

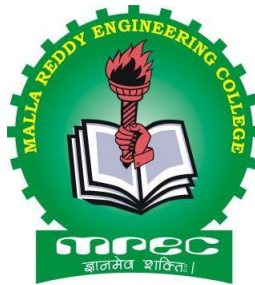
Department of Computer Science and Engineering
(Internet of Things)

III B. Tech I Semester

Subject Name: Operating Systems Lab Manual

Subject Code: B0514

Regulations: MR-20



Academic Year: 2022-23



MALLAREDDY ENGINEERING COLLEGE

(An UGC Autonomous Institution, Approved by AICTE and Affiliated to JNTUH Hyderabad,
Recognized under section 2(f) & 12(B) of UGC Act 1956,
Accredited by NAAC with 'A' Grade (II Cycle) and NBA
Maisammaguda, Dhulapally (Post Via Kompally), Secunderabad-500 100

2020-21 Onwards (MR-20)	MALLA REDDY ENGINEERING COLLEGE (Autonomous)	B.Tech. V Semester		
Code: A0534	Operating Systems Lab (Common for CSE, CSE (Cyber Security), CSE (AI and ML), CSE (DS), CSE (IOT) and IT)	L	T	P
Credits: 2		-	1	2

Prerequisites: NIL

Course Objectives:

This course enable the students to interpret main components of operating system and their working, identify the role of Operating System in process scheduling and synchronization, analyze the way of addressing deadlock, understand memory management techniques and I/O systems, describes the way of handling files and security.

Software Requirements: C++/JDK

List of Programs:

1. Simulate the following CPU scheduling algorithms
a) FCFS b) SJF
2. Simulate the following CPU scheduling algorithms
a) Priority b) Round Robin
3. Simulate the Producer Consumer Problem
4. Simulate Bankers Algorithm for Dead Lock Avoidance
5. Simulate MVT and MFT techniques.
6. Simulate Paging Technique of memory management
7. Simulate page replacement algorithms a) FIFO b) LRU c) Optimal
8. Simulate the following Disk Scheduling Algorithms
(a) First Come-First Serve (FCFS)
(b) Shortest Seek Time First (SSTF)
9. Simulate the following Disk Scheduling Algorithms
(a) Elevator (SCAN)
(b) LOOK
10. Simulate all file allocation strategies a) Sequential b) Indexed c) Linked
11. Simulate File Organization Techniques
a) Single level directory b) Two level
12. Simulate File Organization Techniques
a) Hierarchical b) DAG

TEXT BOOKS:

1. Abraham Silberchatz, Peter B. Galvin, Greg Gagne, “**Operating System Principles**” 7th Edition, John Wiley.
2. Stallings “**Operating Systems Internal and Design Principles**”, Fifth Edition-2005, Pearson education/PHI

REFERENCES:

1. Crowley,” **Operating System A Design Approach**”,TMH.
2. Andrew S Tanenbaum,” **Modern Operating Systems**”, 2nd edition Pearson/PHI.
3. Pramod Chandra P. Bhat, “**An Introduction to Operating Systems**”, Concepts and Practice”, PHI, 2003
4. DM Dhamdhare,” **Operating Systems A concept based approach**” ,2nd Edition, TMH

Course Outcomes:

At the end of the course, students will be able to

1. **Implement** various CPU scheduling algorithms, Bankers algorithms used for deadlock avoidance and prevention.
2. **Develop** disk scheduling algorithms and apply File organization techniques.
3. **Simulate** file allocation method

CO- PO, PSO Mapping (3/2/1 indicates strength of correlation) 3-Strong, 2-Medium, 1-Weak															
COs	Programme Outcomes (POs)												PSOs		
	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3
CO1	2	3	1									2	1		
CO2	2	2										2	2		
CO3	1	2										1	1		

(1a) Aim: Write a program to simulate the FCFS CPU scheduling algorithm.

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int i,j,n,bt[20],wt[20],tat[20],avwt=0,avtat=0;
    printf("enter no.of processes:");
    scanf("%d",&n);
    printf("\nenter process burst time\n");
    for(i=0;i<n;i++)
    {
        printf("p[%d]:",i+1);
        scanf("%d",&bt[i]);
    }
    wt[0]=0;
    for(i=0;i<n;i++)
    {
        wt[i]=0;
        for(j=0;j<i;j++)
        {
            wt[i]=wt[i]+bt[j];
        }
    }
    printf("\nP\tBT\tWT\tTAT:\n");
    for(i=0;i<n;i++)
    {
        tat[i]=bt[i]+wt[i];
        avwt+=wt[i];
        avtat+=tat[i];
        printf("p[%d]\t%d\t%d\t%d\n",i+1,bt[i],wt[i],tat[i]);
    }
    avwt/=n;
    printf("avg waiting time=%d\n",avwt);
    avtat/=n;
    printf("avg turnaround time=%d\n",avtat);
    getch();
}
```

OUTPUT:

```
enter no.of processes:3
enter process burst time
p[1]:15
p[2]:4
p[3]:12
P      BT      WT      TAT:
p[1]   15      0      15
p[2]   4       15     19
p[3]   12     19     31
avg waiting time=11
avg turnaround time=21
```

(1 b) AIM : Write a program to simulate the SJF CPU scheduling algorithm

PROGRAM:

```
#include<stdio.h>
void main()
{
    int bt[20],p[20],wt[20],tat[20],i,j,n,total=0,pos,temp;
    float avg_wt,avg_tat;
    printf("Enter number of process:");
    scanf("%d",&n);

    printf("\nEnter Burst Time:\n");
    for(i=0;i<n;i++)
    {
        printf("\np%d:",i+1);
        scanf("%d",&bt[i]);
        p[i]=i+1;        //contains process number
    }

    //sorting burst time in ascending order using selection sort
    for(i=0;i<n;i++)
    {
        pos=i;
        for(j=i+1;j<n;j++)
        {
            if(bt[j]<bt[pos])
                pos=j;
        }

        temp=bt[i];
        bt[i]=bt[pos];
        bt[pos]=temp;
    }

    //calculating waiting and turnaround time
    wt[0]=0;
    for(i=1;i<n;i++)
    {
        wt[i]=wt[i-1]+bt[i-1];
    }

    for(i=0;i<n;i++)
    {
        tat[i]=wt[i]+bt[i];
    }

    avg_wt=(float)total/n;
    avg_tat=(float)total/n;
    printf("\nAverage waiting time is: %f",avg_wt);
    printf("\nAverage turnaround time is: %f",avg_tat);
}
```

