Department of Information Technology

Computer Networks ,Operating Systems and Linux programming Lab Manual

(III B.tech -I Sem)

K.Roshan, Assoc.Prof
T.Jakob Sanjay Kumar, Assoc.Prof
S.Sri Ram Murthy, Asst.Prof

J.B.Institute of Engineering & Technology
Yenkapally, Moinabad(Mandal)
Himathnagar(Post),Hydreabad

INDEX

Lab:Part-A

Computer Networks

S.NO	Program Name
1	Implement the data link layer framing methods such as character, character stuffing and bit stuffing
2	Implement on a data set of characters the polynomials-CRC 12,CRC 16 and CRCCCIP
3	Implement Dijkstra's algorithm to compute the shortest path through a graph
4	Take an example subnet graph with weights indicating delay between nodes. Now obtain Routing table art each node using distance vector routing algorithm
5	Take an example subnet of hosts. Obtain broadcast tree for it
6	Take a 64 bit playing text and encrypt the same using DES algorithm
7	Write a program to break the DES coding
8	Using RSA algorithm Encrypt a text data and Decrypt the same

Lab: Part-B

Operating Systems

1	Simulate the following CPU scheduling algorithms a)Round Robin b)SJF c)FCFS d)Prority
2	Simulate all File Organization Techniques a)Single level directory b)Two level c)Hierarchical d)DAG
3	Simulate Banker's Algorithm for Dead Lock Avoidance
4	Simulate Banker's Algorithm for Dead Lock Prevention
5	Simulate all file allocating strategies a)Sequential b)Indexed c)Linked
6	Simulate MVT and MFT

<u>Lab Part-C</u>

Linux Programming

1	Write a shell script that accepts a file name, starting and ending line numbers as arguments and displays all the lines between the given line numbers
2	Write a shell script that deletes all lines containing a specific word in one or more arguments to it
3	Write a shell script that displays a list of all files in the current directory to which the user has read, write and execute permissions
4	Write a shell script that receives any number of file names as arguments checks if every argument supplied is a file or a directory and reports accordingly. Whenever the argument is a file, the number of lines on it is also reported
5	Write a shell script accepts a list of file names as its arguments, counts and reports the occurrence of each word that is present in the first argument file on other argument files
6	Write a shell script to list all of the directory files in a directory
7	Write a shell script to find factorial of a given integer
8	Write an awk script to count the number of lines in a file that do not contain vowels
9	Write an awk script to count the number of charcters,words and lines in a file
10	Write a C program that makes a copy a file using standard I/O and system calls

PART – A [CN]

1. Implement the data link layer framing methods such as character stuffing.

```
#include<stdio.h>
#include<string.h>
#include<conio.h>
void main()
{
        int j,l,m,c,k;
        char a[50],b[50];
        printf("Enter the
        string:"); scanf("%s",a);
        strcpy(b,"DLESTX");
        m=strlen(a);
        for(j=0;j<m;)
        {
                if(a[j]=='d')
                {
                        if(a[j+1]=='l')
                        {
                                if(a[j+2]=='e')
                                {
                                        c=j+2;
                                        for(l=0;l<3;l++)
                                        {
                                                for(k=m;k>c;k--)
                                                {
                                                        a[k]=a[k-1];
                                                }
                                                m++;
                                                a[m]='\0';
                                                c+=1;
                                        }
                                        a[j+3]='d';
                                        a[j+4]='l';
                                        a[j+5]='e';
                                        a[m]='\0';
                                        j+=5;
                                }
                        }
                }
                j++;
        }
        strcat(b,a);
        strcat(b,"DLEETX");
        printf("\n%s",b);
        printf("\nReceiver
        side:"); m=strlen(a);
        for(j=0;j<m;)
        {
                if(a[j]=='d'){
                        if(a[j+1]=='l')
{
                                        if(a[j+2]=='e')
                                        {
                                                c=j;
                                                for(l=0;l<3;l++)
                                                {
                                                        for(k=c;k<m;k++)
                                                                a[k]=a[k+1];
                                                }
                                                c++;
                                        }
                                        j=c;
                                }
                        }
                        j++;
                }
                printf("\n%s",a);
                getch();
}
```

2. Implement the data link layer framing methods such as character count and bit stuffing.

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<stdlib.h>
void main()
{
        int i,m,n,j,x;
        char str1[50]="",str2[50]="",temp[40];
        clrscr();
        printf("Enter the
        string:");
        scanf("%s",str1);
        n=strlen(str1); while(n>0)
        {
                j=0;
                x=0;
                m=3+random(n);
                if(m>n)
                {
                        m=n;
                        printf("%d%s",m+1,str1);
                        break;
                }
                else
                {
                        printf("%d",m+1);
                        strcpy(str2,str1);
                        for(i=0;i<m;i++)
                          {
                                temp[x++]=str2[i];
                                temp[x]='\0';
                        }
                        strcpy(str2,temp);
                        printf("%s",str2);
                        for(i=m;i<n;i++)
                                temp[j++]=str1[i];
                }
                temp[j]='\0';
                strcpy(str1,temp);
                n=strlen(str1);
        }
        getch();
}
```

Computer Networks & Operating Systems Lab Manual

3. **Implement on a data set of characters the three CRC polynomials – CRC 12, CRC 16 and CRC CCIP.**

```
#include<stdio.h>
#include<conio.h>
int frame[50],msg[50],key[20]={0},temp[20];
int n,fl,k,ch,i,j;
void main()
{
        void read();
        void   crc();
        clrscr();
        printf("\n\t\tCRC\n");
        read();
        crc();
        getch();
}
void read()
{
        printf("\nEnter the frame
        length:"); scanf("%d",&fl);
        printf("\nEnter the
        frame:"); for(i=0;i<fl;i++)
        scanf("%d",&frame[i]);
        printf("\n1:CRC-10\t2:CRC-16\t3:CRC-
        CCITT\n"); scanf("%d",&ch);
        if(ch==1)
        {
                key[0]=key[1]=key[9]=key[10]=key[11]=key[12]=1;
                k=13;
        }
        else if(ch==2)
        {
                key[0]=key[1]=key[14]=key[16]=1;
                k=17;
        }
        else if(ch==3)
        {
                key[0]=key[1]=key[11]=key[16]=1;
                k=17;
        }
        else
                printf("\nInvalid
        Choice"); n=fl+k-1;
        printf("\nPolynomial
        is:"); for(i=0;i<k;i++)
    printf("%d",key[i]);
}
void crc()
{
        int i,g;
        void xor(int);
        for(g=1;g<k;g++)
                temp[g]=msg[g];
        for(i=0;i<fl;i++)
        {
```

```
                if(temp[0]==0)
                        xor(0);
                else
                {
                        if(temp[0]==1)
                                xor(1);
                        if(i!=fl) temp[k-
                                1]=msg[g++];
                }
        }
        printf("\nCheck Sum=\n");
        for(i=fl,j=0;i<n;i++,j++)
                msg[i]=temp[j];
        for(i=0;i<k;i++)
                printf("%d",msg[i]);
}
void xor(int x)
{
        for(i=1;i<k;i++) temp[i-
        1]=(x==0)?temp[i]^0:temp[i]^key[i];
}
```

4. Implement Dijkstra‘s algorithm to compute the Shortest path thru a graph.

```
#include<stdio.h>
#include<limits.h>
#define MAXNODE
10 #define PERM 1
#define TENT 2
#define infinity INT_MAX
typedef struct NODELABEL
{
        int predecessor;
        int length,label;
}NODELABEL;
int shortpath(a,n,s,t,path,dist)
int a[MAXNODE][MAXNODE],n,s,t,path[MAXNODE],*dist;
{
        NODELABEL state[MAXNODE];
```

```
            int i,k,min,count;
            int rpath[MAXNODE];
            *dist=0;
            for(i=0;i<=n;i++)
            {
                        state[i].predecessor=0;
        state[i].length=0;
                        state[i].label=TENT;
            }
            state[s].predecessor=0;
            state[s].length=0;
            state[s].label=PERM;
            k=s;
            do
            {
                        for(i=1;i<=n;i++)
                        {
                                    if(a[k][i]>0&&state[i].label==TENT)
                                    {
                                                if(state[k].length+a[k][i]<state[i].length)
                                                {
                                                            state[i].predecessor=k;
                                                            state[i].length=state[i].length+a[k][i];
                                                }
                                    }
                        }
min=infinity;
            k=0;
                        for(i=1;i<=n;i++)
                        {
                                    if(state[i].label==TENT&&state[i].length<min)
                                    {
                                                min=state[i].length;
                                                k=i;
                                    }
                        }
                        if(k==0) return(0);

                                    state[k].label=PERM;
```

```
	}while(k!=t);
	k=t;
	count=0;
	do
	{
		count=count+1;
		rpath[count]=k;
		k=state[k].predecessor;
	}while(k!=0);
	for(i=0;i<=count;i++)
		path[i]=rpath[count-
	i+1]; for(i=0;i<=count;i++)
		*dist+=a[path[i]][path[i+1]];
	return(count);
}
void main()
{
	int a[MAXNODE][MAXNODE],i,j,path[MAXNODE],from,to,dist,count,n;
	printf("\nEnter how many nodes:");
	scanf("%d",&n);
	printf("%d",n);
	for(i=1;i<=n;i++)
	{
		printf("\nEnter how many
		nodes:"); scanf("%d",&n);
		printf("%d",n);
		for(i=1;i<=n;i++)
		{
			printf("\nEnter node%d
			connectivity:",i); for(j=1;j<=n;j++)
			{
				scanf("%d",&a[i][j]);
				printf("%d\n",a[i][j]);
			}
		}
		printf("\nfrom to where:");
		scanf("%d%d",&from,&to);
		printf("%d%d",from,to);
		count=shortpath(a,n,from,to,path,&dist);
		if(dist)
		{
			printf("\nShortest path");
			printf("%d",path[1]);
			for(i=2;i<=count;i++)
				printf("->%d",path[i]);
			printf("\nMaximum distance=%d\n",dist);
		}
		else
			printf("\nPath does not exist\n");
	}
}
```

5. Take an example subnet graph with weights indicating delay between nodes. Now obtain Routing table art each node using distance vector routing algorithm.

```
#include<stdio.h>
#include<conio.h>
   struct rr
   {
   int
   bt,num; };
   struct rr a[20],temp;
   int i,ts,n;
   void main()
   {
      void read();
      void process();
      read();
      process();
   }
   void read()
   {
      printf("Enter number of
      jobs:"); scanf("%d",&n);
      printf("\nEnter the time
      sequence:"); scanf("%d",&ts);
      printf("\nEnter burst
      time:"); for(i=1;i<=n;i++)
      {
            printf("Job(%d)",i);
            scanf("%d",a[i].bt);
            a[i].num=i;
      }
   }
   void process()
   {
      int temp=0;
   x:  for(i=1;i<=n;i++)
      {
            for(i=1;a[i].bt<=ts;i++)
            {
                  do
                  {
                        a[i].bt=a[i].bt-ts;
                        printf("Job(%d)",i);
         printf("%d",temp);
                        temp=temp+ts;
                        goto x;
                  }while(a[i].bt<=0);
            }
      }
   }
```

6. Take a 64 bit playing text and encrypt the same using DES algorithm.

