat the end of the string
5. Display the string
6. Stop

PROGRAM:

```
#include<stdio.h> #include<string.h>
main()
{
int i,j,k,l,count=0,n; char s[100],cs[50];
clrscr();
printf("\n ENTER THE BIT STRING:");
gets(s);
n=strlen(s);
printf("\nTHE STRING IS\n");
for(i=0;i<n;)
     {
if(s[i]==s[i+1])
{
 count=2; i++;
 while(s[i]==s[i+1])
 { i++;
 count++;
 }
 if(count>=5)
 {
 printf("$");
 if(count<10)
 printf("0");
 printf("%d%c",count,s[i]);
 i++;
 }
 else
 {
 for(j=0;j<count;j++)
 printf("%c",s[i]);
 i++;
 }
 }
 else
 {
 printf("%c",s[i]);
 i++;
 }
 }
   getch(); }
```

INPUT/OUTPUT:

```
 ENTER THE BIT STRING:

123AAAAAAAAAATYKKKPPPP

P THE STRING IS

123$10ATYKKK$05P
```

b) Decode the stuffed data

Algorithm for Character De−stuffing

1. Start

2. Neglect initial DLE STX

3. If DLE is present in the text, ngelect it; if another DLE follows, copy the same to output. 4. Neglect the trailing DLE ETX

5. Stop

PROGRAM:

```
#include<stdio.h>
#include<string.h>
main()
{
int i,j,k,l,n,count;
char s[100],cs[50]; clrscr();
printf("\n ENTER THE STUFFED STRING :");
gets(s); n=strlen(s);
printf("\nTHE STRING IS\n");
for(i=0;i<n;)
   {
      if(s[i]=='$')
      { i++;
       count=(s[i]-'0')*10+(s[i+1]-'0');
       if(count<5)
       { clrscr();
         printf("INVALIDE MESSAGE");
         exit(1);
       }
       while(count>0)
       {
       printf("%c",s[i+2]);
       count--;
       }
       i=i+3;
      }
        else
       {
       printf("%c",s[i]); i++;
       }
    }
  getch();    }
```

<u>INPUT/OUTPUT:</u>

```
ENTER THE STUFFED STRING: 123$10ATY$06K THE STRING IS:
123AAAAAAAAAATYKKKKKK
```

c) Bit stuffing on given binary data

Algorithm for Bit−Stuffing

1. Start

2. Initialize the array for transmitted stream with the special bit pattern 0111 1110 which indicates the beginning of the frame.

3. Get the bit stream to be transmitted in to the array.

4. Check for five consecutive ones and if they occur, stuff a bit 0

5. Display the data transmitted as it appears on the data line after appending 0111 1110 at the end

6. For de−stuffing, copy the transmitted data to another array after detecting the stuffed bits7. Display the received bit stream

8. Stop

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
main()
{
int b[100],b1[100],l,k,n=0,i,j,z,i1,s[20],f[8]={0,1,1,1,1,1,1,0},j1;
static int a[100]; char
ch='y',bs[50]; clrscr();
do
{
i1=z=n=0; clrscr();
printf("\n Enter the bit string(space for each byte)");
 gets(bs);
for(i=0;bs[i]!='\0';i++)
if(bs[i]!=' ')
b[n++]=bs[i]-'0';
for(i=0;i<n;i++)
{
if(b[i]==1)
{
 i1++;
if(i1==5){s[z++]=i+1;i1=0;
}
}
else i1=0;
} j1=j=0;
for(i=0;i<z;i++)
{
while(j<s[i])
b1[j1++]=b[j
++];
b1[j1++]=0;
}
while(j1<n+z)
b1[j1++]=b[j++];
l=n/8;
for(i=0;l>0;i++)
{
a[i]=l%2; l=l/2;
}
printf("\nAfter stuffing :");
for(j=7;j>=0;j--)
printf("%d",a[j]);
printf(" ");
for(k=0;k<8;k++)
 printf("%d",f[k]);
 printf("    ");
for(k=0;k<j1;k++)
printf("%d",b1[k]);
printf(" ");
for(k=0;k<8;k++)
 printf("%d",f[k]);
printf("\n\n Do u want to continue?");
 ch=getch();
}
while(ch=='y' || ch=='Y');
getch();
}
```

INPUT/OUTPUT:

```
 Enter the bit string (space for each
 byte) 11111111 01111110 00111110

After stuffing   : 00000011 01111110 111110111011111010001111100  01111110
```

d)Destuff the given stuffed data frame

```
include<stdio.h>
#include<conio.h>
 #include<math.h>
 main()
{
int i,n,n1,k,j,ni,len;
char f[8]={'0','1','1','1','1','1','1','0'},st[100];
static int ds[100];
clrscr();
printf("\n\nEnter the stuffed data");
 gets(st);
n=strlen(st);
ni=k=0;
        for(i=8;i<16;i++)
         if(st[i]!=f[k++])
        {
        printf("\nError in flag");
        exit(1);
        }
        k=0;
        for(i=n-8;i<n;i++)
        if(f[k++]!=st[i])
        {
        printf("\nError in flag");
        exit(1);
        }
        for(i=0;i<n;i++)
        st[i]=st[i]-'0';
        len=0;
        j=7;
        for(i=0;i<8;i++)
        len+=pow(2,i)*st[j--];
         k=ni=j=0;
        for(i=16;i<n-8;i++)
        {
        if(st[i]==1)
        {
         ni++;
        if(ni==5)
        {
        ds[j++]=1;
         i++;
        k++;
        ni=0
        ;}
        else
        ds[j++]=1;
        }
        else
        ds[j++]=0;
        }
        n1=n-24-k;
         if(len*8!=n1)
        {
        printf("\n Error in data length");
        exit(1);
        }
        printf("\n After destuffing    ");
        for(i=0;i<n1;i++)
        printf("%d",ds[i]);
getch();
}
```

INPUT/OUTPUT:

```
Enter the stuffed data000000110111111011111011101111101000111110001111110
 After destuffing  111111110111101000111100
```

Exercise:

a) **Implement K-Bit run length code on given data**
b) **Decode the K-Bit run length code**

2. a)Generate CRC code for a given data

frame Algorithm

1. A string of n as is appended to the data unit. The length of predetermined divisor is n+ 1.

2. The newly formed data unit 1. A string of n as is appended to the data unit. The length of predetermined divisor is n+ 1.

i.e. original data + string of n as are divided by the divisor using binary division and remainder is obtained. This remainder is called CRC.

3. Now, string of n Os appended to data unit is replaced by the CRC remainder (which is also of n bit).

4. The data unit + CRC is then transmitted to receiver.

5. The receiver on receiving it divides data unit + CRC by the same divisor & checks the remainder.

6. If the remainder of division is zero, receiver assumes that there is no error in data and it accepts it.

7. If remainder is non-zero then there is an error in data and receiver rejects it.

PROGRAM:

```
#include<stdio.h>
#include<math.h>
main()
{

int  i,j,k,m,n,cl;
char a[10],b[100],c[100];
clrscr();
printf("\n ENTER POLYNANOMIAL:");
scanf("%s",a);
printf("\n ENTER THE FRAME:");
scanf("%s",b);
m=strlen(a);
n=strlen(b);
for(i=0;i<m;i++) /* To eliminat first zeros in
polynomial */
{
 if(a[i]=='1')
 {
   m=m-i; break;
 }
}
 for(k=0;k<m;k++) /* To Adjust the polynomial
 */ a[k]=a[k+i];



 cl=m+n-1;
for(i=0;i<n;i++) /* To copy the original frame to c[]*/
c[i]=b[i];
for(i=n;i<cl;i++) /* To add n-1 zeros at the end of frame */
c[i]='0';
c[i]='\0';        /*To make it as a string */
for(i=0;i<n;i++) /* To set polynomial remainder at end of c[]*/
if(c[i]=='1')
{
for(j=i,k=0;k<m;k++,j++)
if(a[k]==c[j])
c[j]='0';
else c[j]='1';
}
for(i=0;i<n;i++) /* To copy original data in c[] */ c[i]=b[i];
printf("\n THE MESSAGE IS: %s",c);
getch();
}
```

INPUT/OUTPUT:

```
ENTER POLYNANOMIAL:1011 ENTER THE FRAME:10011101

THE MESSAGE IS: 10011101011


ENTER POLYNANOMIAL:00101 ENTER THE FRAME:10101011

THE MESSAGE IS: 1010101101
```

b) Verify the CRC code

```
#include<stdio.h>
#include<math.h>
main()
{

int  i,j,k,m,n,cl;
char a[10],c[100]; clrscr();
printf("\n ENTER POLYNANOMIAL:");
scanf("%s",a);
printf("\n ENTER THE CRC FRAME:");
scanf("%s",c); m=strlen(a);
cl=strlen(c);
for(i=0;i<m;i++) /* To eliminat first zeros in polynomial
*/
{
  if(a[i]=='1')
  { m=m-i; break; }

}
  for(k=0;k<m;k++) /* To Adjust the polynomial */
  a[k]=a[k+i];
     n=cl-m+1;
   for(i=0;i<n;i++) /* To check polynomial remainder is
  zero or not */ if(c[i]=='1')
  {
  for(j=i,k=0;k<m;k++,j++) if(a[k]==c[j])
  c[j]='0';
  else c[j]='1';
  }
  for(i=0;i<cl;i++) /* To copy original data in c[] */
  if(c[i]=='1')
  {
  printf("\n THERE IS SOME ERROR IN MESSAGE :");
  break;
  }
   if(i==cl)
   printf("\n MESSAGE IS CORRECT");
  getch();  }
```

<u>INPUT/OUTPUT :</u>