```

    bt[i]=bt[pos];
    bt[pos]=temp;

    temp=p[i];
    p[i]=p[pos];
    p[pos]=temp;
}

wt[0]=0;      //waiting time for first process will be zero

//calculate waiting time
for(i=1;i<n;i++)
{
    wt[i]=0;
    for(j=0;j<i;j++)
    wt[i]+=bt[j];

    total+=wt[i];
}

avg_wt=(float)total/n;    //average waiting time
total=0;

printf("\nProcess\t Burst Time \tWaiting Time\tTurnaround Time");
for(i=0;i<n;i++)
{
    tat[i]=bt[i]+wt[i];    //calculate turnaround time
    total+=tat[i];
    printf("\n%d\t\t %d\t\t %d\t\t%d",p[i],bt[i],wt[i],tat[i]);
}

avg_tat=(float)total/n;    //average turnaround time
printf("\n\nAverage Waiting Time=%f",avg_wt);
printf("\n\nAverage Turnaround Time=%f\n",avg_tat);
}

```

OUTPUT:

```
Enter number of process:3
Enter Burst Time:
p1:15
p2:4
p3:12
Process      Burst Time      Waiting Time      Turnaround Time
p2           4           0           4
p3           12          4          16
p1           15          16          31
Average Waiting Time=6.666667
Average Turnaround Time=17.000000
```

(2a) Aim : Write a program to simulate the Priority CPU scheduling algorithm

PROGRAM:

```
#include<stdio.h>
int main()
{
    int bt[20],p[20],wt[20],tat[20],pr[20],i,j,n,total=0,pos,temp,avg_wt,avg_tat;
    printf("Enter Total Number of Process:");
    scanf("%d",&n);

    printf("\nEnter Burst Time and Priority\n");
    for(i=0;i<n;i++)
    {
        printf("\nP[%d]\n",i+1);
        printf("Burst Time:");
        scanf("%d",&bt[i]);
        printf("Priority:");
        scanf("%d",&pr[i]);
        p[i]=i+1; //contains process number
    }

    //sorting burst time, priority and process number in ascending order using selection sort
    for(i=0;i<n;i++)
```

```

    {
        pos=i;
        for(j=i+1;j<n;j++)
        {
            if(pr[j]<pr[pos])
                pos=j;
        }

        temp=pr[i];
        pr[i]=pr[pos];
        pr[pos]=temp;

        temp=bt[i];
        bt[i]=bt[pos];
        bt[pos]=temp;

        temp=p[i];
        p[i]=p[pos];
        p[pos]=temp;
    }
    wt[0]=0;//waiting time for first process is zero
    //calculate waiting time
    for(i=1;i<n;i++)
    {
        wt[i]=0;
        for(j=0;j<i;j++)
            wt[i]+=bt[j];

        total+=wt[i];
    }
    avg_wt=total/n;    //average waiting time
    total=0;

    printf("\nProcess\t Burst Time \tWaiting Time\tTurnaround Time");
    for(i=0;i<n;i++)
    {
        tat[i]=bt[i]+wt[i];    //calculate turnaround time
        total+=tat[i];
        printf("\nP[%d]\t\t %d\t\t %d\t\t\t%d",p[i],bt[i],wt[i],tat[i]);
    }

    avg_tat=total/n;    //average turnaround time
    printf("\n\nAverage Waiting Time=%d",avg_wt);
    printf("\n\nAverage Turnaround Time=%d\n",avg_tat);

```



```

    return 0;
}

```

OUTPUT:

```

Enter Burst Time and Priority

P[1]
Burst Time:5
Priority:2

P[2]
Burst Time:15
Priority:4

P[3]
Burst Time:17
Priority:1

P[4]
Burst Time:20
Priority:3

Process      Burst Time      Waiting Time      Turnaround Time
P[3]          17              0                 17
P[1]           5              17                22
P[4]          20              22                42
P[2]          15              42                57

Average Waiting Time=20
Average Turnaround Time=34

```

(2b) Aim: Write a program to simulate the Round Robin CPU scheduling Algorithm.

PROGRAM:

```

#include<stdio.h>
#include<conio.h>
main()
{
    int st[10],bt[10],wt[10],tat[10],n,tq;
    int i,count=0,swt=0,stat=0,temp,sq=0;
    float awt=0.0,atat=0.0;
    printf("Enter number of processes:");
    scanf("%d",&n);
    printf("Enter burst time for sequences:");
    for(i=0;i<n;i++)
    {
        printf("\np[%d]:",i+1);
    }
}

```

```

scanf("%d",&bt[i]);
st[i]=bt[i];
}
printf("Enter time quantum:");
scanf("%d",&tq);
while(1)
{
for(i=0,count=0;i<n;i++)
{
temp=tq;
if(st[i]==0)
{
count++;
continue;
}
if(st[i]>tq)
st[i]=st[i]-tq;
else
if(st[i]>=0)
{
temp=st[i];
st[i]=0;
}
sq=sq+temp;
tat[i]=sq;
}
if(n==count)
break;
}
for(i=0;i<n;i++)
{
wt[i]=tat[i]-bt[i];
swt=swt+wt[i];
stat=stat+tat[i];
}
awt=(float)swt/n;
atat=(float)stat/n;
printf("Process_no Burst time Wait time Turn around time ");
for(i=0;i<n;i++)
printf("\n\t %d \t %d \t %d \t %d ",i+1,bt[i],wt[i],tat[i]);
printf("\n Avg wait time is %f\n Avg turn around time is %f",awt,atat);
getch();
return 0;
}

```

OUTPUT:

```
Enter number of processes:3
Enter burst time for sequences:15
4
12
Enter time quantum:4
Process_no Burst time Wait time Turn around time
          1          15          16          31
          2           4           4           8
          3          12          16          28
Avg wait time is 12.000000
Avg turn around time is 22.333334
```

(3) Aim : Write a program to simulate the producer consumer problem.

PROGRAM:

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

int mutex=1,full=0,empty=3,x=0;

int main()
{
int n;
void producer();
void consumer();
int wait(int);
int signal(int);
printf("\n1.Producer\n2.Consumer\n3.Exit");
while(1)
{
printf("\nEnter your choice:");
scanf("%d",&n);
switch(n)
{
case 1:if((mutex==1)&&(empty!=0))
producer();
else
printf("Buffer is full!!");
break;
case 2:if((mutex==1)&&(full!=0))
consumer();
else
```

```

        printf("Buffer is empty!!");
        break;
    case 3:
        exit(0);
        break;
}

}
return 0;
}
int wait(int s)
{
    return (--s);
}
int signal(int s)
{
    return(++s);
}
void producer()
{
    mutex=wait(mutex); //reduce mutex
    full=signal(full); //increase full
    empty=wait(empty); //reduce empty
    x++;
    printf("\nProducer produces the item %d",x);
    mutex=signal(mutex); //increase mutex
}
void consumer()
{
    mutex=wait(mutex);
    full=wait(full);
    empty=signal(empty);
    printf("\nConsumer consumes item %d",x);
    x--;
    mutex=signal(mutex);
}

```

OUTPUT:

```
1.Producer
2.Consumer
3.Exit
Enter your choice:1

Producer produces the item 1
Enter your choice:1

Producer produces the item 2
Enter your choice:1

Producer produces the item 3
Enter your choice:1
Buffer is full!!
Enter your choice:2

Consumer consumes item 3
Enter your choice:2

Consumer consumes item 2
Enter your choice:2

Consumer consumes item 1
Enter your choice:2
Buffer is empty!!
Enter your choice:3
```