```
/*DES CODE*/
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<string.h>
void main()
{
        int i,ch,lp;
        char cipher[50],plain[50];
        char key[50];
        clrscr();
        while(1)
        {
                printf("\n-----MENU-----");
                printf("\n1:Data Encryption\t2:Data
                Decryption\t3:Exit"); printf("\nEnter your choice:");
                scanf("%d",&ch);
                switch(ch)
                {
                        case 1:
                                        printf("\nData Encryption");
                                        printf("\nEnter the plain
                                        text:"); fflush(stdin);
                                        gets(plain);
                                        printf("\nEnter the encryption
                                        key:"); gets(key);
                                        lp=strlen(key);
                                        for(i=0;plain[i]!='\0';i++)
                                                cipher[i]=plain[i]^lp;
                                        cipher[i]='\0';
                                        printf("\nThe encrypted text
                                        is:"); puts(cipher);
                                        break;
                        case 2:
                                            printf("\nData decryption");
                                              for(i=0;cipher[i]!='\0';i++)
                                                   plain[i]=cipher[i]^lp;
                                          printf("\nDecrypted text is:");
                                        puts(plain);
                                        break;
                        case 3:
                                        exit(0);
                }
        }
   getch();
}
```

7. **Write a program to break the above DES coding.**

```
/*Breaking DES Code*/
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<ctype.h> void
main()
{
        char pwd[20];
        char alpha[26]="abcdefghijklmnopqrstuvwxyz"; int
        num[20],i,n,key;
        clrscr();
        printf("\nEnter the
        password:");
        scanf("%s",&pwd);
        n=strlen(pwd); for(i=0;i<n;i++)
                num[i]=toascii(tolower(pwd[i]))-
        'a'; printf("\nEnter the key:");
        scanf("%d",&key);
        for(i=0;i<n;i++)
                num[i]=(num[i]+key)%26;
        for(i=0;i<n;i++)
                pwd[i]=alpha[num[i]];
        printf("\nThe key is:%d",key);
        printf("\nEncrypted text
        is:%s",pwd); for(i=0;i<n;i++)
        {
                num[i]=(num[i]-
                key)%26; if(num[i]<0)
                        num[i]=26+num[i];
                pwd[i]=alpha[num[i]];
        }
        printf("\nDecrypted text
        is:%s",pwd); getch();}
```

8. Using RSA algorithm Encrypt a text data and Decrypt the same.

```
/*RSA  Algorithm*/
#include<stdio.h>
#include<conio.h>
#include<ctype.h>
#include<math.h>
#include<string.h>
void main()
{
    int a,b,i,j,t,x,n,k=0,flag=0,prime[100];
    char m[20],pp[20];
    float p[20],c[20];
    double e,d;
    clrscr();
    for(i=0;i<50;i++)
    {
            flag=0;
            for(j=2;j<i/2;j++)
                    if(i%j==0)
                    {
                            flag=1;
                            break;
                    }
                    if(flag==0)
                            prime[k++]=i;
    }
    a=prime[k-1];
    b=prime[k-2];
    n=a*b; t=(a-1)*(b-
    1);
    e=(double)prime[2];
    d=1/(float)e;
    printf("\nKey of encryption
    is:%lf",d); printf("\nEnter the
    text:"); scanf("%s",&m);
    x=strlen(m); printf("\nSource------------destination");
    printf("\nChar\tnumeric\tcipher\t\tnumeric\t\tchar\n");
    for(i=0;i<x;i++)
    {
            printf("%c",m[i]);
            printf("\t%d",m[i]-97);
            c[i]=pow(m[i]-97,(float)e);
            c[i]=fmod(c[i],(float)n);
            printf("\t%f",c[i]);
            p[i]=pow(c[i],(float)d);
            p[i]=fmod(p[i],(float)n);
            printf("\t%f",p[i]);
            pp[i]=p[i]+97;
            printf("\t%c\n",pp[i]);
    }
    getch();
}
```

```
/*Output:
                    key of encryption is:0.500000
                    Enter the text: sacet
char        numeric             cipher          numeric             char
S                   18          324.000000 18.000000            S
A                   0               0.000000        0.000000            A
C                   2               4.000000        2.000000            C
E                   4               16.000000       4.000000            E
T                   19              361.000000      19.000000           T*/
```

PART – B [OS]

1. [Aim:] Simulate the following CPU scheduling algorithms

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

Theory:

Scheduling is a fundamental operating system function.

CPU scheduling is the basis of multi programming operating system. CPU scheduling algorithm determines how the CPU will be allocated to the process. These are of two types.

1. **Primitive scheduling algorithms**
2. **Non-Primitive scheduling algorithms**

1) **Primitive Scheduling algorithms: In this, the CPU can release the process even in the middle of execution. For example: the cpu executes the process p1, in the middle of execution the cpu received a request signal from process p2, then the OS compares the priorities of p1&p2. If the priority p1 is higher than the p2 then the cpu continue the execution of process p1. Otherwise the cpu preempt the process p1 and assigned to process p2.**
2) **Non-Primitive Scheduling algorithm: In this, once the cpu assigned to a process the processor do not release until the completion of that process. The cou will assign to some other job only after the previous job has finished.**

Scheduling methodology:

Though put: It means how many jobs are completed by the CPU with in a time period.

Turn around time: The time interval between the submission of the process and the time of the completion is the turn around time.

Turn around time=Finished time – arrival time

Waiting time: it is the sum of the periods spent waiting by a process in the ready queue

Waiting time=Starting time- arrival time

Response time: it is the time duration between the submission and first response

Response time=First response-arrival time

CPU Utilization: This is the percentage of time that the processor is busy. CPU utilization may range from 0 to 100%

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.

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.
Priority Scheduling: These are of two types.
One is internal priority, second is external priority. The cpu is allocated to the process with the highest priority. Equal priority processes are scheduled in the FCFS order. Priorities are generally some fixed range of numbers such as 0 to 409. The low numbers represent high priority

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.

SOURCE CODE:

```
#include<stdio.h>
#include<conio.h>
struct fcfs
{
        int num,at,bt,wt;
};
struct fcfs
a[20],temp; int i,n,j,t;
float sum1=0,sum2=0;
void main()
{
        void read();
        void sort();
        void
        process();
        void print();
        clrscr();
        read(); sort();
        process();
        print();
        getch();
}
void read()
{
        printf("JAGAN ENTER NO. OF PROCESS DO YOU
        WANT............"); scanf("%d",&n);
        printf("\n\tENTER ARRIVAL TIME & BUST
        TIME"); for(i=1;i<=n;i++)
        {
                a[i].num=i;
                printf("\nJOB(%d)",i);
                scanf("%d%d",&a[i].at,&a[i].bt);
        }
}
void sort()
{
        for(i=1;i<=n-1;i++)
        {
                for(j=i+1;j<=n;j++)
                {
                        if(a[i].at>a[j].at)
                        {
```

```
                        temp=a[i];
                        a[i]=a[j];
                        a[j]=temp;
                    }
                }
        }
        printf("\n\tARRAVIAL TIME & BURST
        TIME"); for(i=1;i<=n;i++)
        {
                printf("\njobs[%d]",a[i].num);
                printf("\t%d\t%d",a[i].at,a[i].bt);
        }
}
void process()
{
        for(i=1;i<=(n*10);i++)
        {
                printf("-");
        }
        for(i=1;i<=n;i++)
        {
                printf("\nJOB %d\t\n",a[i].num);
        }
        for(i=1;i<=(n*10);i++)
        {
                printf("-");
        }
        for(i=1
        ;i<=n;i++)
        {
                a[i].wt=t-a[i].at;
                printf("\t%d\t",t);
                t=t+a[i].bt;
        }
        printf("%d\n\t",t);
}
void print()
{
}
```

OUTPUT:

```
enter no of
jobs 5
enter arraival time burst
time job(1)0
5
job(2)0
24
job(3)0
16
job(4)0
10
job(5)0
3
jobs arrival_time burst_time

job[1]        0     5

job[2]        0     24

job[3]        0     16

job[4]        0     10

job[5]        0     3
the gantt is

----------------------------------------
```

```
Job 1   job 2  job 3   job 4   job 5
---------------------------------------
0       5      29      45     55     58

Jobs waiting time t.a.t
------------ -------- -------
Job[1]     0     5
Job[2]     5     29
Job[3]     29    45
Job[4]     45    55
Job[5]     55    58avg waiting time=26.800000
Avg t.a.t=38.400000
```

(a) SOURCE CODE:

```
#include<stdio.h>
#include<conio.h>
struct sjf
{
        int num,bt,wt;
};
struct sjf
a[20],temp; int i,j,n;
float sum1=0,sum2=0;
void main()
{
        void read();
        void sort();
        void
        process();
        void print();
        clrscr();
        read(); sort();
        process();
        print();
}
void read()
{
        printf("enter no of
        jobs\n"); scanf("%d",&n);
        printf("enter arrival time burst
        time\n"); for(i=1;i<=n;i++)
        {
                printf("job(%d)",i);
                scanf("%d",&a[i].bt);
                a[i].num=i;
        }
}
void sort()
{
        for(i=1;i<=n-1;i++)
        {
                for(j=i+1;j<=n;j++)
                {
                        if(a[i].bt>a[j].bt)
                        {
                                temp=a[i];
                                a[i]=a[j];
                                a[j]=temp;
                        }
                }
        }
        printf("jobs burst_time\n");
        for(i=1;i<=n;i++)
        {
                printf("\njob[%d]\t",a[i].num);
                printf("\t%d\n",a[i].bt);
        }
}
void process()
{
        int t=0;
        printf("the gnatt is\n");
         for(i=1;i<(n*10);i++)
               printf("-");
        printf("\n");
        for(i=1;i<=n;i++)
                printf("job
        %d\t",a[i].num); printf("\n");
        for(i=1;i<=(n*10);i++)
                printf("-");
        printf("\n");
        for(i=1;i<=n;i++)
        {
                a[i].wt=t;
                printf("%d\t",t);
                t=t+a[i].bt;
        }
        printf("%d\n",t);
}
void print()
{
        int i;
        printf("jobs waiting_time
        t.a.t\n"); printf("---- ------- ---
        -"); for(i=1;i<=n;i++)
        {
                printf("\n job[%d]\t%d\t%d",a[i].num,a[i].wt,a[i].wt+a[i].bt);
                sum1+=a[i].wt;
                sum2+=a[i].wt+a[i].bt;
        }
        printf("avg wt=%f\n",sum1/n);
        printf("avg t.a.t=%f\n",sum2/n);
}
```

OUTPUT:

```
enter no of jobs
5
enter arrival time burst time
job(1)5
job(2)24
job(3)16
job(4)10
job(5)3
jobs  burst_time

job[5]       3

job[1]       5

job[4]       10

job[3]       16

job[2]       24
the gnatt is
----------------------------------------------
job 5  job 1  job 4  job 3  job 2
-----------------------------------------------
0      3     8     18     34     58
jobs waiting_time t.a.t
------- -------- --------
Job[5] 0      3
Job[1] 3      8
Job[4] 8      18
Job[3] 18     34
Job[2] 34     58avg wt=12.600000
Avg t.a.t=24.200000
```

(b) SOURCE CODE:

```
#include<graphics.h>
#include<stdio.h>
void main()
{
        int  gd=DETECT,gm,i,k,n,ts,t=0,r=0,pos=1,x1=10,y1=450,no;
        int  pt[10],need[10],turn[10],wt[10],resp[10],tot=10,tp[50];
        float avg;
        char found,str[30],ch,jname[50][3];
        //graphics initialize mathod
        initgraph(&gd,&gm,"..//bgi");
        //setbkcolor(BLUE);
        puts("Enter number of jobs do u
        have"); scanf("%d",&n);
        //reading cpu burst
        times for(i=0;i<n;i++)
        {
                printf("Enter the cpu bursttime of process %d
                ",i+1); scanf("%d",&pt[i]);
                need[i]=pt[i];
        }

        //reading time quantum
        puts("enter the time
        Quantum"); scanf("%d",&ts);
        for(i=0;i<n;i++)
          if(pt[i]>pt[pos])
                pos=i;
            tp[0]=0,k=1;
          do
          {
                for(i=0;i<n;i++)
                {
                        //if need time is greater then time
                        quantm if(need[i]>=ts)
                        {
                                t=t+ts;
                                tp[k]=t;
                                k++;
                                turn[i]=t;
                                need[i]-=ts;
                                str[0]='p';
                                str[1]=(i+1)+48;
                                str[2]='\0';
                                strcpy(jname[k],str);
                        }
                        //if time quantum is less than time
                        quantum else if(need[i]>0)
                        {
                                t=t+need[i];
                                turn[i]=t;
                                need[i]=0;
```

```
                                tp[k]=t;
                                k++;
                                str[0]='p';
                                str[1]=(i+1)+48;
                                str[2]='\0';
                                strcpy(jname[k],str);
                        }
                }
          }while(need[pos]>0);
          //finding response time
          of job resp[0]=0;
          for(i=1;i<n;i++)
          {
                if(pt[i-1]<ts)
                {
                        r=r+pt[i-1];
                        resp[i]=r;
                }
                else
                {
                        r=r+ts;
                        resp[i]=r;
                }
          }
          //printing frames each time we
          enter a no settextstyle(2,0,4);
          for(i=0;i<k;i++)
          {
                if(i<k-1)
                {
                        rectangle(x1,y1-70,x1+50,y1-20);
                        outtextxy(x1+15,y1-55,jname[i+2]);
                }
                no=tp[i];
                itoa(no,str,10);
                outtextxy(x1,y1,str);
                x1+=50;
                //delay(2000);
        }
        printf("\n\n********************************************************");
        printf("\njob no turn around time waiting time response time");
        printf("\n********************************************************\n");
        for(i=0;i<n;i++)
        printf("%5d%13d%15d%15d\n",i+1,turn[i],turn[i]-
        pt[i],resp[i]); //finding avg turnaround time,waiting
        time,response time for(i=0,tot=0;i<n;i++)
          tot=tot+turn[i];
          avg=(float)tot/n;
          printf("\nThe avg turn around
          time=%2.3f",avg); for(i=0,tot=0;i<n;i++)
          tot=tot+(turn[i]-pt[i]);
          avg=(float)tot/n;
          printf("\nThe avg waiting
          time=2.3f",avg); for(i=0,tot=0;i<n;i++)
          tot=tot+resp[i];
          avg=(float)tot/n;
          printf("\n The avg response time
          =%2.3f",avg); getch(); }
```

OUTPUT:

```
Enter no.of jobs do u
have 3
Enter the CPU burst time of process 1 30
Enter the CPU burst time of process 2 6
```

```
Enter the CPU burst time of process 2 8

Enter the quantum    5
------------------------------------------------------------------------------
Job no.              Turn around time     waiting time  Response time
------------------------------------------------------------------------------
1          44                    14          0
2          21                    15          5
3          24                    16          10
The avg turn around time=29.667
The avg waiting time=2.3f
The avg response time=5.00000
```

(c) SOURCE CODE:

```
#include<stdio.h>
#include<conio.h>
struct priority
 {
   int
 num,at,bt,wt,pt; };
 struct priority
 a[20],temp; int i,j,n;
 float sum1=0,sum2=0;
```

```
 void main()
 {
       void read();
       void sort();
       void
       process();
       void print();
       clrscr();
       read(); sort();
       process();
       print();
 }
 void read()
 {
           printf("enter no.of
           jobs\n"); scanf("%d",&n);
           printf("enter priority burst
           time\n"); for(i=1;i<=n;i++)
           {
                 printf("job(%d)",i);
                 scanf("%d%d",&a[i].pt,&a[i].bt);
                 a[i].num=i;
           }
 }
 void sort()
 {
   for(i=1;i<=n-1;i++)
   {
      for(j=i+1;j<=n;j++)
      {
            if(a[i].pt>a[j].pt)
            {
                temp=a[i];
                a[i]=a[j];
                a[j]=temp;
            }
      }
  }
  printf("jobs priority
  burst_time\n"); for(i=1;i<=n;i++)
  {
      printf("\njob[%d]\t",a[i].num);
      printf("\t%d\t%d\n",a[i].pt,a[i].bt);
  }
}
void process()
{
      int t=0;
      printf("the gantt is\n");
      for(i=1;i<(n*10);i++)
      printf("-"); printf("\n");
      for(i=1;i<=n;i++)

      printf("\job%d\t",a[i].num);
      printf("\n");
      for(i=1;i<(n*10);i++)
      printf("-");
      printf("\n");
      for(i=1;i<=n;i++)
      {
           a[i].wt=t;
           printf("%d\t",t);
           t=t+a[i].bt;
```

```
    }
    printf("%d\n",t);
}
void print()
{
  int i;
  printf("jobs waiting time t.a.t\n");
  printf("_____ _______ _____");
  for(i=1;i<=n;i++)
  {
        printf("\njob[%d]\t%d\t%d",a[i].num,a[i].wt,a[i].wt+a[i].bt);
        sum1+=a[i].wt;
        sum2+=a[i].wt+a[i].bt;
  }
  printf("avg w.t=%f\n",(float)sum1/n);
  printf("avg t.a.t =%f\n",(float)sum2/n);
}
```

OUTPUT:

```
enter no.of
jobs 5
enter priority burst
time job(1)2
6
job(2)4
12
job(3)5
1
job(4)1
3
job(5)3
4
jobs priority burst_time

job[4]      1    3

job[1]      2    6

job[5]      3    4

job[2]      4    12
```

```
job[3]      5      1
the gantt is

-----------------------------------------------
job 4  job 1   job 5   job 2   job 3
------------------------------------------------
0      3     9      13     25     26
jobs waiting_time t.a.t
------- -------- --------
Job[4] 0      3
Job[1] 3      9
Job[5] 9      13
Job[2] 13     25
Job[3] 25     26avg wt=10.000000
Avg t.a.t=15.200000
```

2. [Aim:] Simulate all file allocation strategies

a) Sequential

b) Indexed

c) Linked

Theory:

Files are normally stored on the disks. So the main problem is how to allocate space to those files. So that disk space is utilized effectively and files can be accessed quickly. Three major strategies of allocating disc space are in wide use. Sequential, indexed and linked.

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.

b) 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.

c) 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.

(a) File name: SEQUENTIAL.C Source code:

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

```
#include<process.h
> struct sequence
{
        char
        n[20]; int
        i;

}s[20];
int create(int);
int del(int);
void
display(int);
void main()
{
        int
        x=0,j=0;
        clrscr();
        while(1)
        {
                printf("1.creation\n2.delete\n3.display\n4.exit"
                ); printf("\nenter one option");
                scanf("%d",&x
                ); switch(x)
                {
                        case 1:
                                j=create(j
                                ); break;
                        case 2: j=del(j);
```

```
                                break;
                        case 3: display(j);
                                break;
                        case 4: exit(1);
                        default : printf("wrong option");
                }
        }
}
int create(int j)
{
        int m,v;
        j++;
        w:printf("\nenter the file
        name:"); scanf("%s",&s[j].n);
        m=1;
        while(m<j)
        {
                v=strcmp(s[j].n,s[m].n);
                if(v==0)
                {
                        printf("file is already exist\nplease enter another
                        name"); goto w;
                }
                m++;
        }
        printf("\nenter field:");
        scanf("%d",&s[j].i);
        return(j);
}
int del(int j)
{
        j--;
```

```
          return(j);
}
void display(int j)
{
          int l;
          printf("filename\tfield");
          for(l=1;l<=j;l++)
                    printf("\n%s\t\t%d\n",s[l].n,s[l].i);
}
```

Output:

```
1.creation
2.delete
3.display
4.exit
enter one option1

enter the file name:1.c

enter field:1
1.creation
2.delete
3.display
4.exit
enter one option1

enter the file name:2.c
```

```
enter field:2
1.creation
2.delete
3.display
4.exit
enter one option3
filename      field
1.c              1