```
ENTER POLYNANOMIAL: 1011

ENTER THE CRC FRAME:

10101011101 THERE IS SOME

ERROR IN MESSAGE:


ENTER POLYNANOMIAL: 01011

ENTER THE CRC FRAME:

10011101011 MESSAGE IS

CORRECT
```

Exercise:

1. *a)* **Implement Hamming code Generation for a given binary code**

b) Hamming code verification and checking

2. **Implement even and odd parity for given binary code**

3. Implement Dijkstra’s algorithm to compute the shortest path through a graph

Algorithm:

1. Assign to every node a tentative distance value: set it to zero for our initial node and to infinity for all other nodes.
2. Mark all nodes unvisited. Set the initial node as current. Create a set of the unvisited nodes called the unvisited set consisting of all the nodes.
3. For the current node, consider all of its unvisited neighbors and calculate their tentative distances. For example, if the current node A is marked with a distance of 6, and the edge connecting it with a neighbor B has length 2, then the distance to B (through A) will be 6 + 2 = 8. If this distance is less than the previously recorded tentative distance of B, then overwrite that distance. Even though a neighbor has been examined, it is not marked as "visited" at this time, and it remains in the unvisited set.
4. When we are done considering all of the neighbors of the current node, mark the current node as visited and remove it from the unvisited set. A visited node will never be checked again.
5. If the destination node has been marked visited (when planning a route between two specific nodes) or if the smallest tentative distance among the nodes in the unvisited set is infinity (when planning a complete traversal; occurs when there is no connection between the initial node and remaining unvisited nodes), then stop. The algorithm has finished.
6. Select the unvisited node that is marked with the smallest tentative distance, and set it as the new "current node" then go back to step 3.

PROGRAM:

```
#include<stdio.h>
#include<string.h>
#include<math.h>
main()
{
int u,v,num,i,j,l,k,s[10],min,cost[10][10],dist[10],path[10],n;
clrscr();
printf("\n ENTER VERTECES:");
scanf("%d",&n);
printf("\n ENTER ADJECENCY MATRIX:\n");
for(i=1;i<=n;i++)
{
for(j=1;j<=n;j++)
scanf("%d",&cost[i][j]);
}

for(i=1;i<=n;i++)
for(j=1;j<=n;j++)
    if(i==j) cost[i][j]=0;
    else if(cost[i][j]==-1)
    cost[i][j]=30000;
    printf("\nENTER SOURCE VERTEX:");
    scanf("%d",&v); clrscr();
    for(i=1;i<=n;i++)
    {
      s[i]=0;
      path[i]=v; dist[i]=cost[v][i];
    }
      dist[v]=0;
      for(num=2;num<=n;num++)
      {
      min=30000; u=0;
           for(i=1;i<=n;i++)
           {
              if(s[i]!=1) if(min>dist[i])
             {
              u=i;  min=dist[i];
             }
       }
             s[u]=1;
           for(i=1;i<=n;i++)
           { if(s[i]!=1)
              if(dist[i]>(min+cost[u][i]))
              {
                 dist[i]=min+cost[u][i];
                 path[i]=u;
              }
           }
      }
    printf("\n");
    printf("\nPATH MATRIX:\n");
    printf("\nDISTANCE   NODE    PATH\n");
    for(i=1;i<=n;i++)
    { printf("\n %d",dist[i]);
    printf(" %d ",i);
       j=i;
       do
       {
       printf(" --> %d ",path[j]);
           u=path[j];
           j=u;
       }while(u!=v);
    }
   getch();}
```

INPUT/OUTPUT:

```
 ENTER VERTECES:8

 ENTER ADJECENCY MATRIX:


 0  2 -1 -1 -1 -1  6 -1
 2  0  7 -1  2 -1 -1 -1
-1  7  0  3 -1  3 -1 -1
-1 -1  3  0 -1 -1 -1  2
 -1 2 -1 -1  0 2  1  -1
 -1 -1 3 -1  2 0 -1   2
 6 -1 -1 -1  1 -1  0  4
 -1 -1 -1 2  -1 2  4   0

ENTER SOURCE VERTEX:1


PATH MATRIX:

DISTANCE   NODE    PATH

 0   1  --> 1
 2   2  --> 1
 9   3  --> 2  --> 1
 10 4  --> 8  --> 6 --> 5 --> 2 --> 1
 4   5  --> 2  --> 1
 6   6  --> 5  --> 2 --> 1
 5   7  --> 5  --> 2 --> 1
 8   8  --> 6  --> 5 --> 2 --> 1
```

Exercise:

1. **Write a program for implementing link state routing using Dijkstra’s algorithm**
2. **Write a program for Flooding algorithm**

4. Take an example subnet graph with weights indicating delay between nodes

```
 #include<stdio.h>
  #include<conio.h>
 struct full
 {
char line[10],dest[10];
int hops;
 }f[20];
   main()
 {
  int
  nv,min,minver,i;
  char
  sv[2],temp;
  clrscr();
  printf("\nEnter number of vertices:");
  scanf("%d",&nv);
  printf("\n Enter source vertex: ");
  scanf("%s",sv);
  printf("\n Enter full table for source vertex %s :\n",sv);
 for(i=0;i<nv;i++)
 scanf("%s %s %d",f[i].dest,f[i].line,&f[i].hops);
 printf("\n HIERARCHIAL TABLE\n\n");
 for(i=0;i<nv;)
  {
  if(sv[0]==f[i].dest[0])
  {
  printf("\n %s %s %d",f[i].dest,f[i].line,f[i].hops);
   i++;
  }
  else
  {
  min=1000; minver=0;
  temp=f[i].dest[0];
  while(temp==f[i].dest[0])
  {
  if(min>f[i].hops)
  {
  min=f[i].hops; minver=i;
  } i++;
  }
printf("\n %c %s %d ",temp,f[minver].line,f[minver].hops);
}
  }
  getch();
 }
```

INPUT/OUTPUT:

```
Enter number of vertices: 8

Enter source vertex :1A

Enter full table for source vertex 1A
: 1A - -
1B 1B 1
1C 1C 1
2A 1B 1
2B 1B 2
3A 1C 2
3B 1C 3
4A 1C 3

 HIERARCHIAL

 TABLE

 1A - 0
 1B 1B 1
 1C 1C 1
 2  1B 1
 3  1C 2
 4  1C 3
```

Exercise:

1. **Implement path vector routing protocol.**
2. **Routing Information Protocol**

5. Now obtain Routing table for each node using distance vector routing

algorithm Algorithm:

Input: Graph and a given vertex *src*
Output: Shortest distance to all vertices from *src*. If there is a negative weight cycle, then shortest distances are not calculated, negative weight cycle is reported.

1) This step initializes distances from source to all vertices as infinite and distance to source itself as 0. Create an array dist[] of size |V| with all values as infinite except dist[src] where src is source vertex.
2) This step calculates shortest distances. Do following |V|-1 times where |V| is the number of vertices in given graph.
…..**a)** Do following for each edge v-u
………………If dist[v] > dist[u] + weight of edge uv, then update dist[v]
………………….dist[v] = dist[u] + weight of edge uv
3) This step reports if there is a negative weight cycle in graph. Do following for each edge u-v
……If dist[v] > dist[u] + weight of edge uv, then "Graph contains negative weight cycle" The idea of step 3 is, step 2 guarantees shortest distances if graph doesn't contain negative weight cycle. If we iterate through all edges one more time and get a shorter path for any vertex, then there is a negative weight cycle

Program:

```
#include<stdio.h>
#include<math.h> #include<conio.h> main()
{
int i,j,k,nv,sn,noadj,edel[20],tdel[20][20],min;
char sv,adver[20],ch;
clrscr();
printf("\n ENTER THE NO.OF VERTECES:");
scanf("%d",&nv);
printf("\n    ENTER    THE    SOURCE    VERTEX
NUM,BER AND NAME:");
scanf("%d",&sn);
flushall(); sv=getchar();
printf("\n NETER NO.OF ADJ VERTECES TO
VERTEX %c",sv);
scanf("%d",&noadj);
for(i=0;i<noadj;i++)
{
printf("\n ENTER TIME DELAY and NODE NAME:");
scanf("%d %c",&edel[i],&adver[i]);
}
for(i=0;i<noadj;i++)
{
 printf("\n ENTER THE TIME DELAY FROM %c to ALL OTHER
 NODES: ",adver[i]);
 for(j=0;j<nv;j++)
 scanf("%d",&tdel[i][j]);
}

printf("\n DELAY VIA--VERTEX \n ");
 for(i=0;i<nv;i++)
{
   min=1000; ch=0;
  for(j=0;j<noadj;j++)
  if(min>(tdel[j][i]+edel[j]))
  {
   min=tdel[j][i]+edel[j];
   ch=adver[j];
  }
   if(i!=sn-1)
    printf("\n%d   %c",min,ch);
     else
    printf("\n0     -");
  }
 getch();
}
```

INPUT/OUTPUT:

```
 ENTER THE NO.OF VERTECES:12

 ENTER THE SOURCE VERTEX NUMBER AND NAME:10  J


 ENTER NO.OF ADJ VERTECES TO

 VERTEX 4 ENTER TIME DELAY and NODE

 NAME:8 A ENTER TIME DELAY and NODE

 NAME:10 I ENTER TIME DELAY and NODE

 NAME:12 H ENTER TIME DELAY and

 NODE NAME:6  K

 ENTER THE TIME DELAY FROM A to ALL OTHER
NODES: 0 12 25 40 14 23 18 17 21 9 24 29

 ENTER THE TIME DELAY FROM I to ALL OTHER
NODES: 24 36 18 27 7 20 31 20 0 11 22 33

 ENTER THE TIME DELAY FROM H to ALL OTHER
NODES: 20 31 19 8 30 19 6 0 14 7 22 9

 ENTER THE TIME DELAY FROM K to ALL OTHER
NODES: 21 28 36 24 22 40 31 19 22 10 0 9

 DELAY  VIA--VERTEX
8   a
20   a
28   i
20   h
17   i
30   i
18   h
12   h
10   i
0   -
6   k
15 k
```

Exercise:

1. **Routing table for each node using hierarchical routing algorithm**

2. Open Shortest Path First

6. Take an example subnet of hosts. Obtain broadcast tree for it.

/*PROGRAM TO IMPLEMENT BROADCAST ROUTING ALGORITHM*/

Algorithm:

- A router creates a data packet and then sends it to each host one by one. In this case, the router creates multiple copies of single data packet with different destination addresses. All packets are sent as unicast but because they are sent to all, it simulates as if router is broadcasting.

  This method consumes lots of bandwidth and router must destination address of each node.

- Secondly, when router receives a packet that is to be broadcasted, it simply floods those packets out of all interfaces. All routers are configured in the same way.

Program:

```
#include<stdio.h>
int a[10][10],n;
void main()
{
int i,j,root;
clrscr();
printf("Enter no.of nodes:");
scanf("%d",&n);
printf("Enter adjacent matrix\n");
for(i=1;i<=n;i++)
for(j=1;j<=n;j++)
{
printf("Enter connecting of %d-->%d::",i,j);
scanf("%d",&a[i][j]);
}
printf("Enter root node:");
scanf("%d",&root);
adj(root);
}
adj(int k)
{
int i,j;
printf("Adjacent node of root node::\n");
printf("%d\n\n",k);
for(j=1;j<=n;j++)
{
if(a[k][j]==1 || a[j][k]==1)
printf("%d\t",j);
}
printf("\n");
for(i=1;i<=n;i++)
{
if((a[k][j]==0) && (a[i][k]==0) && (i!=k))
printf("%d",i);
}
}
```

OUTPUT
****

```
Enter  no.of  nodes:5
Enter adjacent matrix
Enter connecting of 1-->1::0
Enter connecting of 1-->2::1
Enter connecting of 1-->3::1
Enter connecting of 1-->4::0
Enter connecting of 1-->5::0
Enter connecting of 2-->1::1
Enter connecting of 2-->2::0
Enter connecting of 2-->3::1
Enter connecting of 2-->4::1
Enter connecting of 2-->5::0
Enter connecting of 3-->1::1
Enter connecting of 3-->2::1
Enter connecting of 3-->3::0
Enter connecting of 3-->4::0
Enter connecting of 3-->5::0
Enter connecting of 4-->1::0
Enter connecting of 4-->2::1
Enter connecting of 4-->3::0
Enter connecting of 4-->4::0
Enter connecting of 4-->5::1
Enter connecting of 5-->1::0
Enter connecting of 5-->2::0
Enter connecting of 5-->3::0
Enter connecting of 5-->4::1
Enter connecting of 5-->5::0
Enter root node:2
Adjacent node of root node::
2
1 3 4
5
```

Exercise: 1. Implement Core-Based Tree(CBT) protocol obtain broadcast tree for it.
2. Implement an optimal algorithm for Broadcasting multiple messages in trees.

7. Take a 64 bit plain text and encrypt the same using DES algorithm.

/* Program to implement DES */

Algorithm:

1.) Firstly, we need to process the key.

1.1 Get a 64-bit key from the user. (Every 8th bit is considered a parity bit. For a key to have correct parity, each byte should contain an odd number of "1" bits.)

1.2 Calculate the key schedule.

1.2.1 Perform the following permutation on the 64-bit key. (The parity bits are discarded, reducing the key to 56 bits. Bit 1 of the permuted block is bit 57 of the original key, bit 2 is bit 49, and so on with bit56 being bit 4 of the original key.)

1.2.2 Split the permuted key into two halves. The first 28 bits are called C[0] and the last 28 bits are called D[0].

1.2.3 Calculate the 16 subkeys. Start with i = 1.

1.2.3.1 Perform one or two circular left shifts on both C[i-1] and D[i-1] to get C[i] and D[i], respectively.

1.2.3.3 Loop back to 1.2.3.1 until K[16] has been calculated.

2 Process a 64-bit data block.

2.1 Get a 64-bit data block. If the block is shorter than 64 bits, it should be padded as appropriate for the application.

2.2 Perform the following permutation on the data block.

Initial Permutation (IP)

2.3 Split the block into two halves. The first 32 bits are called L[0], and the last 32 bits are called R[0].

2.4 Apply the 16 subkeys to the data block. Start with i = 1.

2.4.1 Expand the 32-bit R[i-1] into 48 bits according to the bit-selection

2.4.2 Exclusive-or E(R[i-1]) with K[i].

2.4.3 Break E(R[i-1]) xor K[i] into eight 6-bit blocks. Bits 1-6 are B[1], bits 7-12 are B[2], and so on with bits 43-48 being B[8].

2.4.4 Substitute the values found in the S-boxes for all B[j]. Start with j = 1. All values in the S-boxes should be considered 4 bits wide.

2.4.4.1 Take the 1st and 6th bits of B[j] together as a 2-bit value (call it m) indicating the row in S[j] to look in for the substitution.

2.4.4.2 Take the 2nd through 5th bits of B[j] together as a 4-bit value (call it n) indicating the column in S[j] to find the substitution.

2.4.4.3 Replace B[j] with S[j][m][n].

Substitution Box 1 (S[1])

2.4.4.4 Loop back to 2.4.4.1 until all 8 blocks have been replaced.

2.4.5 Permute the concatenation of B[1] through B[8] as indicated below.

Permutation P

2.4.6 Exclusive-or the resulting value with L[i-1]. Thus, all together, your R[i] = L[i-1] xor P(S[1](B[1])...S[8](B[8])), where B[j] is a 6-bit block of E(R[i-1]) xor K[i]. (The function for R[i] is written as, R[i] = L[i-1] xor f(R[i-1], K[i]).)

2.4.7 L[i] = R[i-1].

2.4.8 Loop back to 2.4.1 until K[16] has been applied.

2.5 Perform the following permutation on the block

R[16]L[16]. Final Permutation (IP**-1)

Program:

```
#include<stdio.h>
#include<conio.h>
void main()
{

int k[15],k1[15],k2[15],i,j,p[15],p1[15],p2[15];
int p10[10]={3,5,2,7,4,10,1,9,8,6};
int p8[10]={6,3,7,4,8,5,10,9};
int t1,t2,t3,t4,t[4],b[10],s,h,a;
int ip[8]={2,6,3,1,4,8,5,7};
int ep[8]={4,1,2,3,2,3,4,1};
int ip1[8]={4,1,3,5,7,2,8,6};
int p4[4]={2,3,4,1};
int s0[4][4]={{1,0,3,2},{3,2,1,0},{0,2,1,3},{3,1,3,2}};
int s1[4][4]={{0,1,2,3},{2,0,1,3},{3,0,1,0},{2,1,0,3}};
clrscr();
printf("\n\tSimplified-DES\n");
printf("\n");
printf("\nEnter the plain text of 8 bits length::\n");
for(i=0;i<8;i++)
scanf("%d",&p2[i]);
```

```
printf("\n\nEnter the key of 10 bits length::\n");
for(i=0;i<10;i++)
scanf("%d",&k[i]);
printf("\n\nKey Generation::\n");
for(i=0;i<10;i++)
{
j=p10[i];
p[i]=k[j-1];
}
t1=p[0];
t2=p[5];
for(i=0;i<4;i++)
p[i]=p[i+1];
p[i]=t1;
for(i=5;i<9;i++)
p[i]=p[i+1];
p[i]=t2;
for(i=0;i<8;i++)
{
j=p8[i];
k1[i]=p[j-1];
}
t1=p[0];
t2=p[1];
t3=p[5];
t4=p[6];
for(i=0;i<3;i++)
{
p[i]=p[i+2];
}
p[i]=t1;
i++;
p[i]=t2;
for(i=5;i<8;i++)
p[i]=p[i+2];
p[i]=t3;
i++;
p[i]=t4;
for(i=0;i<8;i++)
{
j=p8[i];
k2[i]=p[j-1];
}
printf("\nkey k1::");
for(i=0;i<8;i++)
printf("%d",k1[i]);
printf("\nkey k2::");
for(i=0;i<8;i++)
printf("%d",k2[i]);
for(i=0;i<8;i++)
p[i]=p2[i];
```