(4) Aim: Write a program to simulate Bankers algorithm for Deadlock Avoidance PROGRAM:

```
#include<stdio.h>
#include<conio.h>
int main()
{
int clm[7][5],req[7][5],alloc[7][5],rsrc[5],avail[5],comp[7];
int first,p,r,i,j,prc,count,t;
count=0;
for(i=1;i<=7;i++)
comp[i]=0;
printf("\t BANKERS ALGORITHM IN C \n\n");
printf("Enter the no of processes : ");
scanf("%d",&p);
printf("\n\nEnter the no of resources : ");
scanf("%d",&r);
printf("\n\nEnter the claim for each process : ");
for(i=1;i<=p;i++)
{
printf("\nFor process %d : ",i);
for(j=1;j<=r;j++)
{
scanf("%d",&clm[i][j]);
}
}
}
```

```

printf("\n\nEnter the allocation for each process : ");
for(i=1;i<=p;i++)
{
printf("\nFor process %d : ",i);
for(j=1;j<=r;j++)
{
scanf("%d",&alloc[i][j]);
}
}
printf("\n\nEnter total no of each resource : ");
for(j=1;j<=r;j++)
scanf("%d",&rsrc[j]);

for(j=1;j<=r;j++)
{
int total=0;
avail[j]=0;
for(i=1;i<=p;i++)
{total+=alloc[i][j];}
avail[j]=rsrc[j]-total;
}
do
{
for(i=1;i<=p;i++)
{
for(j=1;j<=r;j++)
{
req[i][j]=clm[i][j]-alloc[i][j];
}
}
printf("\n\nAvailable resources are : ");
for(j=1;j<=r;j++)
{ printf("%d ",avail[j]); }
printf("\nClaim matrix:\tAllocation matrix:\n");
for(i=1;i<=p;i++)
{
for(j=1;j<=r;j++)
{
printf("%d\t",clm[i][j]);
}
printf("\t\t");
for(j=1;j<=r;j++)
{
printf("%d\t",alloc[i][j]);
}
}
}

```

```

    }
    printf("\n");
    }
    prc=0;
    for(i=1;i<=p;i++)
    {
        if(comp[i]==0)//if not completed
        {
            prc=i;
            for(j=1;j<=r;j++)
            {
                if(avail[j]==0)
                {
                    prc=0;
                    break;
                }
            }
        }
        if(prc!=0)
            break;
    }
    if(prc!=0)
    {
        printf("\nProcess ",prc,"runs to completion!");
        count++;
        for(j=1;j<=r;j++)
        {
            avail[j]+=alloc[prc][j];
            alloc[prc][j]=0;
            clm[prc][j]=0;
            comp[prc]=1;
        }
    }
    }
    while(count!=p &&prc!=0);
    if(count==p)
        printf("\nThe system is in a safe state!!");
    else
        printf("\nThe system is in an unsafe state!!");
    getch();
    return 0;
}

```

OUTPUT:

```
BANKERS ALGORITHM IN C

Enter the no of processes : 5

Enter the no of resources : 3

Enter the claim for each process :
For process 1 : 7 5 3

For process 2 : 3 2 2

For process 3 : 9 0 2

For process 4 : 2 2 2

For process 5 : 4 3 3
```

```
Enter the allocation for each process :
For process 1 : 0 1 0

For process 2 : 2 0 0

For process 3 : 3 0 2

For process 4 : 2 1 1

For process 5 : 0 0 2

Enter total no of each resource : 10 5 7

Available resources are : 3 3 2
Claim matrix:      Allocation matrix:
7      5      3      0      1      0
3      2      2      2      0      0
9      0      2      3      0      2
2      2      2      2      1      1
4      3      3      0      0      2

Process
Available resources are : 3 4 2
Claim matrix:      Allocation matrix:
0      0      0      0      0      0
3      2      2      2      0      0
9      0      2      3      0      2
2      2      2      2      1      1
4      3      3      0      0      2

Process
Available resources are : 5 4 2
Claim matrix:      Allocation matrix:
0      0      0      0      0      0
0      0      0      0      0      0
9      0      2      3      0      2
2      2      2      2      1      1
4      3      3      0      0      2

Process
Available resources are : 8 4 4
```



```

Claim matrix:  Allocation matrix:
0      0      0      0      0      0
0      0      0      0      0      0
0      0      0      0      0      0
2      2      2      2      1      1
4      3      3      0      0      2

Process

Available resources are : 10 5 5
Claim matrix:  Allocation matrix:
0      0      0      0      0      0
0      0      0      0      0      0
0      0      0      0      0      0
0      0      0      0      0      0
4      3      3      0      0      2

Process
The system is in a safe state!!

```

(5a) Aim : Write a program to simulate MFT (Multiprogramming Fixed Task technique)

PROGRAM:

```

#include<stdio.h>
#include<conio.h>
void main()
{
int m,p,s,p1;
int m1[4],i,f,f1=0,f2=0,fra1,fra2;
printf("Enter the memory size:");
scanf("%d",&m);
printf("Enter the no of partitions:");
scanf("%d",&p);
s=m/p;
printf("Each partn size is:%d",s);
printf("\nEnter the no of processes:");
scanf("%d",&p1);

for(i=0;i<p1;i++)
{
printf("\nEnter the memory req for process%d:",i+1);
scanf("%d",&m1[i]);
if(m1[i]<=s)
{
printf("\nProcess is allocated in partition%d",i+1);

```

```

fra1=s-m1[i];
printf("\nInternal fragmentation for process is:%d",fra1);
f1=f1+fra1;
}
else
{
printf("\nProcess not allocated in partition%d",i+1);
fra2=s;
f2=f2+fra2;
printf("\nExternal fragmentation for partition is:%d",fra2);
}
}
printf("\nProcess\tmemory\tallocatedmemory");
for(i=0;i<p1;i++)
printf("\n%5d\t%5d\t%5d",i+1,s,m1[i]);
f=f1+f2;
printf("\nThe tot no of fragmentation is:%d",f);
getch();
}

```

OUTPUT:

```

Enter the memory size:600
Enter the no of partitions:6
Each partn size is:100
Enter the no of processes:4

Enter the memory req for process1:60

Process is allocated in partition1
Internal fragmentation for process is:40
Enter the memory req for process2:40

Process is allocated in partition2
Internal fragmentation for process is:60
Enter the memory req for process3:200

Process not allocated in partition3
External fragmentation for partition is:100
Enter the memory req for process4:100

Process is allocated in partition4
Internal fragmentation for process is:0
Process memory    allocatedmemory
1      100      60
2      100      40
3      100      200
4      100      100
The tot no of fragmentation is:200

```

(5b) Aim : Write a program to simulate MVT (Multiprogramming Variable Partition technique)

PROGRAM:

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

OUTPUT:

```
enter the memory capacity:400
enter the no of processes:4

enter memory req for process1: 400
there is no further memory remaining:
external fragmentation for this process is:0
enter memory req for process2: 100
the memory allocated for process2 is: 400
remaining memory is: 300
external fragmentation for this process is:300
enter memory req for process3: 400
memory is not allocated for process3
external fragmentation for this process is:300
enter memory req for process4: 50
the memory allocated for process4 is: 300
remaining memory is: 250
external fragmentation for this process is:250
```

(6) Aim: Write a program to simulate Paging Technique of memory management

PROGRAM:

```
#include<stdio.h>
int n,nf;
int in[100];
int p[50];
int hit=0;
int i,j,k;
int pgfaultcnt=0;
void getData()
{
printf("\nEnter length of page reference sequence:");
scanf("%d",&n);
printf("\nEnter the page reference sequence:");
for(i=0; i<n; i++)
scanf("%d",&in[i]);
printf("\nEnter no of frames:");
scanf("%d",&nf);
}

void initialize()
{
pgfaultcnt=0;
for(i=0; i<nf; i++)
p[i]=9999;
}
```

```

int isHit(int data)
{
    hit=0;
    for(j=0; j<nf; j++)
    {
        if(p[j]==data)
        {
            hit=1;
            break;
        }
    }

    return hit;
}

int getHitIndex(int data)
{
    int hitind;
    for(k=0; k<nf; k++)
    {
        if(p[k]==data)
        {
            hitind=k;
            break;
        }
    }
    return hitind;
}

void dispPages()
{
    for (k=0; k<nf; k++)
    {
        if(p[k]!=9999)
        printf(" %d",p[k]);
    }
}

void dispPgFaultCnt()
{
    printf("\nTotal no of page faults:%d",pgfaultcnt);
}

```

```

void fifo()
{
    initialize();
    for(i=0; i<n; i++)
    {
        printf("\nFor %d :",in[i]);

        if(isHit(in[i])==0)
        {

            for(k=0; k<nf-1; k++)
                p[k]=p[k+1];

            p[k]=in[i];
pgfaultcnt++;
dispPages();
        }
        else
        printf("No page fault");
    }
    dispPgFaultCnt();
}

void optimal()
{
    initialize();
    int near[50];
    for(i=0; i<n; i++)
    {

        printf("\nFor %d :",in[i]);

        if(isHit(in[i])==0)
        {

            for(j=0; j<nf; j++)
            {
                int pg=p[j];
                int found=0;
                for(k=i; k<n; k++)
                {
                    if(pg==in[k])
                    {
near[j]=k;

```

```

        found=1;
        break;
    }
    else
        found=0;
    }
    if(!found)
near[j]=9999;
    }
    int max=-9999;
    int repindex;
    for(j=0; j<nf; j++)
    {
        if(near[j]>max)
        {
            max=near[j];
repindex=j;
        }
    }
    p[repindex]=in[i];
pgfaultcnt++;

dispPages();
    }
    else
printf("No page fault");
    }
dispPgFaultCnt();
}

void lru()
{
    initialize();

    int least[50];
    for(i=0; i<n; i++)
    {
printf("\nFor %d :",in[i]);
if(isHit(in[i])==0)
    {

        for(j=0; j<nf; j++)
        {
            int pg=p[j];

```

```

        int found=0;
        for(k=i-1; k>=0; k--)
        {
            if(pg==in[k])
            {
                least[j]=k;
                found=1;
                break;
            }
            else
                found=0;
        }
        if(!found)
            least[j]=-9999;
    }
    int min=9999;
    int repindex;
    for(j=0; j<nf; j++)
    {
        if(least[j]<min)
        {
            min=least[j];
            repindex=j;
        }
    }
    p[repindex]=in[i];
    pgfaultcnt++;

    dispPages();
    }
    else
    printf("No page fault!");
    }
    dispPgFaultCnt();
}

void lfu()
{
    int usedcnt[100];
    int least,repin,sofarcnt=0,bn;
    initialize();
    for(i=0; i<nf; i++)
        usedcnt[i]=0;

```



```

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

printf("\n For %d :",in[i]);
    if(isHit(in[i]))
    {
        int hitind=getHitIndex(in[i]);
usedcnt[hitind]++;
printf("No page fault!");
    }
    else
    {
pgfaultcnt++;
        if(bn<nf)
        {
            p[bn]=in[i];
usedcnt[bn]=usedcnt[bn]+1;
            bn++;
        }
        else
        {
            least=9999;
            for(k=0; k<nf; k++)
                if(usedcnt[k]<least)
                {
repin=k;
least=usedcnt[k];

        }

            p[repin]=in[i];
sofarcnt=0;
            for(k=0; k<=i; k++)
                if(in[i]==in[k])
sofarcnt=sofarcnt+1;
usedcnt[repin]=sofarcnt;
        }

dispPages();
    }

}
dispPgFaultCnt();
}

```

```

void secondchance()
{
    int usedbit[50];
    int victimptr=0;
    initialize();
    for(i=0; i<nf; i++)
usedbit[i]=0;
    for(i=0; i<n; i++)
    {
        printf("\nFor %d:",in[i]);
        if(isHit(in[i]))
        {
            printf("No page fault!");
            int hitindex=getHitIndex(in[i]);
            if(usedbit[hitindex]==0)
            usedbit[hitindex]=1;
        }
        else
        {
            pgfaultcnt++;
            if(usedbit[victimptr]==1)
            {
                do
                {
                    usedbit[victimptr]=0;
                    victimptr++;
                    if(victimptr==nf)
                    victimptr=0;
                }
                while(usedbit[victimptr]!=0);
            }
            if(usedbit[victimptr]==0)
            {
                p[victimptr]=in[i];
                usedbit[victimptr]=1;
                victimptr++;
            }
            dispPages();

        }
        if(victimptr==nf)
        victimptr=0;
    }
    dispPgFaultCnt();
}

```

```

}

int main()
{
    int choice;
    while(1)
    {
        printf("\nPage Replacement Algorithms\n1.Enter
data\n2.FIFO\n3.Optimal\n4.LRU\n5.LFU\n6.Second Chance\n7.Exit\nEnter your choice:");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1:
                getData();
                break;
            case 2:
                fifo();
                break;
            case 3:
                optimal();
                break;
            case 4:
                lru();
                break;
            case 5:
                lfu();
                break;
            case 6:
                secondchance();
                break;
            default:
                return 0;
                break;
        }
    }
}

```

OUTPUT:

```
Page Replacement Algorithms
1.Enter data
2.FIFO
3.Optimal
4.LRU
5.LFU
6.Second Chance
7.Exit
Enter your choice:1

Enter length of page reference sequence:12

Enter the page reference sequence:1 2 3 4 1 2 5 1 2 3 4 5

Enter no of frames:3

Page Replacement Algorithms
1.Enter data
2.FIFO
3.Optimal
4.LRU
5.LFU
6.Second Chance
7.Exit
Enter your choice:2
```

```
Page Replacement Algorithms
1.Enter data
2.FIFO
3.Optimal
4.LRU
5.LFU
6.Second Chance
7.Exit
Enter your choice:2

For 1 : 1
For 2 : 1 2
For 3 : 1 2 3
For 4 : 2 3 4
For 1 : 3 4 1
For 2 : 4 1 2
For 5 : 1 2 5
For 1 :No page fault
For 2 :No page fault
For 3 : 2 5 3
For 4 : 5 3 4
For 5 :No page fault
Total no of page faults:9
Page Replacement Algorithms
1.Enter data
2.FIFO
3.Optimal
4.LRU
5.LFU
6.Second Chance
```