2.c              2
1.creation
2.delete
3.display
4.exit
enter one option4
```

(b) File name:
INDEXED.C Source code:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
struct file
{
        char n[20];
        int fld,ind;
}s[20];
int no,i=-1,a,b,f,j=-
1,fe,t; char tem[20];
void create();
void display();
void del();
void main()
{
        clrscr();
        while(1)
        {
                printf("\n\nmenu");
                printf("\n1.create\n2.display\n3.delete\n4.exit");
                printf("enter ur choice:");
```

```
                scanf("%d",&no);
                switch(no)
                {
                        case 1: create();
                                break;
                        case 2: display();
                                break;
                        case 3: del();
                                break;
                        case 4: exit(0);
                        default: printf("wrong choice");
                }
        }
}
void create()
{
        i++;
        printf("\nenter the name of the
        recoed:"); scanf("%s",&s[i].n);
        printf("\nenter the index
        no:"); scanf("%d",&s[i].ind);
        printf("\nenter the field no:");
        scanf("%d",&s[i].fld);
        j++;
}
void display()
```

```
{
        for(a=0;a<i;a++)
        {
                for(b=0;b<i;b++)
                {
                        if(s[b].ind>s[b+1].ind)
                        {
                                t=s[b].ind;
                                s[b].ind=s[b+1].ind;
                                s[b+1].ind=t;
                                strcpy(tem,s[b].n);
                                strcpy(s[b].n,s[b+1].n);
                                strcpy(s[b+1].n,tem);
                                t=s[b].fld;
                                s[b].fld=s[b+1].fld;
                                s[b+1].fld=t;
                        }
                        else
                                continue;
                }
        }
         printf("\n ---------------------------------");
         printf("\n\t Index   Recordname    FieldNo");
         for(i=0;i<=j;i++)
         {
                printf("\n\t%d\t",s[i].ind);
                printf("\t%s",s[i].n);
                printf("\t%d",s[i].fld);
         }
         i--;
         printf("\n ------------------------------------\n");
}
void del()
{
        int de,index=-
        1,k=0,l; if(i!=-1)
        {
```

```
                printf("enter index no to be
                deleted"); scanf("%d",&de);
                index=de;
                while(s[k].ind!=de)
                {
                        k++;
                        printf("\n\t\t\t%d",k);
                }
                for(l=k;l<=j;l++)
                        s[l]=s[l+1];
                i--;
                j--;
                printf("\nindex no %d file is deleted",index);
        }
}
```

Output:

```
menu
1.create
2.display
3.delete
4.exitenter ur choice:1
```

```
enter the name of the recoed:a.java

enter the index no:0

enter the field no:1


menu
1.create
2.display
3.delete
4.exitenter ur choice:1

enter the name of the recoed:b.c

enter the index no:1

enter the field no:2


menu
1.create
2.display
3.delete
4.exitenter ur choice:2

 --------------------------------
```

```
       Index   Recordname    FieldNo
      0            a.java  1
      1            b.c    2
 ----------------------------------


menu
1.create
2.display
3.delete
4.exitenter ur choice:4
```

(c) File name:
LINKED.C Source code:

```
#include<stdio.h>
#include<stdlib.h>
typedef struct
{
        int
bno,flag,next; }block;
block b[200],b1;
void main()
{
        int rnum();
        int i,n,s,s1,p[30],r,k[20];
        clrscr();
        printf("\nEnter number of
        programs:"); scanf("%d",&n);
        printf("\nEnter the memory block
        request"); for(i=1;i<=n;i++)
        {
                printf("\nEnter program
                requirement"); scanf("%d",&p[i]);
        }
        for(i=1;i<=n;i++)
        {
                s=rnum();
                b[s].bno=0;
                b[s].flag=1;
                k[i]=0;
                r=p[i]-1;
                while(r!=0)
                {
                        s1=rnum();
                        b[s].next=s1;
                        b[s1].flag=1;
                        b[s1].bno=0;
                        s=s1;
                        r=r-1;
                }
                b[s1].next=NULL;
        }
        printf("\n Starting blocks for program");
```

```
        for(i=1;i<=n;i++)
                printf("\n%5d%5d",i,k[i]);
        printf("\n allocated blocks");
        for(i=1;i<=200;i++)
        {
                if(b[i].flag==1)
                        printf("\n%5d%5d",b[i].bno,b[i].next);
        }
}
int rnum()
{
        int k,i;
        for(i=1;i<=200;i++)
        {
                k=rand()%200;
                if(b[i].flag!=1)
                        break;
        }

        return k;
}
```

<u>Output:</u>

```
Enter number of programs:2

Enter the memory block request
Enter program requirement3

Enter program requirement4

 Starting blocks for program
     1   0
     2   0
 allocated blocks
    0 117
    0  56
    0 195
    0 182
    0 130
    0   0
    0   0
```

Computer Networks & Operating Systems Lab Manual

3. [AIM:] Simulate MVT and MFT <u>**Theory:**</u>

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.

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

<u>**File name: MVT.C Source code:**</u>

```
#include<stdio.h
> typedef struct
{
        int prano,memreq,flag;
}mvt;
void
main()
{
        mvt
        a[10],que[10],part[10];
        int
        i,n,rear=1,k=1,mvt=200;
        clrscr();
        printf("Enter no of
        processes");
        scanf("%d",&n);
        printf("\nEnter memory required for each
        process"); for(i=1;i<=n;i++)
        {
                a[i].prano
                =i;
                a[i].flag=0
                ;
                scanf("%d",&a[i].memreq);
        }
        for(i=1;i<=n;i++)
        {
                if(mvt>=a[i].memreq)
                {
                        a[i].flag=mvt-
                        a[i].memreq; mvt-
                        =a[i].memreq;
                        part[k]=a[i];
                        k++;
                }
                else
                {
                        que[rear]=a[i
                        ]; rear++;
                }
        }
        printf("\n Enter
        process");
        for(i=1;i<=n;i++)
        {
                printf("\n%4d%4d%4d",a[i].prano,a[i].memreq,a[i].flag);
        }
        printf("\n");
        printf("\n Process which are
        partioned are"); for(i=1;i<k;i++)
        {
```

```
                 printf("\n%4d%4d%4d",part[i].prano,part[i].memreq,part[i].flag);
        }
        printf("\n");
        printf("\n Process which are in waiting
        queues"); for(i=1;i<rear;i++)
        {
                 printf("\n%4d%4d%4d",que[i].prano,que[i].memreq,que[i].flag);
          }
        getch();
}
```

Output:

```
Enter no of processes4

Enter memory required for each
process2 3 4 5



 Enter process
   1 2 198 2 3
   195 3 4 191
   4 5 186


 Process which are
   partioned are 1 2 198 2 3
   195 3 4 191 4 5 186



 Process which are in waiting queues
```

File name: MFT.C
Source code:

```
#include<stdio.h>
typedef struct
{
        int prono,memreq,flag;
}
mftnode;
void main()
{
        mftnode a[10],memover[10],que[10],part[10];
        int larpar=250,medpar=100,smapar=50,ips=0,mps=0,sps=0;
        int i,n,c;
        float rear=1,j=1,k=1,front=1;
        clrscr();
        printf("\n enter no of
        processes\n"); scanf("%d",&n);
        printf("\n enter memory required for each
        process"); for(i=1;i<=n;i++)
        {
                a[i].prono=i;
                a[i].flag=0;
                scanf("%d",&a[i].memreq);
        }
        for(i=1;i<=n;i++)
        {
                if(larpar<a[i].memreq)
                {
                        memover[i]=a[i];j++;
                }
                else
                {
                        if(larpar>=a[i].memreq&&medpar<a[i].memreq)
                        {
                                if(ips==0)
                                {
                                        a[i].flag=larpar-
                                        a[i].memreq; part[c]=a[i];
                                        k++;
                                        ips=1;
                                }
                                else
                                {
                                        que[rear]=a[i];
                                        rear++;
                                }
                        }
                        if(medpar<=a[i].memreq&&smapar<a[i].memreq)
                        {
                                if(mps==0)
                                {
                                        a[i].flag=medpar-
                                        a[i].memreq; part[k]=a[i];
                                        k++;
                                        mps=1;
                                }
                                else
                                {
                                        que[rear]=a[i];
                                        rear++;
                                }
                        }
                        else
```

```
                    {
                            if(smapar>=a[i].memreq)
                            {
                                    if(sps==0)
                                    {
                                            a[i].flag=smapar-
                                            a[i].memreq; part[k]=a[i];
                                            k++;
                                            sps=1;
                                    }
                                    else
                                    {
                                            que[rear]=a[i];
                                            rear++;
                                    }
                            }
                    }
            }
        }
        printf("\n entered processes
        are"); for(i=1;i<=n;i++)
                printf("\n%4d%4d%4d",a[i].prono,a[i].memreq,a[i].flag);
        printf("\n");
        getch();
        printf("process which partitioned
        are:\n"); for(i=1;i<k;i++)
                printf("\n%4d%4d%4d",part[i].prono,part[i].memreq,part[i].flag);
        printf("\n");
        printf("\n Process which are in waiting
        que are"); for(i=front;i<rear;i++)
                printf("\n%4d%4d%4d",que[i].prono,que[i].memreq,que[i].flag);
        printf("\n\n");
        printf("Process which are in memory overflow
        stateore\n"); for(i=1;i<j;i++)
                printf("\n%4d%4d%4d",memover[i].prono,memover[i].memreq,memover[i].flag);
        printf("\n\n");
        getch();
}
```

Output:

```
 enter no of
processes 4
 enter memory required for each
process3 1 6 7



 entered processes are
   1  3  47
   2  1    0
   3  6    0
   4  7    0
 process which partitioned are:

   1  3  47

 Process which are in waiting que are
    2   1 0
    3   6 0
    4   7 0

Process which are in memory overflow state are
```

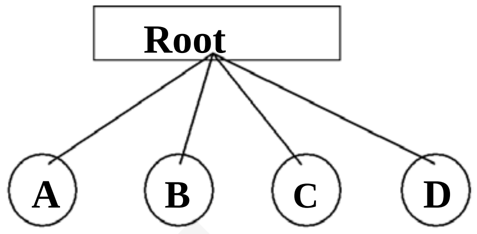
4. [AIM:] Simulate file organization techniques

- **a) Single Level Directory**
- **b) Two Level**
- **c) Hierarchical**
- **d) DAG**

THEORY:

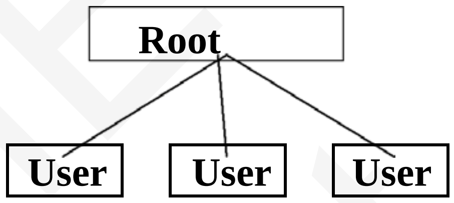
The directory contains information about the files, including attributes, location and ownership. Sometimes the directories consisting of subdirectories also. The directory is itself a file, owned by the o.s and accessible by various file management routines.

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).



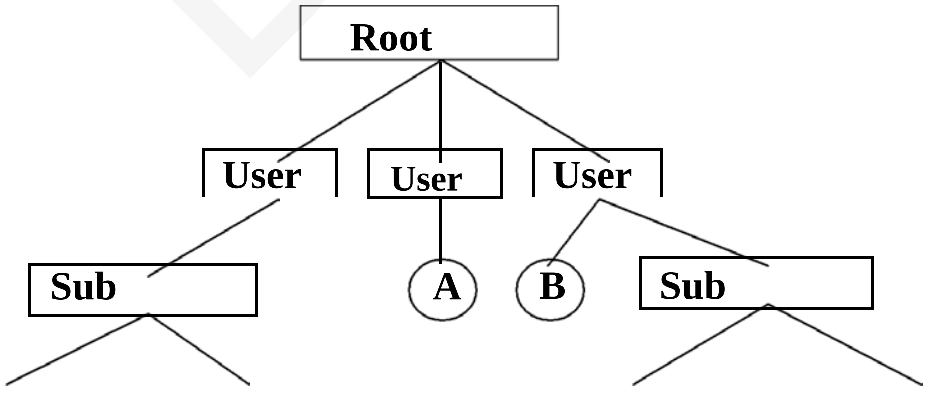
It has the simplicity and ability to locate files quickly. it is not used in the multi-user system, it is used on small embedded system.