```
for(a=0;a<2;a++)
{
if(a==0)
{
for(i=0;i<8;i++)
{
j=ip[i];
p1[i]=p[j-1];
}
}
for(i=0;i<4;i++)
{
if(a==0)
k[i]=p1[i+4];
if(a==1)
{
k[i]=p[i+4];
for(i=0;i<8;i++)
b[i]=p[i];
}
}
for(i=0;i<8;i++)
{
j=ep[i];
p[i]=k[j-1];
}
for(i=0;i<8;i++)
{
if(a==0)
{
if(p[i]==k1[i])
k[i]=0;
else
k[i]=1;
}
if(a==1)
{
if(p[i]==k2[i])
k[i]=0;
else
k[i]=1;
}
}
j=0;
for(i=0;i<8;i=i+4)
{
if(k[i]==0&&k[i+3]==0)
{
t[j]=0;
j++;
}
if(k[i]==0&&k[i+3]==1)
{
t[j]=1;
j++;
}
if(k[i]==1&&k[i+3]==0)
{
t[j]=2;
j++;
}
if(k[i]==1&&k[i+3]==1)
{
t[j]=3;
j++;
}
if(k[i+1]==0&&k[i+2]==0)
{
t[j]=0;
j++;
}
if(k[i+1]==0&&k[i+2]==1)
{
t[j]=1;
j++;
}
if(k[i+1]==1&&k[i+2]==0)
{
t[j]=2;
j++;
}
if(k[i+1]==1&&k[i+2]==1)
{
t[j]=3;
j++;
}
}
s=s0[t[0]][t[1]];
h=s1[t[2]][t[3]];
if(s==0)
{
k[0]=0;
k[1]=0;
}
if(s==1)
{
k[0]=0;
k[1]=1;
}
```

```
if(s==2)
{
k[0]=1;
k[1]=0;
}
if(s==3)
{
k[0]=1;
k[1]=1;
}
if(h==0)
{
k[2]=0;
k[3]=0;
}
if(h==1)
{
k[2]=0;
k[3]=1;
}
if(h==2)
{
k[2]=1;
k[3]=0;
}
if(h==3)
{
k[2]=1;
k[3]=1;
}
for(i=0;i<4;i++)
{
j=p4[i];
p[i]=k[j-1];
}
for(i=0;i<4;i++)
{
if(a==0)
{
if(p1[i]==p[i])
k[i]=0;
else
k[i]=1;
}
if(a==1)
{
if(b[i]==p[i])
k[i]=0;
else
k[i]=1;
}}
```

```
if(a==0)
{
for(i=0;i<4;i++)
p[i]=p1[i+4];
for(i=0;i<4;i++)
p[i+4]=k[i];
}
if(a==1)
{
for(i=4;i<8;i++)
k[i]=b[i];
for(i=0;i<8;i++)
{
j=ip1[i];
p[i]=k[j-1];
}
}
}
printf("\n\nThe cipher text::");
for(i=0;i<8;i++)
printf("%d",p[i]);
getch(); }
```

OUTPUT
****

```
Simplified-DES

Enter the plain text of 8 bits length::

1

0

1

0

1

0

1

0
```

```
Enter the key of 10 bits length::

1

0
1

0

1

0

1

0

1

0

Key Generation::

key k1::11100100

key k2::01010011

The cipher text::01100110
```

Exercise:

1. **Take a plain text and implement AES algorithm**
2. **Implement a Blowfish algorithm.**

8. Write a program to break the above DES coding.

```
#include<stdio.h>

#include<conio.h>

#include<string.h>

int p10[]={3,5,2,7,4,10,1,9,8,6},

p8[]={6,3,7,4,8,5,10,9},

p4[]={2,4,3,1};

int ip[]={2,6,3,1,4,8,5,7},

ipinv[]={4,1,3,5,7,2,8,6},

ep[]={4,1,2,3,2,3,4,1};

int s0[][4]={{1,0,3,2,},{3,2,1,0},{0,2,1,3,},{3,1,3,2}};

int s1[][4]={{0,1,2,3},{2,0,1,3},{3,0,1,0},{2,1,0,3}};

void permute(char op[],char ip[],int p[], int n)
{

int i;

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

op[i]=ip[p[i]-1];

op[i]='\0';

}
void circularls(char pr[],int n)

{

int i;

char ch=pr[0];

for(i=0;i<n-1;i++)

pr[i]=pr[i+1];
```

```
pr[i]=ch;

}

void keygen(char k1[],char k2[],char key[])

{

char keytemp[11];

permute(keytemp,key,p10,10);

circularls(keytemp,5);

circularls(keytemp+5,5);

permute(k1,keytemp,p8,8);

circularls(keytemp,5);

circularls(keytemp,5);

circularls(keytemp+5,5);

circularls(keytemp+5,5);

permute(k2,keytemp,p8,8);

}

void xor(char op[],char ip[])

{

int i;

for(i=0;i<strlen(op)&&i<strlen(ip);i++)

op[i]=(op[i]-'0')^(ip[i]-'0')+'0';

}
void sbox(char op[],char ip[],int s[][4])

{

int value;

value=s[(ip[0]-'0')*2+(ip[3]-'0')][(ip[1]-'0')*2+(ip[2]-'0')];
```

```
op[0]=value/2+'0';

op[1]=value%2+'0';

op[2]='\0';

}

void fk(char op[],char ip[],char k[])

{

char l[5],r[5],tmp[9],tmp1[9],tmp2[9];

strncpy(l,ip,4);

l[4]='\0';

strncpy(r,ip+4,4);

r[4]='\0';

permute(tmp,r,ep,8);

xor(tmp,k);

sbox(tmp1,tmp,s0);

sbox(tmp2,tmp+4,s1);

strcat(tmp1,tmp2);

permute(tmp,tmp1,p4,4);

xor(tmp,l);

strcat(tmp,r);

strcpy(op,tmp);

}

void sw(char pr[])

{

char tmp[9];

strncpy(tmp,pr+4,4);
strncpy(tmp+4,pr,4);

tmp[8]='\0';

strcpy(pr,tmp);

}

void main()

{

char key[11],k1[9],k2[9],plain[9],cipher[9],tmp[9];

clrscr();

printf("enter 10 bit key:");

gets(key);

if(strlen(key)!=10) printf("invalid key length !!");

else

{

keygen(k1,k2,key);

printf("sub key k1::");

puts(k1);

printf("subkey k2::");

puts(k2);

printf("enter 8 bit plain text:");

gets(plain);

if(strlen(plain)!=8) printf("invalid length plain text !!");

permute(tmp,plain,ip,8);

fk(cipher,tmp,k1);

sw(cipher);

fk(tmp,cipher,k2);
permute(cipher,tmp,ipinv,8);

printf("cipher teaxt is::");

puts(cipher);

/* decryption process*/

permute(tmp,cipher,ip,8);

fk(plain,tmp,k2);

sw(plain);

fk(tmp,plain,k1);

permute(plain,tmp,ipinv,8);

printf("decrypted text is::");

puts(plain);

}

getch();

}
```

Exercise:1. Using AES algorithm decrypt the cipher

2. Implement HMAC algorithm

9. Using RSA algorithm encrypt a text data and Decrypt the same.

a) RSA encryption algorithm

Algorithm:

RSA encrypts messages through the following algorithm, which is divided into 3 steps:

1. Key Generation

I. Choose two distinct prime numbers p and q.

II. Find n such that n = pq.
n will be used as the modulus for both the public and private keys.

III. Find the totient of n, $\phi(n)$

$\phi(n)=(p-1)(q-1)$.

IV. Choose an e such that $1 < e < \phi(n)$, and such that e and $\phi(n)$ share no divisors other than 1 (e and $\phi(n)$ are relatively prime).
e is kept as the public key exponent.

V. Determine d (using modular arithmetic) which satisfies the congruence relation

$de \equiv 1 \pmod{\phi(n)}$.

In other words, pick d such that de - 1 can be evenly divided by (p-1)(q-1), the totient, or $\phi(n)$.
This is often computed using the Extended Euclidean Algorithm, since e and $\phi(n)$ are relatively prime and d is to be the modular multiplicative inverse of e.
d is kept as the private key exponent.

The public key has modulus n and the public (or encryption) exponent e. The private key has modulus n and the private (or decryption) exponent d, which is kept secret.

2. Encryption

I. Person A transmits his/her public key (modulus n and exponent e) to Person B, keeping his/her private key secret.

II. When Person B wishes to send the message "M" to Person A, he first converts M to an integer such that $0 < m < n$ by using agreed upon reversible protocol known as a padding scheme.

III. Person B computes, with Person A's public key information, the ciphertext c corresponding to

$c \equiv m^e \pmod{n}$.

IV. Person B now sends message "M" in ciphertext, or c, to Person A.

3. Decryption

I. Person A recovers m from c by using his/her private key exponent, d, by the computation

$m \equiv c^d \pmod{n}$.

II. Given m, Person A can recover the original message "M" by reversing the padding scheme.

This procedure works since

$c \equiv m^e \pmod{n}$,
$c^d \equiv (m^e)^d \pmod{n}$,
$c^d \equiv m^{de} \pmod{n}$.

By the symmetry property of mods we have that

$m^{de} \equiv m^{de} \pmod{n}$.

Since de = 1 + kϕ(n), we can write

$m^{de} \equiv m^{1 + k\phi(n)} \pmod{n}$,
$m^{de} \equiv m(m^k)^{\phi(n)} \pmod{n}$,
$m^{de} \equiv m \pmod{n}$.

From Euler's Theorem and the Chinese Remainder Theorem, we can show that this is true for all m and the original message

$c^d \equiv m \pmod{n}$, is obtained.

Program:

```
#include<stdio.h>
main()
{
  int k,b,bin[20];
  int i;
  long int c,m,e,d,n;
  char ch;
  char in_file[20],out_file[20];
  FILE *in,*out;
  clrscr();
  printf("\n Enter any input text file name : ");
  gets(in_file);
  printf("\n Enter file name to store enc output : ");
  gets(out_file);
  in = fopen(in_file,"r");
  out = fopen(out_file,"w");

  printf("\n Enter values of e and n : ");
  scanf("%ld%ld",&e,&n);

  i=-1;
  b=e;
  while(b>0)
  {
    bin[++i] = b%2;
    b=b/2;
  }
  k=i;
  do
  {
    m = fgetc(in);
    d = 1;

    for( i=k; i>=0; i--)
    {
     d = (d*d) % n;
     if (bin[i] == 1)
        d = (d*m) % n;
    }

    fputc(d,out);
  }
  while(!feof(in));
  printf("\n File is encrfypted successfully....");
  getch();
}
```

INPUT/OUTPUT:

```
 Enter any input text file name : inp.txt

 Enter file name to store enc output : out.txt

 Enter values of e and n :
7 187

 File is encrfypted successfully....