(7a) Aim: Write a program to simulate FIFO page replacement algorithm

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
int i,j,nof,nor,flag=0,ref[50],frm[50],pf=0,victim=-1;
void main()
{
printf("\n\t\t\t FIFO PAGE REPLACEMENT ALGORITHM");
printf("\n Enter no. of frames:");
scanf("%d",&nof);
printf("Enter number of Pages:\n");
scanf("%d",&nor);
printf("\n Enter the Page No:");
for(i=0;i<nor;i++)
scanf("%d",&ref[i]);
printf("\nThe given Pages are:");
for(i=0;i<nor;i++)
printf("%4d",ref[i]);
for(i=1;i<=nof;i++)
frm[i]=-1;
printf("\n");
for(i=0;i<nor;i++)
{
flag=0;
printf("\n\t page no %d->\t",ref[i]);
for(j=0;j<nof;j++)
{
if(frm[j]==ref[i])
{
flag=1;
break;
}
}
if(flag==0)
{
pf++;
victim++;
victim=victim%nof;
frm[victim]=ref[i];
for(j=0;j<nof;j++)
printf("%4d",frm[j]);
}
}
printf("\n\n\t\t No.of pages faults...%d",pf);
```

```

getch();
}

```

OUTPUT:

```

Enter no. of frames:3
Enter number of Pages:
20

Enter the Page No:7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

The given Pages are:  7  0  1  2  0  3  0  4  2  3  0  3  2  1  2  0  1  7  0  1

page no 7->          7  -1  -1
page no 0->          7  0  -1
page no 1->          7  0  1
page no 2->          2  0  1
page no 0->
page no 3->          2  3  1
page no 0->          2  3  0
page no 4->          4  3  0
page no 2->          4  2  0
page no 3->          4  2  3
page no 0->          0  2  3
page no 3->
page no 2->
page no 1->          0  1  3
page no 2->          0  1  2
page no 0->
page no 1->
page no 7->          7  1  2
page no 0->          7  0  2
page no 1->          7  0  1

No.of pages faults...15

```

(7b) Aim: Write a program to simulate LRU page replacement algorithm

PROGRAM:

```

#include<stdio.h>
#include<conio.h>
int i,j,nof,nor,flag=0,ref[50],frm[50],pf=0,victim=-1;
int recent[10],lrucal[50],count=0;
int lruvictm();
void main()
{
printf("\n\t\t\t LRU PAGE REPLACEMENT ALGORITHM");
printf("\n Enter no. of Frames:");
scanf("%d",&nof);
printf(" Enter no. of reference string:");
scanf("%d",&nor);
printf("\n Enter reference string:");
for(i=0;i<nor;i++)
scanf("%d",&ref[i]);
printf("\n\t The given reference string is:");
for(i=0;i<nor;i++)
printf("%4d",ref[i]);
for(i=1;i<=nof;i++)
{

```

```

frm[i]=-1;
lrucal[i]=0;
}
for(i=0;i<10;i++)
recent[i]=0;
printf("\n");
for(i=0;i<nor;i++)
{
flag=0;
printf("\n\t Reference NO %d->\t",ref[i]);
for(j=0;j<nof;j++)
{
if(frm[j]==ref[i])
{
flag=1;
break;
}
}
if(flag==0)
{
count++;
if(count<=nof)
victim++;
else
victim=lruvictim();
pf++;
frm[victim]=ref[i];
for(j=0;j<nof;j++)
printf("%4d",frm[j]);
}
recent[ref[i]]=i;
}
printf("\n\n\t No. of page faults:%d",pf);
getch();
}
int lruvictim()
{
int i,j,temp1,temp2;
for(i=0;i<nof;i++)
{
temp1=frm[i];
lrucal[i]=recent[temp1];
}
temp2=lrucal[0];

```

```

for(j=1;j<nof;j++)
{
if(temp2>lrucal[j])
temp2=lrucal[j];
}
for(i=0;i<nof;i++)
if(ref[temp2]==frm[i])
return i;
return 0;}

```

OUTPUT:

```

LRU PAGE REPLACEMENT ALGORITHM
Enter no. of Frames:3
Enter no. of reference string:20

Enter reference string:7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

The given reference string is:  7  0  1  2  0  3  0  4  2  3  0  3  2  1  2  0  1  7  0  1

Reference NO 7->      7  -1  -1
Reference NO 0->      7   0  -1
Reference NO 1->      7   0   1
Reference NO 2->      2   0   1
Reference NO 0->
Reference NO 3->      2   0   3
Reference NO 0->
Reference NO 4->      4   0   3
Reference NO 2->      4   0   2
Reference NO 3->      4   3   2
Reference NO 0->      0   3   2
Reference NO 3->
Reference NO 2->
Reference NO 1->      1   3   2
Reference NO 2->
Reference NO 0->      1   0   2
Reference NO 1->
Reference NO 7->      1   0   7
Reference NO 0->
Reference NO 1->

No. of page faults:12

```

(7c) Aim: Write a program to simulate optimal page replacement algorithm

PROGRAM:

```

#include<stdio.h>
#include<conio.h>
int i,j,nof,nor,flag=0,ref[50],frm[50],pf=0,victim=-1;
int recent[10],optcal[50],count=0;
int optvictim();
void main()
{
printf("\n OPTIMAL PAGE REPLACEMENT ALGORITHM");
printf("\nEnter the no. of frames:");
scanf("%d",&nof);
printf("Enter the no. of reference string:");
scanf("%d",&nor);
printf("Enter the reference string:");

```



```

for(i=0;i<nor;i++)
scanf("%d",&ref[i]);
printf("\nThe given string is:");
for(i=0;i<nor;i++)
printf("%4d",ref[i]);
for(i=0;i<nof;i++)
{
frm[i]=-1;
optcal[i]=0;
}
for(i=0;i<10;i++)
recent[i]=0;
printf("\n");
for(i=0;i<nor;i++)
{
flag=0;
printf("\n\tref no %d ->\t",ref[i]);
for(j=0;j<nof;j++)
{
if(frm[j]==ref[i])
{
flag=1;
break;
}
}
if(flag==0)
{
count++;
if(count<=nof)
victim++;
else
victim=optvictim(i);
pf++;
frm[victim]=ref[i];
for(j=0;j<nof;j++)
printf("%4d",frm[j]);
}
}
printf("\n Number of page faults: %d",pf);
getch();
}
int optvictim(int index)
{
int i,j,temp,notfound;

```