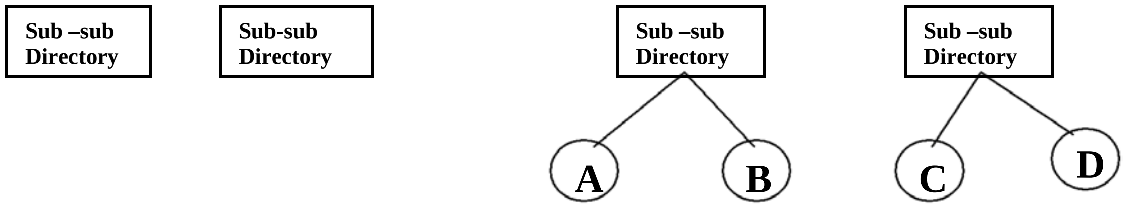
b) Two Level Directory: The problem in single level directory is different users may be accidentally using the same names for their files. To avoid this problem, each user need a private directory. In this way names chosen by one user don't interface with names chosen by a different user. The following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



Here root directory is the first level directory it consisting of entries of user directory. User1, User2, User3 are the user levels of directories. A, B, C are the files

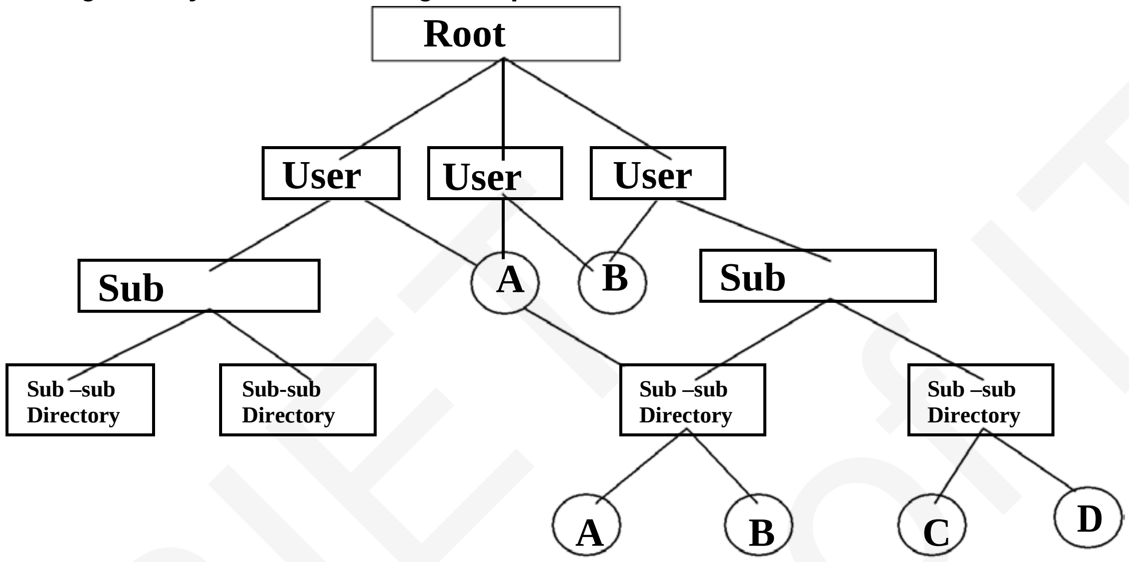
c) Hierarchical Directory: The two level directories eliminate name conflicts among users but it is not satisfactory for users but it is not satisfactory for users with a large no of files. To avoid this, create the subdirectory and load the same type of the files into the subdirectory. So, in this method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





This directory structure looks like tree, that's why it is also said to be tree-level directory structure

d) 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.

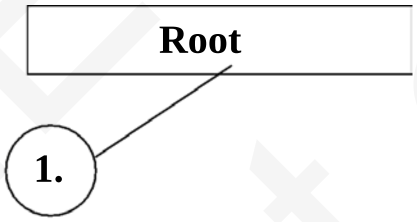


(a) File name:
SLD.c Source Code:

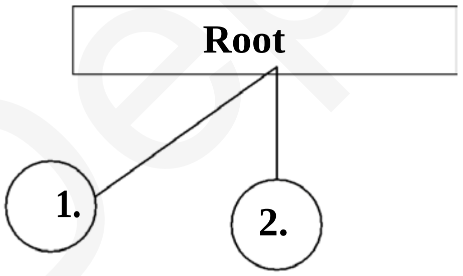
```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<graphics.h>
void main()
{
        int gd=DETECT,gm,count,i,j,mid,cir_x;
        char fname[10][20];
        clrscr();
        initgraph(&gd,&gm,"c:\\tc\\bgi");
        cleardevice(); setbkcolor(GREEN);

        puts("Enter no of files do u
        have?"); scanf("%d",&count);
        for(i=0;i<count;i++)
        {
                cleardevice(); setbkcolor(GREEN);
                printf("Enter file %d name",i+1);
                scanf("%s",fname[i]);
                setfillstyle(1,MAGENTA);
                mid=640/count;

                cir_x=mid/3;
                bar3d(270,100,370,150,0,0);
                settextstyle(2,0,4);
                settextjustify(1,1);
                outtextxy(320,125,"Root
                Directory"); setcolor(BLUE);
                for(j=0;j<=i;j++,cir_x+=mid)
                {
                        line(320,150,cir_x,250);
                        fillellipse(cir_x,250,30,30);
                        outtextxy(cir_x,250,fname[j]);
                }
                getch();
        }
}
```

OUTPUT:

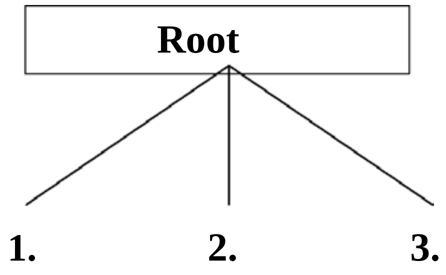
```
Enter no of files do u
have? 3

Enter file 1 name: 1.c
```



```
Enter File 2 name: 2.c
```



```
Enter File 3 name : 3.c
```



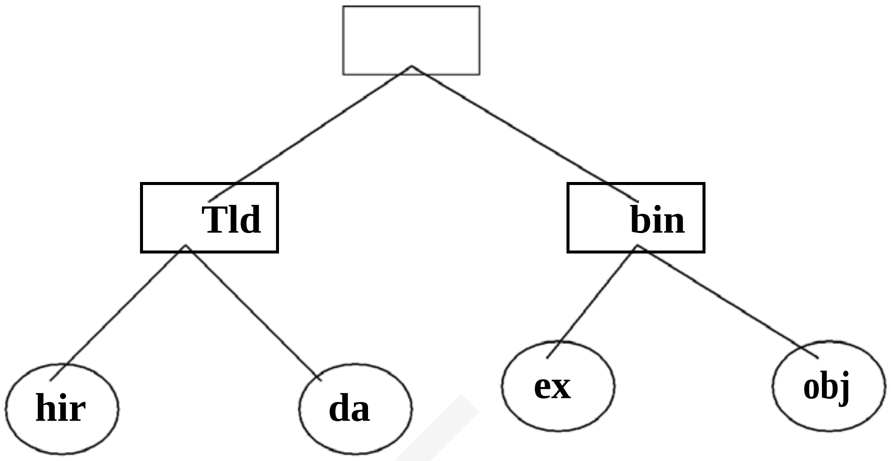
(b) File name:
TLD.c Source Code:

```
#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,"null",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);
                if(lev==0||lev==1)
                (*root)->ftype=1; else
                (*root)->ftype=2;
                (*root)->level=lev;

                (*root)->y=50+lev*50;
                (*root)->x=x; (*root)-
                >lx=lx; (*root)->rx=rx;
                for(i=0;i<5;i++)
                (*root)->link[i]=NULL;
                if((*root)->ftype==1)

                {
                        if(lev==0||lev==1)
                        {
                                if((*root)->level==0)
                                printf("How many
                                users"); else
                                printf("hoe many files");
                                printf("(for%s):",(*root)->name);
                                scanf("%d",&(*root)->nc);
                        }
                        else (*root)-
                        >nc=0; if((*root)-
                        >nc==0) gap=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
null):sld How many users(forsld):2
enter name of dir/file(under
sld):tld hoe many files(fortld):2
enter name of dir/file(under tld):hir
enter name of dir/file(under tld):dag
enter name of dir/file(under sld):bin
hoe many files(forbin):2
enter name of dir/file(under bin):exe
enter name of dir/file(under bin):obj
```



(c) File name:
HLD.c 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]);
                                }
```

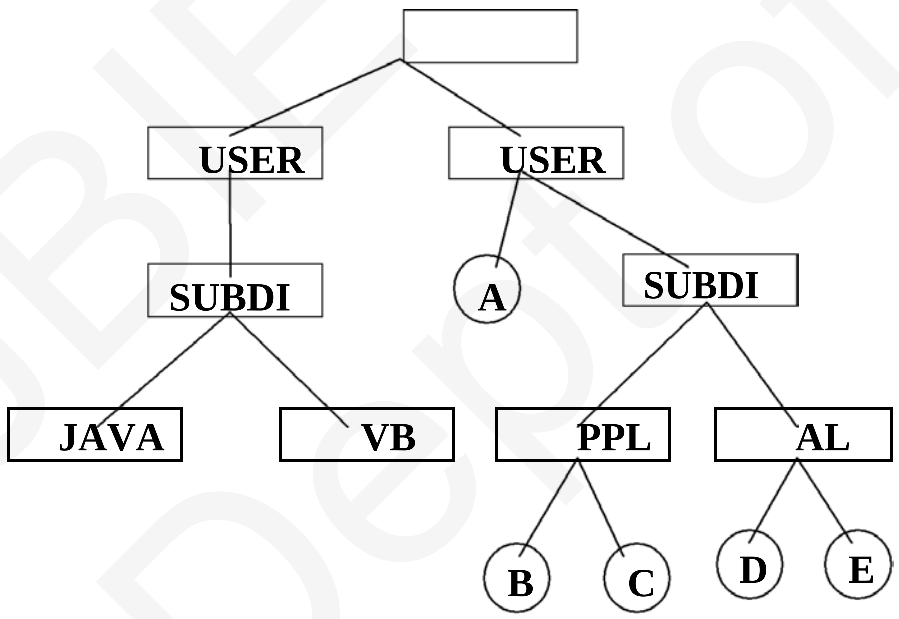
```
                    }
            }
```

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
```

OUTPUT:



(d) File name:
GGD.c Source Code:

```
#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
```

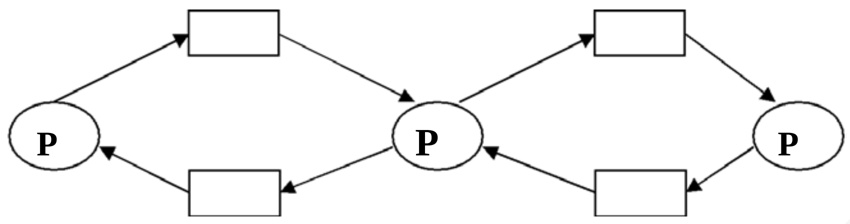
5. [Aim:] Simulate Bankers Algorithm for Deadlock Avoidance.