C:\TURBOC2>type inp.txt
abcdefghijklmnop123&*()

C:\TURBOC2>type out.txt
\§░ÉTwë│`òp0âB¢l↓v◄/☼t.
```

b) RSA Decryption algorithm

```
#include<stdio.h>
main()
{
  int k,b,bin[20];
  int i;
  long int c,m,e,d,n;
  char ch;
  char in_file[20],out_file[20];
  FILE *in,*out;
  clrscr();
  printf("\n Enter any ciphertext file name : ");
  gets(in_file);
  printf("\n Enter file name to store dec output : ");
  gets(out_file);
  in = fopen(in_file,"r");
  out = fopen(out_file,"w");
  printf("\n Enter values of d and n : ");
  scanf("%ld%ld",&e,&n);
  i=-1;
  b=e;
  while(b>0)
  {
     bin[++i] = b%2;
     b=b/2;
  }
  k=i;
  do
  {
     m = fgetc(in);
     d = 1;
     for( i=k; i>=0; i--)
     {
      d = (d*d) % n;
      if (bin[i] == 1)
         d = (d*m) % n;
     }
    fputc(d,out);
  }
  while(!feof(in));
  printf("\n File is decrypted successfully....");
  getch();
 }
```

INPUT/OUTPUT:

```
Enter any ciphertext file name : out.txt

Enter file name to store dec output : inp1.txt

Enter values of d and n : 23 187

File is decrypted successfully....
```

```
C:\TURBOC2>type out.txt
\§░ÉTwë│`òp0âB¢l↓v◄/☼t.

C:\TURBOC2>type inp1.txt
abcdefghijklmnop123&*()U
```

Exercise:

1. **Implement Diffie- Hellman cryptosystem using RSA algorithm.**
2. **Implement SHA-512 algorithm**

OPERATING SYSTEM LAB MANUAL

EXPERIMENT NO.1

CPU SCHEDULINGALGORITHMS

A). FIRST COME FIRST SERVE:

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

DESCRIPTION:

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

ALGORITHM:

Step 1: Start the process
Step 2: Accept the number of processes in the ready Queue
Step 3: For each process in the ready Q, assign the process name and the burst time Step 4: Set the waiting of the first process as ‗0'and its burst time as its turnaround time Step 5: for each process in the Ready Q calculate
a). Waiting time (n) = waiting time (n-1) + Burst time (n-1)
b). Turnaround time (n)= waiting time(n)+Burst time(n)
Step 6: Calculate
a) Average waiting time = Total waiting Time / Number of process

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

Step 7: Stop the process

SOURCE CODE :

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

INPUT

```
Enter the number of processes --          3
Enter Burst Time for Process 0 --         24
Enter Burst Time for Process 1 --         3
Enter Burst Time for Process 2 --         3
```

OUTPUT

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

Average Waiting Time--   17.000000
Average Turnaround Time --                27.000000
```

B). SHORTEST JOB FIRST:

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

DESCRIPTION:

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

ALGORITHM:

Step 1: Start the process

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

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

Step 4: Start the Ready Q according the shortest Burst time by sorting according to lowest to highest burst time.

Step 5: Set the waiting time of the first process as ‗0' and its turnaround time as its burst time.

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

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

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

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

Step 8: Calculate

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

d) Average Turnaround time = Total Turnaround Time / Number of process Step

9: Stop the process

SOURCE CODE :

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

}
for(i=0;i<n;i++)
for(k=i+1;k<n;k++)
if(bt[i]>bt[k])
{
temp=bt[i];
bt[i]=bt[k];
bt[k]=temp;

temp=p[i];
p[i]=p[k];
p[k]=temp;
}
wt[0] = wtavg = 0;
tat[0] = tatavg = bt[0]; for(i=1;i<n;i++)
{
wt[i] = wt[i-1] +bt[i-1];
 tat[i] = tat[i-1] +bt[i];
wtavg = wtavg + wt[i];
tatavg = tatavg + tat[i];
}
printf("\n\t PROCESS \tBURST TIME \t WAITING TIME\t TURNAROUND TIME\n");
for(i=0;i<n;i++)
        printf("\n\t P%d \t\t %d \t\t %d \t\t %d", p[i], bt[i], wt[i], tat[i]); printf("\nAverage
Waiting Time -- %f", wtavg/n);
printf("\nAverage Turnaround Time -- %f", tatavg/n); getch();
        }
```

INPUT

```
Enter the number of processes --          4
Enter Burst Time for Process 0 --         6
Enter Burst Time for Process 1 --         8
Enter Burst Time for Process 2 --         7
Enter Burst Time for Process 3 --         3
```

OUTPUT

```
                                          WAITING        TURNARO
PROCESS              BURST TIME           TIME           UND TIME
P3                   3                    0              3
P0                   6                    3              9
P2                   7                    9              16
P1                   8                    16             24
Average Waiting Time --                   7.000000
Average Turnaround Time --                13.000000
```

C). ROUND ROBIN:

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

DESCRIPTION:

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

ALGORITHM:

Step 1: Start the process

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

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

Step 4: Calculate the no. of time slices for each process where No. of time slice for process (n) = burst time process (n)/time slice

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

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

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

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

Step 7: Calculate

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

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

Step 8: Stop the process

SOURCE CODE

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

INPUT:

```
Enter the no of processes – 3
 Enter Burst Time for process 1 – 24
Enter Burst Time for process 2 -- 3
Enter Burst Time for process 3 – 3
Enter the size of time slice – 3
```

OUTPUT:

```
PROCESS      BURST TIME        WAITING TIME     TURNAROUNDTIME

1                 24                     6             30

2                 3                     4                7

3                3                      7            10

The Average Turnaround time is – 15.666667
The Average Waiting time is --- 5.666667
```

D). PRIORITY:

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

DESCRIPTION:

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

ALGORITHM:

Step 1: Start the process
Step 2: Accept the number of processes in the ready Queue
Step 3: For each process in the ready Q, assign the process id and accept the CPU burst time
Step 4: Sort the ready queue according to the priority number.
Step 5: Set the waiting of the first process as ‗0‘ and its burst time as its turnaround time
Step 6: Arrange the processes based on process
priority
Step 7: For each process in the Ready Q calculate
Step 8: for each process in the Ready Q calculate

a) Waiting time(n)= waiting time (n-1) + Burst time (n-1)
b) Turnaround time (n)= waiting time(n)+Burst time(n)

Step 9: Calculate

c) Average waiting time = Total waiting Time / Number of process
d) Average Turnaround time = Total Turnaround Time / Number of process Print the results in an order.

Step10: Stop

SOURCE CODE:

```
#include<stdio.h>
main()
{
int p[20],bt[20],pri[20], wt[20],tat[20],i, k, n, temp; float wtavg,
tatavg;
clrscr();
printf("Enter the number of processes --- ");
scanf("%d",&n);
for(i=0;i<n;i++)
{
p[i] = i;
printf("Enter the Burst Time & Priority of Process %d --- ",i); scanf("%d
%d",&bt[i], &pri[i]);
}
for(i=0;i<n;i++) for(k=i+1;k<n;k++)
if(pri[i] > pri[k])
{
temp=p[i];
p[i]=p[k];
p[k]=temp;
temp=bt[i];
bt[i]=bt[k];
bt[k]=temp;
temp=pri[i];
pri[i]=pri[k];
pri[k]=temp;
}
wtavg = wt[0] = 0;
tatavg = tat[0] = bt[0];
for(i=1;i<n;i++)
{
wt[i] = wt[i-1] + bt[i-1];
tat[i] = tat[i-1] + bt[i];
wtavg = wtavg + wt[i];
 tatavg = tatavg + tat[i];
}
printf("\nPROCESS\t\tPRIORITY\tBURST TIME\tWAITING TIME\tTURNAROUND
TIME");
for(i=0;i<n;i++)
printf("\n%d \t\t %d \t\t %d \t\t %d \t\t %d ",p[i],pri[i],bt[i],wt[i],tat[i]);
printf("\nAverage Waiting Time is --- %f",wtavg/n);
printf("\nAverage Turnaround Time is --- %f",tatavg/n);
getch();
}
```

INPUT

```
Enter the number of processes -- 5
Enter the Burst Time & Priority of Process 0 --- 10        3
Enter the Burst Time & Priority of Process 1 --- 1         1
Enter the Burst Time & Priority of Process 2 --- 2         4
Enter the Burst Time & Priority of Process 3 --- 1         5
Enter the Burst Time & Priority of Process 4 --- 5         2
```

OUTPUT