```

for(i=0;i<nof;i++)
{
notfound=1;
for(j=index;j<nor;j++)
if(frm[i]==ref[j])
{
notfound=0;
optcal[i]=j;
break;
}
if(notfound==1)
return i;
}
temp=optcal[0];
for(i=1;i<nof;i++)
if(temp<optcal[i])
temp=optcal[i];
for(i=0;i<nof;i++)
if(frm[temp]==frm[i])
return i;
return 0;
}

```

OUTPUT:

```

OPTIMAL PAGE REPLACEMENT ALGORITHM
Enter the no. of frames:3
Enter the no. of reference string:20
Enter the reference string:7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
The given string is:  7  0  1  2  0  3  0  4  2  3  0  3  2  1  2  0  1  7  0  1

ref no 7 ->      7  -1  -1
ref no 0 ->      7   0  -1
ref no 1 ->      7   0   1
ref no 2 ->      7   2   1
ref no 0 ->      0   2   1
ref no 3 ->      3   2   1
ref no 0 ->      0   2   1
ref no 4 ->      4   2   1
ref no 2 ->
ref no 3 ->      3   2   1
ref no 0 ->      0   2   1
ref no 3 ->      3   2   1
ref no 2 ->
ref no 1 ->
ref no 2 ->
ref no 0 ->      0   2   1
ref no 1 ->
ref no 7 ->      0   7   1
ref no 0 ->
ref no 1 ->
Number of page faults: 13

```

(8a) Aim: Write a program to simulate FCFS disk scheduling algorithm.

PROGRAM:

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

```

#include<conio.h>
void main()
{
    int queue[100],n,head,i,j,k,seek=0,diff;
    float avg;
    // clrscr();
    printf("*** FCFS Disk Scheduling Algorithm ***\n");
    printf("Enter the size of Queue\t");
    scanf("%d",&n);
    printf("Enter the Queue\t");
    for(i=1;i<=n;i++)
    {
        scanf("%d",&queue[i]);
    }
    printf("Enter the initial head position\t");
    scanf("%d",&head);
    queue[0]=head;
    printf("\n");
    for(j=0;j<=n-1;j++)
    {
        diff=abs(queue[j+1]-queue[j]);
        seek+=diff;
        printf("Move from %d to %d with Seek %d\n",queue[j],queue[j+1],diff);
    }
    printf("\nTotal Seek Time is %d\t",seek);
    avg=seek/(float)n;
    printf("\nAverage Seek Time is %f\t",avg);
    getch();
}

```

OUTPUT:

```

*** FCFS Disk Scheduling Algorithm ***
Enter the size of Queue 8
Enter the Queue 98 183 37 122 14 124 65 67
Enter the initial head position 53

Move from 53 to 98 with Seek 45
Move from 98 to 183 with Seek 85
Move from 183 to 37 with Seek 146
Move from 37 to 122 with Seek 85
Move from 122 to 14 with Seek 108
Move from 14 to 124 with Seek 110
Move from 124 to 65 with Seek 59
Move from 65 to 67 with Seek 2

Total Seek Time is 640
Average Seek Time is 80.000000

```

(8b) Aim: Write a program to simulate SSTF disk scheduling algorithm.

PROGRAM:

```
/*
    SSTF Disk Scheduling Algorithm
    Created By: Pirate
*/
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main()
{
    int queue[100],t[100],head,seek=0,n,i,j,temp;
    float avg;
    // clrscr();
    printf("*** SSTF Disk Scheduling Algorithm ***\n");
    printf("Enter the size of Queue\t");
    scanf("%d",&n);
    printf("Enter the Queue\t");
    for(i=0;i<n;i++)
    {
        scanf("%d",&queue[i]);
    }
    printf("Enter the initial head position\t");
    scanf("%d",&head);
    for(i=1;i<n;i++)
        t[i]=abs(head-queue[i]);
    for(i=0;i<n;i++)
    {
        for(j=i+1;j<n;j++)
        {
            if(t[i]>t[j])
            {
                temp=t[i];
                t[i]=t[j];
                t[j]=temp;
                temp=queue[i];
                queue[i]=queue[j];
                queue[j]=temp;
            }
        }
    }
    for(i=1;i<n-1;i++)
    {
        seek=seek+abs(head-queue[i]);
    }
}
```

```

        head=queue[i];
    }
    printf("\nTotal Seek Time is%d\t",seek);
    avg=seek/(float)n;
    printf("\nAverage Seek Time is %f\t",avg);
    getch();
}

```

OUTPUT:

```

*** SSTF Disk Scheduling Algorithm ***
Enter the size of Queue 8
Enter the Queue 98 183 37 122 14 124 65 67
Enter the initial head position 53

Total Seek Time is177
Average Seek Time is 22.125000

```

(9a) Aim: Write a program to simulate SCAN disk scheduling algorithm.

PROGRAM:

```

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

int main()
{
    int i,j,sum=0,n;
    int d[20];
    int disk; //loc of head
    int temp,max;
    int dloc; //loc of disk in array
    printf("enter number of location\t");
    scanf("%d",&n);
    printf("enter position of head\t");
    scanf("%d",&disk);
    printf("enter elements of disk queue\n");
    for(i=0;i<n;i++)
    {
        scanf("%d",&d[i]);
    }
}

```

```

d[n]=disk;
n=n+1;
for(i=0;i<n;i++) // sorting disk locations
{
    for(j=i;j<n;j++)
    {
        if(d[i]>d[j])
        {
            temp=d[i];
            d[i]=d[j];
            d[j]=temp;
        }
    }
}
max=d[n];
for(i=0;i<n;i++) // to find loc of disc in array
{
    if(disk==d[i]) { dloc=i; break; }
}
for(i=dloc;i>=0;i--)
{
    printf("%d -->",d[i]);
}
printf("0 -->");
for(i=dloc+1;i<n;i++)
{
    printf("%d-->",d[i]);
}
sum=disk+max;
printf("\nmovement of total cylinders %d",sum);
getch();
return 0;
}

```

OUTPUT:

```

enter number of location      8
enter position of head  53
enter elements of disk queue
37 98 183 124 14 67 122 67
53 -->37 -->14 -->0 -->67-->67-->98-->122-->124-->183-->
movement of total cylinders 53

```

(10a) Aim: Write a program to simulate sequential file allocation strategies.

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
#include<stdlib.h>
#define BL 30
#define BW 15
#define SX 150
#define SY 100
typedef struct
{
    char name[10];
    int st_blk;
    int len,fcolor;
}file_info;
int cols[]={ GREEN,MAGENTA,BLUE,CYAN,RED};
file_info F[10];
int n;
void read_files()
{
    int i;
    printf(" HOW MANY FILES U WANT");
    scanf("%d",&n);

    printf("ENTER THE DETAILS");
    for(i=0;i<n;i++)
    {
        printf("\n FILE NAME IS: ");
        fflush(stdin);
        gets(F[i].name);
        printf("\n STARTING BLOCK:");
        scanf("%d",&F[i].st_blk);
        printf("\n LENGTH: ");
        scanf("%d",&F[i].len);
        F[i].fcolor=cols[i];
    }
}
void fill_space()
{
    int i,b,x1,y1;
    for(i=0;i<n;i++)
    {
        for(b=F[i].st_blk;b<F[i].st_blk+F[i].len;b++)
```

```

{
setfillstyle(1,F[i].fcolor);
x1= SX+(b%7)*50;
y1= SY+(b/7)*40;
bar3d(x1,y1,x1+BL,y1+BW,0,0);
}
}
}
void design_disk()
{
int i,j,x1,y1;
char s[5];
/* title*/
setcolor(LIGHTGREEN);
settextstyle(1,0,3);
settextjustify(1,1);
outtextxy(320,20,"CONTINUOUS FILE ALLOCATION");
setcolor(14);
settextstyle(2,0,3);
settextjustify(1,1);

for(i=0;i<7;i++)
for(j=0;j<7;j++)
{
x1= SX+j*50;
y1= SY+i*40;
rectangle(x1,y1,x1+BL,y1+BW);
sprintf(s,"%d",i*7+j);
outtextxy(x1+BL/2,y1+BW+10,s);
}

/*create out line */
rectangle(SX-20,SY-20,SX+350,SY+280);
settextstyle(1,0,2);
for(i=0;i<n;i++)
{
setcolor(F[i].fcolor);
outtextxy(100,400+i*20,F[i].name);
}
setcolor(14);
}

void main()
{

```



```

int gd, gm;
gd=DETECT;
initgraph(&gd, &gm, "..\\bgi");
randomize();
read_files();
cleardevice();
design_disk();
fill_space();
getch();
closegraph();
}

```

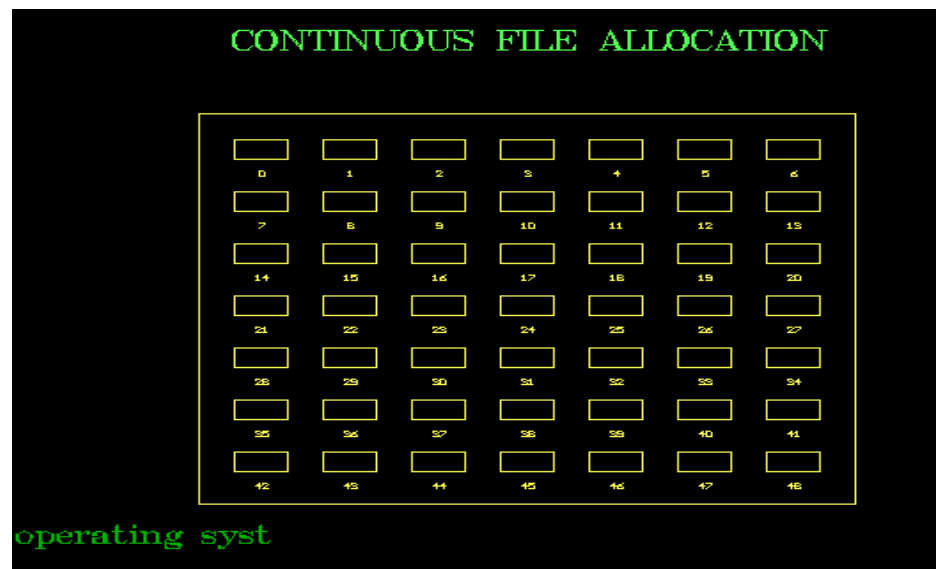
OUTPUT:

```

how many files do you want1
enter the details
FILE NAME IS:os

STARTING BLOCK:operating system

```



(10b) Aim: Write a program to simulate the indexed file allocation strategies

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
#include<stdlib.h>
#define BL 30
#define BW 15
#define SX 140
#define SY 100

typedef struct
{
int x1,y1,x2,y2,row,col;
}block;

typedef struct
{
char name[10];
int st_blk;
int len,fcolor;
}file_info;

void draw_link_lines(int st,int end);
file_info f={"ABC.txt",10,7,RED};
int blocks_allocated[7]={30,2,18,27,44,61};
block b[60];

void fill_space()
{
int i,n;
setfillstyle(1,LIGHTMAGENTA);
n=blocks_allocated[0];
bar3d(b[n].x1,b[n].y1,b[n].x2,b[n].y2,0,0);
setfillstyle(1,f.fcolor);
setcolor(LIGHTCYAN);
circle(b[n].x1+15,b[n].y1+7,18);

for(i=1;i<6;i++)
{
n=blocks_allocated[i];
setcolor(14);
bar3d(b[n].x1,b[n].y1,b[n].x2,b[n].y2,0,0);
draw_link_lines(blocks_allocated[0],blocks_allocated[i]);
sleep(1);
```

```

}
}

/* assings attributes for disk blocks */
void create_block(int bno,block *bp)
{
bp->row=bno/7;
bp->col=bno%7;
bp->x1= SX+(bno%7)*50;
bp->y1= SY+(bno/7)*40;
bp->x2=bp->x1+BL;
bp->y2=bp->y1+BW;
}

/* draws disk structure */

void design_disk()
{
int i,j,x1,y1;
char s[5];
/* title*/
setcolor(LIGHTGREEN);
settextstyle(1,0,3);
settextjustify(1,1);
outtextxy(320,20,"INDEXED FILE ALLOCATION");
setcolor(14);
settextstyle(2,0,3);
settextjustify(1,1);
for(i=0;i<63;i++)
{
create_block(i,&b[i]);
rectangle(b[i].x1,b[i].y1,b[i].x2,b[i].y2);
sprintf(s,"%d",i);
outtextxy(b[i].x1+BL/2,b[i].y1+BW+10,s);
}

/*create out line */
rectangle(SX-20,SY-20,SX+350,SY+360);
settextstyle(1,0,2);
setcolor(14);
}