Theory:

Deadlock: A process request the resources, the resources are not available at that time, so the process enter into the waiting state. The requesting resources are held by another waiting process, both are in waiting state, this situation is said to be Deadlock. A deadlocked system must satisfied the following 4 conditions. These are:

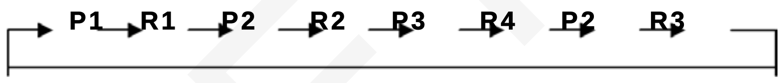
(i) Mutual Exclusion: Mutual Exclusion means resources are in non-sharable mode only, it means only one process at a time can use a process.

(ii) Hold and Wait: Each and every process is the deadlock state, must holding at least one resource and is waiting for additional resources, that are currently being held by another process.



(iii) No Preemption: No Preemption means resources are not released in the middle of the work, they released only after the process has completed its task.

(iv) Circular Wait: If process P1 is waiting for a resource R1, it is held by P2, process P2 is waiting for R2, R2 held by P3, P3 is waiting for R4, R4 is held by P2, P2 waiting for resource R3, it is held by P1.



Deadlock Avoidance: It is one of the method of dynamically escaping from the deadlocks. In this scheme, if a process request for resources, the avoidance algorithm checks before the allocation of resources about the state of system. If the state is safe, the system allocate the resources to the requesting process otherwise (unsafe) do not allocate the resources. So taking care before the allocation said to be deadlock avoidance.

Banker's Algorithm: It is the deadlock avoidance algorithm, the name was chosen because the bank never allocates more than the available cash.

Available: A vector of length 'm' indicates the number of available resources of each type. If available[j]=k, there are 'k' instances of resource types Rj available.

Allocation: An nxm matrix defines the number of resources of each type currently allocated to each process. If allocation[i,j]=k, then process Pi is currently allocated 'k' instances of resources type Rj. **Max:** An nxm matrix defines the maximum demand of each process. If max[i,j]=k, then Pi may request at most 'k' instances of resource type Rj.

Need: An nxm matrix indicates the remaining resources need of each process. If need[I,j]=k, then Pi may need 'k' more instances of resource type Rj to complete this task. There fore, Need[i,j]=Max[i,j]-Allocation[I,j]

FileName: deadlock.c

Source code:

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int work[5],avl[5],alloc[10][10],l;
        int need[10][10],n,m,I,j,avail[10],max[10][10],k,count,fcount=0,pr[10];
        char finish[10]={'f','f','f','f','f','f','f','f','f','f'};
        clrscr();
        printf("\n enter the no of process");
        scanf("%d",&n);
        printf("\n enter the no of resources");
        scanf("%d",&m);
        printf("\n enter the total no of  resources");
        for(i=1;i<=m;i++)
        scanf("%d",&avail[i]);
        printf("\n enter the max resources req by each pr alloc matrix");
        for(i=1;i<=n;i++)
        for(j=1;j<=m;j++)
        scanf("%d",&max[i][j]);
        printf("\n process allocation matrix");
        for(i=1;i<=n;i++)
        for(j=1;j<=m;j++)
        scanf("%d",&alloc[i][j]);
        for(i=1;i<=n;i++)
        for(j=1;j<=m;j++)
        need[i][j]=max[i][j]-alloc[i][j];
        for(i=1;i<=n;i++)
        {
                k=0;
                for(j=1;j<=m;j++)
                {
                        k=k+alloc[i][j];
                }
        avl[i]=avl[i]-k;
        work[i]=avl[i];
        }
        for(k=1;k<=n;k++)
        for(i=1;i<=n;i++)
        {
                count=0;
                for(j=1;j<=m;j++)
                {
                        if((finish[i]=='f')&&(need[i][j]<=work[i]))
                        count++;
                }
                if(count==m)
                {
                        for(l=1;l<=m;l++)
                        work[l]=work[l]+alloc[i][l];
                        finish[i]='t';
                        pr[k]=i;
                        break;
                }
        }
        for(i=1;i<=n;i++)
        if(finish[i]=='t')
        fcount++;
        if(fcount==n)
        {
                printf("\n the system is in safe state");
                for(i=1;i<=n;i++)
                printf("\n %d",pr[i]);
        }
        else
        printf("\n the system is not in safe state");
        getch();
}
```

OUT PUT:

```
Enter the no of process 5
Enter the no of resources
3 Enter the total no of
resources 10 5 7
Enter the max resource req. by each pr alloc matrix
7       5       3
3       2       2
9       0       2
2       2       2
4       3       3
Process allocation matrix
0       1       0
2       0       0
3       0       2
2       1       1
0       0       2
The system is in safe
state 1 3 4 5 2
```

6. [Aim:] Simulate all page replacement algorithms

a) FIFO
b) LRU
c) LFU

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.

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.

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.

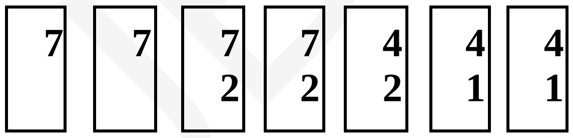
(a) FileName:
Fifo.c Source code:

```
#include<graphics.h>
#include<stdlib.h>
void main()
{
        //declare pages for store page nos, frames to store frame details
        int pages[10], frames[10][10],ppos[10],fillcount=0,least; //no_p
        stands for no of pages, no_f stands for no of frames
        int I,j,k,m,pos=0,no_p,no_f,faults=0;gd=DETECT,gm,no;
        int x1=20,y1=100;
        float ratio;
        char found,
        str[30],ch; clrscr();
        //GRAPHICS initialise method
        initgraph(&gd,&gm,"..//bgi");
        setbkcolor(BLUE);
        puts("Enter no of pages do u
        have"); scanf("%d",&no_p);
        puts("Enter no of frames do u
        have"); scanf("%d",&no_f);

        //initializing each frame
        with 0 for(i=0;i<no_f;i++)
        for(j=0;j<no_p;j++)
                frames[i][j]=0;
```

```
for(i=0;i<no_p;i++)
{
        puts("Enter page num");
        scanf("%d",&pages[i]);
        clrscr();
        cleardevice();
        x1=20,y1=100;
        found='f';
        for(j=0;j<no_f;j++)
        {
                if(i!=0) frames[j][i]=frames[j][i-
                        1];
                //checking whether page is there in frames
                or not if(frames[j][i]==pages[i])
                        found='t';
        }

        //if PAGE is not there in
        frames if(found=='f')
        {
                faults++;
                fillcount++;
                if(fillcount<=no_f)
                {
                        frames[pos][i]=pages[i];
                        pos++;
                }
                else
                {
                        for(j=0;j<no_f;j++)
                                ppos[j]=0;
                        for(j=0;j<no_f;j++)
                        {
                                for(k=i-3;k<i;k++)
                                {
                                        if(frames[j][i]==pages[k])
                                        ppos[j]=k;
                                }
                        }
                        least=ppos[0];
                        for(j=0;j<no_f;j++)
                        {
                                if(ppos[j]<least)
                                least=ppos[j];
                        }
                        for(j=0;j<no_f;j++)
                                if(pages[least]==frames[j][i])
                                pos=j;
                        frames[pos][i]=pages[i];
                }
        }

        //printing frames each time we
        enter a no settextstyle(2,0,6);
        for(k=0;k<no_f;k++)
        {
                for(j=0;j<=i;j++)
                {
                        rectangle(x1,y1,x1+40,y1+45);
                        if(frames[k][j]!=0)
                        {
                                //changing text color in case of
replacement
if(j==i&&frames[k][j]==pages[i]&&found=='f')
                                        setcolor(MAGENTA);
                                        else
                                        setcolor(WHITE);
                                        itoa(frames[k][j],str,10);
                                        outtextxy(x1+15,y1+15,str);
                                        }
                                        else
                                        outtextxy(x1+10,y1+10,"");
                                        setcolor(WHITE);
                                        x1+=55;
                              }
                              y1+=45;
                              x1=20;
                    }
          }
          //printing page fault ratio
          printf("/n/n page fault
          ratio=%f",(float)faults/(float)no_p); getch();
}
```

OUTPUT:

```
Enter no of pages do u have 7
Enter no of frames do u have 3
Enter page no 7
Enter page no 7
Enter page no 2
Enter page no 6
Enter page no 4
Enter page no 1
Enter page no 6

Page fault ratio=0.714286
```

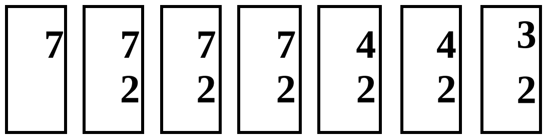


(b) FileName: LRU.c Source code:

```
#include<graphics.h>
#include<stdlib.h>
void main()
{
        //declare pages for stores page nos, frames to store frame
        details int pages[10],frames[10][10],ppos[10],fillcount=0,least;
        //no_pstands for no of pages, no_f stands for no of frames
        int i,j,k,m,pos=0;no_p,no_f,faults=0;gd=DETECT,gm,no;
        int x1=20,y1=100;
        float ratio;
        char found,
        str[30],ch,occur; clrscr();
        //GRAPHICS initialise method
        initgraph(&gd,&gm,"..//bgi");
        setbkcolor(BLUE);
        puts("Enter no of pages do u
        have"); scanf("%d",&no_p);
        puts("Enter no of frames do u
        have"); scanf("%d",&no_f);

        //initializing each frame
        with 0 for(i=0;i<no_f;i++)
        for(j=0;j<no_p;j++)
                frames[i][j]=0;

        for(i=0;i<no_p;i++)
        {
                puts("Enter page num");
                scanf("%d",&pages[i]);
                clrscr();
                cleardevice();
                x1=20,y1=100;
                found='f';
                for(j=0;j<no_f;j++)
                {
                        if(i!=0) frames[j][i]=frames[j][i-
                                1];
                        //checking whether page is there in frames
                        or not if(frames[j][i]==pages[i])
                                found='t';
                }