```
                                                WAITING        TURNARO
PROCESS         PRIORITY        BURST TIME      TIME           UND TIME
1               1               1               0              1
4               2               5               1              6
0               3               10              6              16
2               4               2               16             18
3               5               1               18             19
Average Waiting Time is ---                     8.200000
Average Turnaround Time is                      --- 12.000000
```

VIVA QUESTIONS

1) Define the following
   a) Turnaround time   b) Waiting time   c) Burst time   d) Arrival time
2) What is meant by process scheduling?
3) What are the various states of process?
4) What is the difference between preemptive and non-preemptive scheduling
5) What is meant by time slice?
6) What is round robin scheduling?

EXPERIMENT NO.2

MEMORY MANAGEMAENT

A). MEMORY MANAGEMENT WITH FIXED PARTITIONING TECHNIQUE (MFT)

AIM: To implement and simulate the MFT algorithm.

DESCRIPTION:

In this the memory is divided in two parts and process is fit into it. The process which is best suited in to it is placed in the particular memory where it suits. We have to check memory partition. If it suits, its status should be changed.

ALGORITHM:

```
Step1: Start the process.
Step2: Declarevariables.
Step3: Enter total memory size ms.
Step4: Allocate memory for os.
Ms=ms-os
Step5: Read the no partition to be divided n Partition size=ms/n.
Step6: Read the process no and process size.
Step 7: If process size is less than partition size allot alse blocke the process.While
allocating update memory wastage-external fragmentation.
if(pn[i]==pn[j])f=1;
if(f==0){ if(ps[i]<=siz)
{
extft=extft+size-ps[i];avail[i]=1;
count++;
}
}
Step 8: Print the results
```

SOURCE CODE :

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

main()
{
int ms, bs, nob, ef,n, mp[10],tif=0;
int i,p=0;
clrscr();
printf("Enter the total memory available (in Bytes) -- ");
scanf("%d",&ms);
printf("Enter the block size (in Bytes) -- ");
 scanf("%d", &bs);
nob=ms/bs;
ef=ms - nob*bs;
printf("\nEnter the number of processes -- ");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("Enter memory required for process %d (in Bytes)-- ",i+1);
scanf("%d",&mp[i]);
}
printf("\nNo. of Blocks available in memory -- %d",nob);
printf("\n\nPROCESS\tMEMORY REQUIRED\t ALLOCATED\tINTERNAL
FRAGMENTATION");
for(i=0;i<n && p<nob;i++)
{
printf("\n %d\t\t%d",i+1,mp[i]);
if(mp[i] > bs)
printf("\t\tNO\t\t---");
else
{
printf("\t\tYES\t%d",bs-mp[i]);
tif = tif + bs-mp[i];
p++;
}
}
if(i<n)
printf("\nMemory is Full, Remaining Processes cannot be accomodated");

printf("\n\nTotal Internal Fragmentation is %d",tif);
printf("\nTotal External Fragmentation is %d",ef);
getch();
}
```

INPUT

```
        Enter the total memory available (in Bytes) --          1000
        Enter the block size (in  Bytes)-- 300
        Enter the number of processes – 5
Enter memory required for process 1 (in Bytes) --      275
Enter memory required for process 2 (in Bytes) --      400
Enter memory required for process 3 (in Bytes) --      290
Enter memory required for process 4 (in Bytes) --      293
Enter memory required for process 5 (in Bytes) --      100

No. of Blocks available in memory --        3
```

OUTPUT

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

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

B) MEMORY VARIABLE PARTIONING TYPE (MVT)

AIM: To write a program to simulate the MVT algorithm

ALGORITHM:

Step1: start the process.

Step2: Declare variables.

Step3: Enter total memory size ms.

Step4: Allocate memory for os.

Ms=ms-os

Step5: Read the no partition to be divided n Partition size=ms/n.

Step6: Read the process no and process size.

Step 7: If process size is less than partition size allot alse blocke the process. While allocating update memory wastage-external fragmentation.

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

Step 8: Print the results

Step 9: Stop the process.

SOURCE CODE:

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

OUTPUT:

```
Enter the total memory available (in Bytes) – 1000
 Enter memory required for process 1 (in Bytes) – 400
 Memory is allocated for Process 1
Do you want to continue(y/n) -- y
Enter memory required for process 2 (in Bytes) -- 275
Memory is allocated for Process 2
Do you want to continue(y/n) – y
 Enter memory required for process 3 (in Bytes) – 550


Memory is Full

 Total Memory Available – 1000

PROCESS          MEMORY ALLOCATED

 1                    400
 2                    275


  Total Memory Allocated is 675 Total External Fragmentation is 325
```

VIVA QUESTIONS

1) What is MFT?
2) What is MVT?
3) What is the difference between MVT and MFT?
4) What is meant by fragmentation?
5) Give the difference between internal and external fragmentation

EXERCISE:

1. **Write a program for simulating Best fit and worst fit**

EXPERIMENT NO.3

PAGE REPLACEMENT ALGORITHMS

A) FIRST IN FIRST OUT:

AIM: To implement FIFO page replacement technique.

DESCRIPTION:

- The FIFO page-replacement algorithm is easy to understand and program. However, its performance is not always good.
- On the one hand, the page replaced may be an initialization module that was used a long time ago and is no longer needed.
- On the other hand, it could contain a heavily used variable that was initialized early and is in constant use.

ALGORITHM:

1. Start the process
2. Read number of pages n
3. Read number of pages no
4. Read page numbers into an array a[i]
5. Initialize avail[i]=0 .to check page hit
6. Replace the page with circular queue, while re-placing check page availability in the frame Place avail[i]=1 if page is placed in theframe Count page faults
7. Print the results.
8. Stop the process.

SOURCE CODE :

```
#include<stdio.h>
#include<conio.h>
int fr[3];
void main()
{
void display();
int i,j,page[12]={2,3,2,1,5,2,4,5,3,2,5,2};
int flag1=0,flag2=0,pf=0,frsize=3,top=0;
clrscr();
for(i=0;i<3;i++)
{
fr[i]=-1;
}
for(j=0;j<12;j++)
{
flag1=0;
flag2=0;
for(i=0;i<12;i++)
{
if(fr[i]==page[j])
{
flag1=1;
flag2=1;
break;
}
}
if(flag1==0)
{
for(i=0;i<frsize;i++)
{
if(fr[i]==-1)
{
fr[i]=page[j];
flag2=1;
break;
}
}
}
if(flag2==0)
{
fr[top]=page[j];
top++;
pf++;
if(top>=frsize)
top=0;
}
display();
}
printf("Number of page faults : %d ",pf+frsize);
 getch();
}
void display()
{
int i;
printf("\n");
for(i=0;i<3;i++)
printf("%d\t",fr[i]);
}
```

OUTPUT:

```
2 -1 -1
2  3 -1
2  3 -1
2  3  1
5  3  1
5  2  1
5  2  4
5  2  4
3  2  4
3  2  4
3  5  4
3  5  2
Number of page faults: 9
```

B) LEAST RECENTLY USED

AIM: To implement LRU page replacement technique.

ALGORITHM:

1. Start the process
2. Declare the size
3. Get the number of pages to be inserted
4. Get the value
5. Declare counter and stack
6. Select the least recently used page by counter value
7. Stack them according the selection.
8. Display the values
9. Stop the process

SOURCE CODE :

```
#include<stdio.h>
#include<conio.h>
int fr[3];
void main()
{
void display();
int p[12]={2,3,2,1,5,2,4,5,3,2,5,2},i,j,fs[3];
int index,k,l,flag1=0,flag2=0,pf=0,frsize=3;
clrscr();
for(i=0;i<3;i++)
{
fr[i]=-1;
}
for(j=0;j<12;j++)
{
flag1=0,flag2=0;
for(i=0;i<3;i++)
{
if(fr[i]==p[j])
{
flag1=1;
flag2=1;
break;
}}
if(flag1==0)
{
for(i=0;i<3;i++)
{
if(fr[i]==-1)
{
fr[i]=p[j];
flag2=1;
break;
}
}
}
if(flag2==0)
{
for(i=0;i<3;i++)
fs[i]=0;
for(k=j-1,l=1;l<=frsize-1;l++,k--)
{
for(i=0;i<3;i++)
{
if(fr[i]==p[k])
fs[i]=1;
}
}
for(i=0;i<3;i++)
{
if(fs[i]==0)
index=i;
}
fr[index]=p[j];
pf++;
}
display();
}
printf("\n no of page faults :%d",pf+frsize);
getch();
}
void display()
{
int i;
printf("\n");
for(i=0;i<3;i++)
printf("\t%d",fr[i]);
}
```

OUTPUT:

```
2 -1 -1
2  3 -1
2  3 -1
2  3  1
2  5  1
2  5  1
2  5  4
2  5  4
3  5  4
3  5  2
3  5  2
3  5  2

no of page faults : 7
```

C) OPTIMAL

AIM: To implement optimal page replacement technique.

ALGORTHIM:

1. Start Program
2. Read Number Of Pages And Frames
3.Read Each Page Value
4. Search For Page In The Frames
5.If Not Available Allocate Free Frame
6. If No Frames Is Free Repalce The Page With The Page That Is LeastlyUsed
7.Print Page Number Of Page Faults
8.Stop process.