/* drwas link lines between aloocated b locks and indexed blocks */

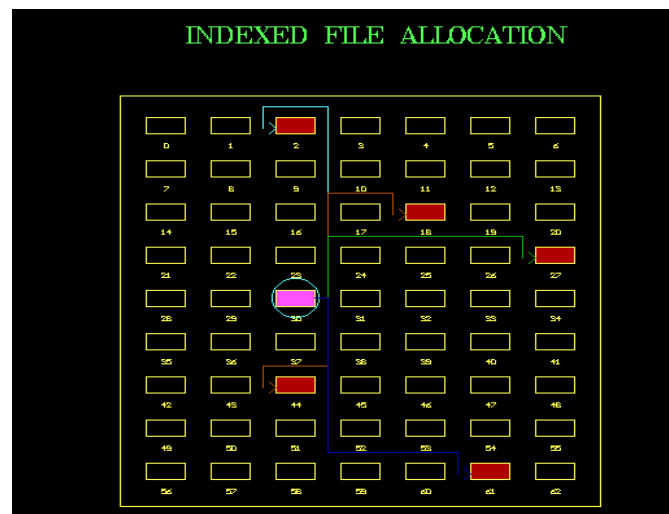
```

```

void draw_link_lines(int st,int end)
{
int row,col,x;
while((x=random(16))==0)
continue;
setcolor(x);
moveto(b[st].x2,b[st].y1+7);
linereel(10,0);
lineto(getx(),b[end].y1-10);
lineto(b[end].x1-10,gety());
lineto(getx(),b[end].y1+10);
line(b[end].x1,b[end].y1+10,b[end].x1-5,b[end].y1+5);
line(b[end].x1,b[end].y1+10,b[end].x1-5,b[end].y1+15);
}
void main()
{
int gd,gm;
gd=DETECT;
initgraph(&gd,&gm,"..\\bgi");
randomize();
cleardevice();
sleep(2);
design_disk();
fill_space();
getch();
closegraph();
}

```

OUTPUT:



(10c) Aim: Write a program to simulate the linked file allocation strategies.

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
#include<stdlib.h>
#define BL 30
#define BW 15
#define SX 140
#define SY 100
typedef struct
{
int x1,y1,x2,y2,row,col;
}block;

typedef struct
{
char name[10];
int st_blk;
int len,fcolor;
}file_info;
void draw_link_lines(int st,int end);
file_info f={"ABC.txt",10,7,RED};
int blocks_allocated[60]={28,16,31,33,38,55,56,29,1,30,48,12,19,20,21,22,39,40,
18,2,1,50,51,52,54,58,17,35,43,13,3,36,49,24,6,4,14,16,
25,37,15,5,56,11,26,9,27,45,55,23,34,41,42,59,8,53};
block b[60];
void read_file_info()
{
printf("\n ENTER FILE ANME");
fflush(stdin);
gets(f.name);
printf("\n LENGTH");
scanf("%d",&f.len);
}

/* draws a table with file info */
void create_info_table()
{
int i;
char *s;
settextstyle(2,0,4);
setcolor(GREEN);
rectangle(500,200,639,300);
```

```

line(580,200,580,300);
line(500,200,580,300);
line(500,260,639,260);
settextjustify(0,1);

outtextxy(505,220,"FILE NAME");
outtextxy(585,220,f.name);

outtextxy(505,250,"ST BLOCK");
sprintf(s,"%d",f.st_blk);
outtextxy(595,250,s);

outtextxy(505,280,"LENGTH");
sprintf(s,"%d",f.len);
outtextxy(595,280,s);
}

void fill_space()
{
int i,n;
for(i=0;i<f.len;i++)
{
n=blocks_allocated[i];
setfillstyle(1,i==0?RED:f.fcolor);

bar3d(b[n].x1,b[n].y1,b[n].x2,b[n].y2,0,0);
if(i<f.len-1)
draw_link_lines(blocks_allocated[i],blocks_allocated[i+1]);
sleep(1);
}
}

/* assings attributes for a disk block */
void create_block(int bno,block *bp)
{
bp->row=bno/7;
bp->col=bno%7;
bp->x1= SX+(bno%7)*50;
bp->y1= SY+(bno/7)*40;
bp->x2=bp->x1+BL;
bp->y2=bp->y1+BW;
}

```

```

/* draws disk structure */
void design_disk()
{
    int i,j,x1,y1;
    char s[5];
    /* title*/
    setcolor(LIGHTGREEN);
    settextstyle(1,0,3);
    settextjustify(1,1);
    outtextxy(320,20,"LINKED FILE ALLOCATION");

    setcolor(14);
    settextstyle(2,0,3);
    settextjustify(1,1);

    for(i=0;i<63;i++)
    {
        create_block(i,&b[i]);
        rectangle(b[i].x1,b[i].y1,b[i].x2,b[i].y2);

        sprintf(s,"%d",i);
        outtextxy(b[i].x1+BL/2,b[i].y1+BW+10,s);
    }

    /*create out line */
    rectangle(SX-20,SY-20,SX+350,SY+280);
    settextstyle(1,0,2);
    setcolor(14);
}

/* drwas link lines between aloocated b locks and indexed blocks */
void draw_link_lines(int st,int end)
{
    int row,col,x;
    while((x=random(16))==0)
        continue;
    setcolor(x);

    moveto(b[st].x2,b[st].y1+7);
    linerel(10,0);
    lineto(getx(),b[end].y1-10);
    lineto(b[end].x1-10,gety());
    lineto(getx(),b[end].y1+10);
}

```

```

line(b[end].x1,b[end].y1+10,b[end].x1-5,b[end].y1+5);
line(b[end].x1,b[end].y1+10,b[end].x1-5,b[end].y1+15);
}

```

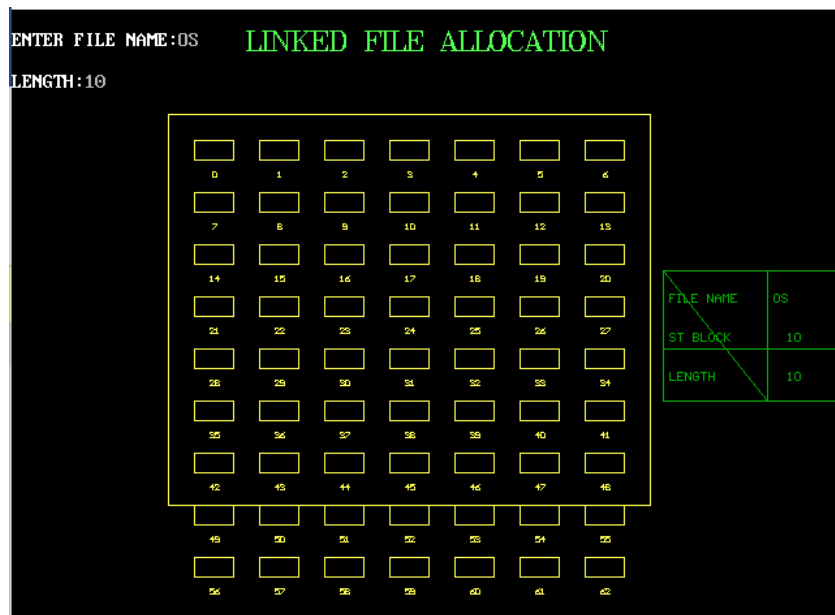
```

void main()
{
int gd,gm;
gd=DETECT;
initgraph(&gd,&gm,"..\\bgi");
cleardevice();

sleep(2);
read_file_info();
design_disk();
fill_space();
create_info_table();
getch();
closegraph();
}

```

OUTPUT:

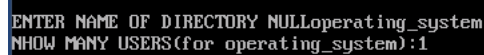


(11a) Aim: Write a program to simulate single level directory file organisation technique.

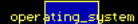
PROGRAM:

```
#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 NAME OF DIRECTORY NULLoperating_system
HOW MANY USERS(for operating_system):1
```



```
operating_system
```

(12a) Aim: Write a program to simulate hierarchical file organisation technique.

PROGRAM:

```
#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,intlev,char *dname,intlx,intrx,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 DIRECTORY root operating_syatem
enter 1 for dir/2 for file1
no.of sub directories/files(for operating_syatem):1

ENTER NAME OF DIRECTORY operating_syatem files
enter 1 for dir/2 for file

```