                //if PAGE is not there in
                frames if(found=='f')
                {
                        faults++;
                        fillcount++;
                        if(fillcount<=no_f)

                                frames[pos][i]=pages[i];
                                pos++;
                        }
                        else
                        {
                                for(j=0;j<no_f;j++)
                                        ppos[j]=0;
                                for(j=0;j<no_f;j++)
                                {
                                        for(k=0;k<i;k++)
                                        {
                                                if(frames[j][i]==pages[k])
                                                ppos[j]++;
                                        }
                                }
                                least=ppos[0];
                                for(j=0;j<no_f;j++)
                                {
                                        if(least>ppos[j])
                                        least=ppos[j];
                                }
                                ocurs='n';
                                for(j=0;j<1&&occur=='n';j++)
                                {
                                        for(k=0;k<no_f;k++)
                                        {
                                if(pages[j]==frames[k][i]&&ppos[k]==least)
                                {
                                        pos=k;
                                        occur='y';
                                }
                        }
                }
                frames[pos][i]=pages[i];
        }
    }
    //printing frames each time we
    enter a no settextstyle(2,0,6);
    for(k=0;k<no_f;k++)
    {
        for(j=0;j<=i;j++)
        {
                rectangle(x1,y1,x1+40,y1+45);
                if(frames[k][j]!=0)
                {
                //changing the text color when page is replaced
                if(j==i&&frames[k][j]==pages[i]&&found=='f')
                        setcolor(MAGENTA);
                else
                        setcolor(WHITE);
                itoa(frames[k][j],str,10);
                outtextxy(x1+15,y1+15,str);
                }
                else
                        outtextxy(x1+10,y1+10,"");
                setcolor(WHITE);
                x1+=55;
        }
        y1+=45;
        x1=20;
    }
    }
    //printing page fault ration
    printf("\n\n page fault ration
    %f",(float)faults/(float)no+p); getch();
    }
    //end of program
```

OUTPUT:

```
Enter no of pages do u have 7
Enter no of frames do u have 3
Enter page no 7
Enter page no 2
Enter page no 3
Enter page no 2
Enter page no 4
Enter page no 7
Enter page no 3

Page fault ratio=0.857143
```

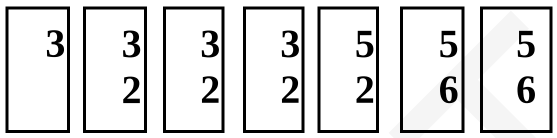


(c) FileName:
LFU.c Source code:

```
#include<graphics.h>
#include<stdlib.h>
void main()
{
        //declare pages for stores page nos, frames to store
        frame details int pages[10],frames[10][10];
        //no_pstands for no of pages, no_f stands for no of frames
        int  i,j,k,m,pos=0;no_p,no_f,faults=0;gd=DETECT,gm,no;  int
        x1=20,y1=100;
        float ratio;
        char found,
        str[30],ch; clrscr();
        //GRAPHICS initialise method
        initgraph(&gd,&gm,"..//bgi");
        setbkcolor(BLUE);
        puts("Enter no of pages do u
        have"); scanf("%d",&no_p);
        puts("Enter no of frames do u
        have"); scanf("%d",&no_f);

        //fill all frames with 0
        for(i=0;i<no_f;i++)
for(i=0;i<no_f;i++)
        for(j=0;j<no_p;j++)
                frames[i][j]=0;

        for(i=0;i<no_p;i++)
        {
                puts("Enter page num");
                scanf("%d",&pages[i]);
                clrscr();
                cleardevice();
                x1=20,y1=100;
                found='f';
                for(j=0;j<no_f;j++)
                {
                        if(i!=0) frames[j][i]=frames[j][i-
                                1];
                        //checking whether page is there in frames
                        or not if(frames[j][i]==pages[i])
                                found='t';
                }

                //if PAGE is not there in
                frames if(found=='f')
                {
                        frames[pos][i]=pages[i];
                        faults++;
                        if(pos<no_f)
                                pos++;
                        else
                                pos=0;
                }
                //printing frames each time we
enter a no settextstyle(2,0,6);
for(k=0;k<no_f;k++)
{
        for(j=0;j<=i;j++)
        {
                rectangle(x1,y1,x1+40,y1+45);
                if(frames[k][j]!=0)
                {
                //changing the text color when page is replaced
                if(j==i&&frames[k][j]==pages[i]&&found=='f')
                        setcolor(MAGENTA);
                else
                        setcolor(WHITE);
                itoa(frames[k][j],str,10);
                outtextxy(x1+15,y1+15,str);
                }
                else
                        outtextxy(x1+10,y1+10,"");
                setcolor(WHITE);
                x1+=55;
        }
        y1+=45;
        x1=20;
}
}
//printing page fault ratio
printf("\n\n page fault ration
%f",(float)faults/(float)no_p); getch();
}
//end of program
```

OUTPUT:

```
Page fault ratio=0.857143
Enter no of pages do u have 7
Enter no frames do u have 3
Enter page no 3
Enter page no 2
Enter page no 4
Enter page no 2
Enter page no 5
Enter page no 6
Enter page no 10
```



7. [Aim:] Simulate paging technique of memory management.

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.

File Name: PTMM.C
Source code:

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int ps,ms,np,nf,pt[20],I,page,offset,id,ph_add;
        clrscr();
        printf(" \n Enter page size, memorysize, no of
        pages"); scanf("%d%d%d",&ps,&ms,&np);
        nf=ms/ps;
        for(i=0;i<np;i++)
        {
                printf(" \n Enter the size of local
                address"); scanf("%d",&id);
                page=id%ps;
                ph_add=pt[page]*ps+offset;
                printf(" \n physical address is %d",ph_add);
                printf(" \n no of frames+%d \n pages=%d \n offset
                %d",nf,page,offset); getch();
        }
}
```

OUTPUT:

```
Enter page size, memory size, no of pages 5 2 2
Enter the size of logical address 10
Physical address is 18906
No of frames+0
Page=0
Offset=3240
```

OUTPUT:

```
Enter the no of process 5
Enter the no of resources 3
Enter the total no of resources
10      5       7

enter the max resource req by each pr alloc matrix

7       5       3
3       2       2
```

```
9       0       2
2       2       2
4       3       3
```

process allocation matrix

```
0       1       0
2       0       0
3       0       2
2       1       1
0       0       2
```

the system is in safe state

```
1
3
4
5
2
```

LPLAB MANUAL

```
Write a shell
script to
generate a
multiplication
table

echo
Multiplication
Table:
echo Which table
do you want ?
(Give Number):
read num
iter=1
while [ $num –le
5 ]
do
res=`expr $num
\* $iter`
echo $num “*”
$iter “=” $res
iter=`expr $iter
+ 1`
done
```

Write a shell script that copies multiple files to a directory.

```
 iter=1
echo Enter new
dir:
read nn
mkdir $nn
echo Enter
number of files:
read na
while [  $iter –le
$na ]
do
echo Enter file
name:
read fn
cp $fn $nn
iter=`expr $iter +
1`
done
```

Write a shell
script which
counts the
 number of lines
and words
present in
 a given file.

```
echo Enter a file
name:
read fn
echo Number of
Lines:
wc –l $fn
echo Number of
Words:
wc –w $fn
echo Number of
Characters:
wc –c $fn
```

Write a shell
script which
displays the
 list of all files in

the given
directory.

```
echo Menu
echo 1.Short
format display
echo 2.Long
format display
echo 3.Hidden
files to display
echo Enter ur
choice:
read ch
case ch in
1)      ls $a;;
2)      ls –l $a;;
3)      ls –la $a;;
*)  echo Choice
is not correct;;
Esac
```

Write a shell
script(small
calculator) that
adds, subtracts,
multiplies
and divides the
given two
integers.
There are two
division options:
one returns the
quotient and the
other
returns reminder.
The script
requires
3 arguments: The
operation to be
used and two
integer numbers.
The options are

add(-a),
subtract(-s),
multiply(-m),
quotient(-c) and
reminder(-r).

```
echo "Enter First
Value "
read x
echo "Enter
Second Value "
read y
while  [$q –ne 0 ]
do
echo “Enter –a
for adding”
echo “Enter –s
for subtraction”
echo “Enter –m
for
multiplication”
echo “Enter –c
for Quotient”
echo “Enter –r
for reminder”
read s
case $s in
-a)  p=`expr $x +
$y`
    Echo "Sum =
$p"
;;
-b)  p=`expr $x -
$y`
    Echo
"difference  =
$p"
;;
-m)  p=`expr $x
\* $y`
    Echo "Product
= $p"
;;
```

```
-c)  p=`expr $x /
$y`
    Echo
"quotient = $p"

;;

-r)  p=`expr $x %
$y`
    Echo
“reminder = $p"
;;
```

Write a shell
script to reverse
the rows
and columns of a
matrix.

```
Echo "Enter
Number of rows"
read  r
Echo "Enter
Number of
columns"
read  c
i=0
echo "Enter
elements"
until [ $i –eq
`expr $r \* $c` ]
do
      read a[$i]
      i= `expr $i +
1`
done
i=0 ;  k=0
echo "Transpose
of a Matrix"
until [ $i –eq $c ]
do
      j=0;
      until [ $j –eq
$r ]
        do
                n= `expr
$j \* $c`
                m=
`expr $n + $i
                b[$k] =
${a[$m]}
                echo
"${b[$k]} \t"
                k= `expr
$k + 1`
                j= `expr
$j + 1`
        done
        i = `expr $i +
1`
        echo "\n"
done
```

Write a C program that counts the number of blanks in a text file using standard I/O

```
#include
<fcntl.h>
#include <
sys/stat.h>
#include
<stdio.h>
int main(int argc,
char **argv)
{
   FILE *fd1;
   int n,count=0;
   char buf;

fd1=fopen(argv[1
],"r");

while(!feof(fd1))
   {
   buf=fgetc(fd1);
     if(buf==' ')

count=count+1;
   }
   printf("\n Total
Blanks=
%d",count);
return (0);
}
```


Write a C program that counts the number of blanks in a text file using system calls

```
#include<fcntl.h
>
#include<sys/stat
.h>

int main(int argc,
char **argv)
{
   int fd1;
   int n,count=0;
   char buf;

fd1=open(argv[1]
,O_RDONLY);
while((n=read(fd
1,&buf,1))>0)
   {
     if(buf==' ')

count=count+1;
   }
   printf("\n Total
Blanks=
%d",count);
return (0);
}
```