SOURCE CODE:

```
/* Program to simulate optimal page replacement */
#include<stdio.h>
#include<conio.h>
int fr[3], n, m;
void display();
void main()
{
 int i,j,page[20],fs[10];
 int max,found=0,lg[3],index,k,l,flag1=0,flag2=0,pf=0;
 float pr;
 clrscr();
 printf("Enter length of the reference string: ");
 scanf("%d",&n);
 printf("Enter the reference string: ");
 for(i=0;i<n;i++)
 scanf("%d",&page[i]);
 printf("Enter no of frames: ");
 scanf("%d",&m);
 for(i=0;i<m;i++)
 fr[i]=-1;
 pf=m;
```

```
for(j=0;j<n;j++)
 {
 flag1=0;
 flag2=0;
 for(i=0;i<m;i++)
 {
 if(fr[i]==page[j])
 {
 flag1=1;
 flag2=1;
 break;
 }
 }
 if(flag1==0)
 {
 for(i=0;i<m;i++)
 {
 if(fr[i]==-1)
 {
fr[i]=page[j];
flag2=1;
break;
 }
 }
 }
 if(flag2==0)
 {
 for(i=0;i<m;i++)
 lg[i]=0;
 for(i=0;i<m;i++)
 {
 for(k=j+1;k<=n;k++)
 {
 if(fr[i]==page[k])
 {
 lg[i]=k-j;
 break;
 }
 }
 }
 found=0;
 for(i=0;i<m;i++)
 {
 if(lg[i]==0)
 {
index=i;
 found = 1;
 break;
 }
 }
if(found==0)
 {
 max=lg[0];
 index=0;
 for(i=0;i<m;i++)
 {
 if(max<lg[i])
{
 max=lg[i];

 index=i;
}
 }
 }
 fr[index]=page[j];
 pf++;
}
display();
}
printf("Number of page faults : %d\n", pf);
pr=(float)pf/n*100;
printf("Page fault rate = %f \n", pr);
getch();
}
void display()
{
 int i;
 for(i=0;i<m;i++)
 printf("%d\t",fr[i]);
 printf("\n");
}
```

OUTPUT:

```
Enter length of the reference string: 12
Enter the reference string: 1 2 3 4 1 2 5 1 2 3 4 5
Enter no of frames: 3
1 -1 -1
1 2 -1
1 2 3
1 2 4
1 2 4
1 2 4
1 2 5
1 2 5
1 2 5
3 2 5
4 2 5
4 2 5
Number of page faults : 7 Page fault rate = 58.333332
```

VIVA QUESTIONS

1) What is meant by page fault?
2) What is meant by paging?
3) What is page hit and page fault rate?
4) List the various page replacement algorithm
5) Which one is the best replacement algorithm?

EXERCISE:

1. **Write a C program to simulate LFU page replacement algorithm**

EXPERIMENT NO. 4

FILE ORGANIZATION TECHNIQUES

A) SINGLE LEVEL DIRECTORY:

AIM: Program to simulate Single level directory file organization technique.

DESCRIPTION:

The directory structure is the organization of files into a hierarchy of folders. In a single-level directory system, all the files are placed in one directory. There is a root directory which has all files. It has a simple architecture and there are no sub directories. Advantage of single level directory system is that it is easy to find a file in the directory.

SOURCE CODE :

```
#include<stdio.h>
struct
{
char dname[10],fname[10][10]; int
fcnt;
}dir;

void main()
{
int i,ch; char f[30];
clrscr();
dir.fcnt = 0;
printf("\nEnter name of directory -- ");
scanf("%s", dir.dname);
while(1)
{
printf("\n\n1. Create File\t2. Delete File\t3. Search File \n
4. Display Files\t5. Exit\nEnter your choice -- ");
scanf("%d",&ch);
switch(ch)
{
case 1: printf("\nEnter the name of the file -- ");
scanf("%s",dir.fname[dir.fcnt]); dir.fcnt++;
break;
case 2: printf("\nEnter the name of the file -- "); scanf("%s",f);
for(i=0;i<dir.fcnt;i++)
{
if(strcmp(f, dir.fname[i])==0)
{
        printf("File %s is deleted ",f); strcpy(dir.fname[i],dir.fname[dir.fcnt-1]); break;
        }
        }
        if(i==dir.fcnt)
        printf("File %s not found",f);

                                else
                                        dir.fcnt--;
                                        break;

                case 3:                 printf("\nEnter the name of the file -- ");
                                        scanf("%s",f);
                                        for(i=0;i<dir.fcnt;i++)
                                        {
                                        if(strcmp(f, dir.fname[i])==0)
                                        {
                                        printf("File %s is found ", f);
                                        break;
                                        }
                                        }
                                        if(i==dir.fcnt)
                                        printf("File %s not found",f);
                                        break;
                case 4:                 if(dir.fcnt==0)
                                        printf("\nDirectory Empty");
                                        else
                                        {
                                        printf("\nThe Files are -- ");
                                        for(i=0;i<dir.fcnt;i++)
                                        printf("\t%s",dir.fname[i]);
                                        }
                                        break;
                default: exit(0);
                }
}
getch();}
```

OUTPUT:

```
Enter name of directory -- CSE
1. Create File 2. Delete File 3. Search File
4. Display Files 5. Exit Enter your choice – 1

Enter the name of the file -- A
1. Create File 2. Delete File 3. Search File
4. Display Files 5. Exit Enter your choice – 1

Enter the name of the file -- B
1. Create File 2. Delete File 3. Search File
4. Display Files 5. Exit Enter your choice – 1

Enter the name of the file -- C
1. Create File 2. Delete File 3. Search File
4. Display Files 5. Exit Enter your choice – 4

The Files are -- A B C
1. Create File 2. Delete File 3. Search File
4. Display Files 5. Exit Enter your choice – 3

Enter the name of the file – ABC
File ABC not found
1. Create File 2. Delete File 3. Search File
4. Display Files 5. Exit Enter your choice – 2
24

Enter the name of the file – B
File B is deleted
1. Create File 2. Delete File 3. Search File
4. Display Files 5. Exit Enter your choice – 5
```

B) TWO LEVEL DIRECTORY

AIM: Program to simulate two level file organization technique

Description:

In the two-level directory system, each user has own user file directory (UFD). The system maintains a master block that has one entry for each user. This master block contains the addresses of the directory of the users. When a user job starts or a user logs in, the system's master file directory (MFD) is searched. When a user refers to a particular file, only his own UFD is searched.

.

SOURCE CODE :

```
#include<stdio.h>
struct
{
        char    dname[10],fname[10][10];
        int fcnt;
}dir[10];

void main()
{
        int i,ch,dcnt,k; char f[30],
        d[30]; clrscr();
        dcnt=0;

        while(1)
        {
                printf("\n\n1. Create Directory\t2. Create File\t3. Delete File");
                printf("\n4. Search File\t\t5. Display\t6. Exit\t Enter your choice --");
                scanf("%d",&ch);
                switch(ch)
                {
                        case 1: printf("\nEnter name of directory -- ");
                                scanf("%s", dir[dcnt].dname);
                                dir[dcnt].fcnt=0;
                                dcnt++;
                                printf("Directory created"); break;
                        case 2: printf("\nEnter name of the directory -- ");
                                scanf("%s",d);
                                for(i=0;i<dcnt;i++)
                                        if(strcmp(d,dir[i].dname)==0)
```

```
                    {
                            printf("Enter   name   of   the   file -- ");
                            scanf("%s",dir[i].fname[dir[i].fcnt]);
                            dir[i].fcnt++;
                            printf("File created"); break;
                    }
            if(i==dcnt)
                    printf("Directory %s not found",d);
            break;
        case 3: printf("\nEnter name of the directory -- ");
            scanf("%s",d);
            for(i=0;i<dcnt;i++)
            for(i=0;i<dcnt;i++)
            {
            if(strcmp(d,dir[i].dname)==0)
            {
                    printf("Enter name of the file -- ");
                    scanf("%s",f); for(k=0;k<dir[i].fcnt;k++)
                    {
                            if(strcmp(f, dir[i].fname[k])==0)
                    {
                    printf("File %s is deleted ",f);
                    dir[i].fcnt--;
                    strcpy(dir[i].fname[k],dir[i].fname[dir[i].fcnt]);
                    goto jmp;
                    }
            }

            printf("File %s not found",f); goto jmp;
        }
        }
        printf("Directory %s not found",d); jmp : break;

    case 4: printf("\nEnter name of the directory -- ");
        scanf("%s",d);
        for(i=0;i<dcnt;i++)
        {
            if(strcmp(d,dir[i].dname)==0)
            {
                    printf("Enter the name of the file -- ");
                    scanf("%s",f); for(k=0;k<dir[i].fcnt;k++)

                    {
                            if(strcmp(f, dir[i].fname[k])==0)
                            {
                            printf("File %s is found ",f); goto jmp1;
```

```
                                        }
                                    }
                                    printf("File %s not found",f); goto jmp1;
                                }
                            }
                            printf("Directory %s not found",d); jmp1: break;
                        case 5: if(dcnt==0)

                                printf("\nNo Directory's ");
                            else

                            {
                                printf("\nDirectory\tFiles");
                                for(i=0;i<dcnt;i++)
                                {
                                    printf("\n%s\t\t",dir[i].dname);
                                    for(k=0;k<dir[i].fcnt;k++)
                                        printf("\t%s",dir[i].fname[k]);
                                }
                            }
                            break;
                        default:exit(0);
                    }

            }
            getch();
        }
```

OUTPUT

```
1. Create Directory 2. Create File 3. Delete File
4. Search File 5. Display 6. Exit

Enter your choice -- 1

Enter name of directory -- DIR1 Directory created

1. Create Directory 2. Create File 3. Delete File
4. Search File 5. Display 6. Exit Enter your choice -- 1

Enter name of directory -- DIR2 Directory created
1. Create Directory 2. Create File 3. Delete File
4. Search File 5. Display 6. Exit Enter your choice -- 2

Enter name of the directory – DIR1

Enter name of the file -- A1

File created
1. Create Directory 2. Create File 3. Delete File
4. Search File 5. Display 6. Exit

Enter your choice -- 2

Enter name of the directory – DIR1

Enter name of the file -- A2
File created
1. Create Directory 2. Create File 3. Delete File
4. Search File 5. Display 6.

Exit Enter your choice – 6
```

VIVA QUESTIONS

1. Define directory?
2. List the different types of directory structures?
3. What is the advantage of hierarchical directory structure?
4. Which of the directory structures is efficient? Why?
5. What is acyclic graph directory?

EXERCISE:

1. Write a C program for hierarchical level directory structure

EXPERIMENT.NO.5
FILE ALLOCATION STRATEGIES

A) SEQUENTIAL:

The most common form of file structure is the sequential file in this type of file, a fixed format is used for records. All records (of the system) have the same length, consisting of the same number of fixed length fields in a particular order because the length and position of each field are known, only the values of fields need to be stored, the field name and length for each field are attributes of the file structure.

ALGORITHM:

Step 1: Start the program.

Step 2: Get the number of files.

Step 3: Get the memory requirement of each file.

Step 4: Allocate the required locations to each in sequential order a).

Randomly select a location from available location s1= random(100);

a) Check whether the required locations are free from the selected location.

```
 if(b[s1].flag==0){
for (j=s1;j<s1+p[i];j++){
if((b[j].flag)==0)count++;
       }
if(count==p[i]) break;
}
```

b) Allocate and set flag=1 to the allocated locations. for(s=s1;s<(s1+p[i]);s++)

```
{
k[i][j]=s; j=j+1; b[s].bno=s;
b[s].flag=1;
}
```

Step 5: Print the results file no, length, Blocks allocated.

Step 6: Stop the program

SOURCE CODE :

```
#include<stdio.h>
main()
{
int f[50],i,st,j,len,c,k;
clrscr();
for(i=0;i<50;i++)
f[i]=0;
X:
printf("\n Enter the starting block & length of file");
scanf("%d%d",&st,&len);
for(j=st;j<(st+len);j++)
if(f[j]==0)
{
f[j]=1;
printf("\n%d->%d",j,f[j]);
}
else
{
printf("Block already allocated");
break;
}
if(j==(st+len))
printf("\n the file is allocated to disk");
printf("\n if u want to enter more files?(y-1/n-0)");
scanf("%d",&c);
if(c==1)
goto X;
else
exit();
getch();
}
```

OUTPUT:

```
Enter the starting block & length of file 4 10
4->1
5->1
6->1
7->1
8->1
9->1
10->1
11->1
12->1
13->1
The file is allocated to disk.
```

B) INDEXED:

AIM: To implement allocation method using chained method

DESCRIPTION:

In the chained method file allocation table contains a field which points to starting block of memory. From it for each bloc a pointer is kept to next successive block. Hence, there is no external fragmentation.

ALGORITHM:

Step 1: Start the program.

Step 2: Get the number of files.

Step 3: Get the memory requirement of each file.

Step 4: Allocate the required locations by selecting a location randomly q= random(100);

a) Check whether the selected location is free .

b) If the location is free allocate and set flag=1 to the allocated locations.

```
q=random(100);
{
if(b[q].flag==0)
b[q].flag=1;
b[q].fno=j;
r[i][j]=q;
```

Step 5: Print the results file no, length ,Blocks allocated.

Step 6: Stop the program

SOURCE CODE :

```
#include<stdio.h>
int f[50],i,k,j,inde[50],n,c,count=0,p;
main()
{
clrscr();
for(i=0;i<50;i++)
f[i]=0;
x: printf("enter index block\t");
scanf("%d",&p);
if(f[p]==0)
{
f[p]=1;
printf("enter no of files on index\t");
scanf("%d",&n);
}
else
{
printf("Block already allocated\n");
goto x;
}
for(i=0;i<n;i++)
scanf("%d",&inde[i]);
for(i=0;i<n;i++)
if(f[inde[i]]==1)
{
printf("Block already allocated");
goto x;
}
for(j=0;j<n;j++)
f[inde[j]]=1;
printf("\n allocated");
printf("\n file indexed");
for(k=0;k<n;k++)
printf("\n %d->%d:%d",p,inde[k],f[inde[k]]);
printf(" Enter 1 to enter more files and 0 to exit\t");
scanf("%d",&c);
if(c==1)
goto x;
else
exit();
getch();
}
```

OUTPUT: enter index block 9

```
Enter no of files on index 3
1 2 3
Allocated
File indexed
9->1:1
9->2;1
9->3:1 enter 1 to enter more files and 0 to exit
```

C) LINKED:

AIM: To implement linked file allocation technique.

DESCRIPTION:

In the chained method file allocation table contains a field which points to starting block of memory. From it for each bloc a pointer is kept to next successive block. Hence, there is no external fragmentation

ALGORTHIM:

Step 1: Start the program.

Step 2: Get the number of files.

Step 3: Get the memory requirement of each file.

Step 4: Allocate the required locations by selecting a location randomly q= random(100);

a) Check whether the selected location is free .

b) If the location is free allocate and set flag=1 to the allocated locations.

While allocating next location address to attach it to previous location

```
for(i=0;i<n;i++)
{
for(j=0;j<s[i];j++)
{
q=random(100); if(b[q].flag==0)
b[q].flag=1;
b[q].fno=j;
r[i][j]=q;
            if(j>0)
            {
}
}
p=r[i][j-1]; b[p].next=q;}
```

Step 5: Print the results file no, length ,Blocks allocated.

Step 6: Stop the program

SOURCE CODE :

```
#include<stdio.h>
main()
{
int f[50],p,i,j,k,a,st,len,n,c;
clrscr();
for(i=0;i<50;i++)
f[i]=0;
printf("Enter how many blocks
that are already allocated");
scanf("%d",&p);
printf("\nEnter the blocks no.s
that are already allocated");
for(i=0;i<p;i++)
{
scanf("%d",&a);
f[a]=1;
}
X:
printf("Enter the starting index
block & length");
scanf("%d%d",&st,&len);
k=len;
for(j=st;j<(k+st);j++)
{
if(f[j]==0)
{
f[j]=1;
printf("\n%d->%d",j,f[j]);
}
else
{
printf("\n %d->file is already
allocated",j);
k++;
}
}
printf("\n If u want to enter one
more file? (yes-1/no-0)");
scanf("%d",&c);
if(c==1)
goto X;
else
exit();
getch( );}
```

OUTPUT:

```
Enter how many blocks that are already allocated 3 Enter the blocks
no.s that are already allocated 4 7  Enter the starting index block &
length 3  7  9
3->1
4->1 file is already allocated
5->1
6->1
7->1 file is already allocated
8->1
9->1file is already allocated
10->1
11->1
12->1
```

VIVA QUESTIONS

1) List the various types of files
2) What are the various file allocation strategies?
3) What is linked allocation?
4) What are the advantages of linked allocation?
5) What are the disadvantages of sequential allocation methods?

EXPERIMENT.NO:6

DEAD LOCK AVOIDANCE

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

DESCRIPTION:

Deadlock is a situation where in two or more competing actions are waiting f or the other to finish, and thus neither ever does. When a new process enters a system, it must declare the maximum number of instances of each resource type it needed. This number may exceed the total number of resources in the system. When the userrequest 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 safestate.

Resource request algorithm

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

1. if Request<=Need I go to step 2. Otherwise raise an error condition.
2. if Request<=Available go to step 3. Otherwise Pi must since the resources are

available.

3. Have the system pretend to have allocated the requested resources to process Pi by modifying the state as follows;

Available=Available-Request I; Allocation I
=Allocation +Request I; Need i=Need i-
Request I;

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

ALGORITHM:

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

SOURCE CODE :

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
int alloc[10][10],max[10][10];
int avail[10],work[10],total[10];
int i,j,k,n,need[10][10];
int m;
int count=0,c=0;
char finish[10];
clrscr();

printf("Enter the no. of processes and resources:");
scanf("%d%d",&n,&m);
for(i=0;i<=n;i++)
finish[i]='n';
printf("Enter the claim matrix:\n");
for(i=0;i<n;i++)
for(j=0;j<m;j++)
scanf("%d",&max[i][j]);
printf("Enter the allocation matrix:\n");
for(i=0;i<n;i++)
for(j=0;j<m;j++)
scanf("%d",&alloc[i][j]);
printf("Resource vector:");
for(i=0;i<m;i++)
scanf("%d",&total[i]);
for(i=0;i<m;i++)
avail[i]=0;
for(i=0;i<n;i++)
for(j=0;j<m;j++)
avail[j]+=alloc[i][j];
for(i=0;i<m;i++)
work[i]=avail[i];
for(j=0;j<m;j++)
work[j]=total[j]-work[j];
for(i=0;i<n;i++)
for(j=0;j<m;j++)
need[i][j]=max[i][j]-alloc[i][j];
A:for(i=0;i<n;i++)
{
c=0;
for(j=0;j<m;j++)
if((need[i][j]<=work[j])&&(finish[i]=='n'))
```

```
 c++;
if(c==m)
{
printf("All the resources can be allocated to Process %d",
 i+1);
printf("\n\nAvailable resources are:");
for(k=0;k<m;k++)
{
work[k]+=alloc[i][k];
printf("%4d",work[k]);
}
printf("\n");
finish[i]='y';
printf("\nProcess %d executed?:%c \n",i+1,finish[i]);
count++;
}
}
if(count!=n)
goto A;
else
printf("\n System is in safe mode");
printf("\n The given state is safe state");
getch();
}
```

OUTPUT

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

VIVA QUESTIONS

1) What is meant by deadlock?
2) What is safe state in banker's algorithms?
3) What is banker's algorithm?
4) What are the necessary conditions where deadlock occurs?
5) What are the principles and goals of protection?

EXERCISE:

1. **Write a C program to simulate deadlock prevention**