Implement in C the cat Unix command using system calls

```
#include<fcntl.h
>
#include<sys/stat
.h>
#define
BUFSIZE 1
int main(int argc,
char **argv)
{
   int fd1;
   int n;
   char buf;

fd1=open(argv[1]
,O_RDONLY);

printf("SuhritSol
utions Printing
Files\n");

while((n=read(fd
1,&buf,1))>0)
   {

printf("%c",buf);
/*           or

write(1,&buf,1);
*/

   }
   return (0);
}


---------------------
---------------------
--------------
```

Implement in C the following ls Unix command using system calls

```
#include
<sys/types.h>
#include
<sys/dir.h>
#include
<sys/param.h>
#include
<stdio.h>

#define FALSE 0
#define TRUE 1

extern  int
alphasort();

char
pathname[MAXP
ATHLEN];

main()   {
int count,i;
struct dirent
**files;
```

```
int file_select();

if
(getwd(pathname
) == NULL )
{ printf("Error
getting pathn");
exit(0);
}

        printf("Cur
rent Working
Directory =
%sn",pathname);

        count =
scandir(pathname
, &files,
file_select,
alphasort);


                if
(count <= 0)
                {

printf("No files
in this
directoryn");




        exit(0);


                }


        printf("Nu
mber of files =
%dn",count);

        for
```

```
(i=1;i<count+1;+
+i)


printf("%s
\n",files[i-1]-
>d_name);


                    }






int
file_select(struct
direct   *entry)

{
if ((strcmp(entry-
>d_name, ".") ==
0)
||(strcmp(entry-
>d_name, "..")
== 0))



return (FALSE);
else


          return
(TRUE);
                    }
```


Implement in C
the Unix

command mv
using system
calls

```
#include<fcntl.h
>
#include<stdio.h
>
#include<unistd.
h>
#include<sys/stat
.h>
int main(int argc,
char **argv)
{
   int fd1,fd2;
   int n,count=0;

fd1=open(argv[1]
,O_RDONLY);
fd2=creat(argv[2]
,S_IWUSR);
rename(fd1,fd2);
unlink(argv[1]);
return (0);
}
```

Write a c
program for
message passing
using pipes.

```
#include
<stdio.h>
#include
<sys/types.h>
#include
<unistd.h>
int main()
{
int fd[2];
if(pipe(fd)<0)
exit(1);
if(fork())
{
close(fd[0]);
write(fd[1],
“Message from
Suhrit”12);
}
else
{
char buf[100];
close(fd[1]);
read(fd[0],buf,10
0);
printf(“Received
by Students of
SuhritSolutions:
%s\n”,buf);
fflush(stdout);
}
exit(0);
}
```

Write a C program that illustrates the creation of child process using fork system call. One process finds sum of even series and other process finds sum of odd series

```
#include
<stdio.h>
#include
<sys/types.h>
#include
<unistd.h>
```

```
#include
<fcntl.h>
int main()
{
int i,n,sum=0;
pid_t pid;
system(“clear”);
printf(“Enter n
value:”);
scanf(“%d”,&n)
pid=fork();
if(pid==0)
{
printf(“From
child process\n”);
for(i=1;i<n;i+=2)
{
printf(“%d\”,i);
sum+=i;
}
printf(“Odd
sum:%d\n”,sum);
}
else
{
printf(“From
process\n”);
for(i=0;i<n;i+=2)
{
printf(“%d\”,i);
sum+=i;
}
printf(“Even
sum:%d\n”,sum);
}
}
```

Write a C program that displays the real time of a day every 60 seconds

```
    #include
<stdio.h>
    #include
<sys/time.h>
    #include
<sys/signal.h>

/* Declarations */
       void
main();
    int
times_up();

    void main()
    {

      for (; ;)
        {

times_up(1);
         sleep(60);

        }
    }

 int times_up(sig)
    int sig;
    {
     long now;
     long
time(struct tms
*ptr);
      char
*ctime();

      time
(&now);
      printf("It is
now %s\n", ctime
(&now));
return (sig);
    }
```

Write a C program that illustrates file locking using semaphores.

```
#include
<stdio.h>
 #include
<sys/file.h>
 #include
<error.h>
 #include
<sys/sem.h>
 #define
MAXBUF 100
 #define KEY
1216
 #define
SEQFILE
“suhritfile”
 int semid,fd;
 void
my_lock(int);
 void
my_unlock(int);
 union semnum
 {
     int val;
     struct
semid_ds *buf;
     short *array;
 }arg;
 int main(])
 {
       int child, i,n,
pid, seqno;
       char
buff[MAXBUF+
1];

pid=getpid();
```

```
if((semid=semget
(KEY, 1,
IPC_CREAT |
0666))= = -1)
        {

perror("semget");

exit(1);
        }
        arg.val=1;

if(semctl(semid,0
,SETVAL,arg)<0
)

perror("semctl");

if((fd=open(SEQ
FILE,2))<0)
        {

perror("open");

exit(1);
        }

pid=getpid();

for(i=0;i<2;i++)
        {

my_lock(fd);

lseek(fd,01,0);

if((n=read(fd,buf
f,MAXBUF))<0)
                 {

perror("read");

exit(1);
                 }
printf("pid:%d,
Seq no:%d\n",
pid, seqno);

seqno++;

sprintf(buff,"%d\
n", seqno);

n=strlen(buff);

lseek(fd,01,0);

if(write(fd,buff,n
)!=n)
                {

perror("write");

exit(1);
                }

sleep(1);

my_unlock(fd);
        }
 }
    void
my_lock(int fd)
    {
             struct
sembuff sbuf=(0,
-1, 0);

if(semop(semid,
&sbuf, 1)= =0)

printf("Locking:
Resource…\n");
             else

printf("Error in
Lock\n");
```

```
    }
    void
my_unlock(int
fd)
    {
              struct
sembuff sbuf=(0,
1, 0);

if(semop(semid,
&sbuf, 1)= =0)

printf("UnLockin
g:
Resource…\n");
              else

printf("Error in
Unlock\n");
    }
```

Write a C program that implements a producer-consumer system with two processes.(using semaphores)

```
#include
<stdio.h>
#include
<stdlib.h>
#include
<unistd.h>
#include
<time.h>
#include
<sys/types.h>
#include
<sys/ipc.h>
```

```
#include
<sys/sem.h>

#define
NUM_LOOPS
          20
int main(int argc,
char* argv[])
{
    int sem_set_id;

    union semun
sem_val;
    int child_pid;

    int i;

    struct sembuf
sem_op;
    int rc;

    struct timespec
delay;

    sem_set_id =
semget(IPC_PRI
VATE, 1, 0600);
    if (sem_set_id
== -1) {
          perror("ma
in: semget");
          exit(1);
    }

printf("semaphor
e set created,
 semaphore set id
'%d'.\n",
sem_set_id);

    sem_val.val =
0;
    rc =
semctl(sem_set_i
d, 0, SETVAL,
sem_val);
    child_pid =
fork();
    switch
(child_pid) {
          case -1:


perror("fork");
              exit(1);
          case 0:

              for (i=0;
i<NUM_LOOPS;
i++) {

          sem_op.se
m_num = 0;

          sem_op.se
m_op = -1;

          sem_op.se
m_flg = 0;

          semop(sem
_set_id,
&sem_op, 1);

          printf("con
sumer: '%d'\n",
i);

          fflush(stdo
ut);

sleep(3);
              }
              break;
          default:

              for (i=0;
i<NUM_LOOPS;
i++)
            {

          printf("pro
ducer: '%d'\n", i);

          fflush(stdo
ut);

          sem_op.se
m_num = 0;

          sem_op.se
m_op = 1;

          sem_op.se
m_flg = 0;

          semop(sem
_set_id,
&sem_op, 1);

          sleep(2);
                    if
(rand() >
3*(RAND_MAX
/4))
                    {

delay.tv_sec = 0;

delay.tv_nsec =
10;

nanosleep(&dela
y, NULL);
                    }
              }
              break;
    }

    return 0;
}
```

Write a C program that illustrates inter process communication using shared memory system calls.

```
#include
<stdio.h>

#include<sys/ipc.
h>

#include<sys/sh
m.h>

#include<sys/typ
es.h>
 #define
SEGSIZE 100
 int main(int
argc, char *argv[
])
 {
        int
shmid,cntr;
        key_t key;
        char
*segptr;
        char buff[
]=”Hello world”;

key=ftok(“.”,’s’);

if((shmid=shmge
t(key, SEGSIZE,
IPC_CREAT |
IPC_EXCL |
0666))= = -1)
```

```
        {

if((shmid=shmge
t(key,SEGSIZE,0
))= = -1)
                {

perror("shmget")
;

exit(1);
                }
        }
        else
        {

printf("Creating a
new shared
memory seg \n");

printf("SHMID:
%d", shmid);
        }

system("ipcs –
m");

if((segptr=shmat(
shmid,0,0))=
=(char*)-1)
       {

perror("shmat");
              exit(1);
       }

printf("Writing
data to shared
memory…\n");

strcpy(segptr,buff
);

printf("DONE\n"
);

printf(“Reading
data from shared
memory…\n”);

printf(“DATA:-
%s\n”segptr);

printf(“DONE\n”
);

print(“Removing
shared memory
Segment…\n”);

if(shmctl(shmid,I
PC_RMID,0)= =
-1)

printf(“Can’t
Remove Shared
memory
Segment…\n”);
      else

printf(“Removed
Successfully”);
 }
```

Write a C
program that
illustrates
the following.
a) Creating a
message queue.
b) Writing to a
message queue.
c) Reading from
a message queue.

```
#include
<stdio.h>
 #include
<sys/ipc.h>
 #include
<fcntl.h>
 #define MAX
255
        struct mesg
        {
              long
type;
              char
mtext[MAX];
        }  *mesg;
        char
buff[MAX];
 main()
 {
        int
mid,fd,n,count=0
;;

if((mid=msgget(1
006,IPC_CREAT
| 0666))<0)
        {

printf("\n Can't
create Message
Q");
                exit(1);
        }
        printf("\n
Queue id:%d",
mid);

mesg=(struct
mesg
*)malloc(sizeof(s
truct mesg));
        mesg -
>type=6;
```

```
fd=open(“fact”,O
_RDONLY);

while(read(fd,buf
f,25)>0)
        {

strcpy(mesg -
>mtext,buff);

if(msgsnd(mid,m
esg,strlen(mesg -
>mtext),0)== -1)

printf(“\n
Message Write
Error”);
        }



if((mid=msgget(1
006,0))<0)
        {

printf(“\n Can’t
create Message
Q”);
                exit(1);
        }

while((n=msgrcv
(mid,&mesg,MA
X,6,IPC_NOWA
IT))>0)

write(1,mesg.mte
xt,n);

count++;
        if((n= = -
1)&(count= =0))

printf(“\n No
Message Queue
on
Queue:%d”,mid);

 }
```