MALLA REDDY ENGINEERING COLLEGE (Autonomous)

Maisammaguda, Dhulapally (post & via Kompally), Secunderabad-500 100.

Department of Computer Science and Engineering

(Cyber Security)

OPERATING SYSTEM LABMANUAL

SUBJECT CODE: A0514

II Year B.Tech CSE - II SEMESTER (MR 20)

ACADAMIC YEAR: 2021 - 2022

INDEX

1. FCFS

AIM: - Program to implement first come first serve job scheduling algorithm

Description:- Consider a set of processes that arrive at time 0, with the length of the CPU burst given in milliseconds

Process	Burst time
P1	24
P2	3
P3	3

Process arrive in order p1, p2, p3 and served in FCFS order, we get the result shown in following Gantt chart

The waiting time is 0ms for process p1, 24ms for p2 and 27ms for p3.

The avg. waiting time is $\frac{0+24+27}{3}$ = **17ms**

SOURCE CODE

// * First come First Serve *//

```
#include<stdio.h>

#include<conio.h>

void main()
```

```
{
        int i,n,wt[5],st[5],tat[5],twt=0;
        float awt;
        clrscr();
        printf("enter the no.of process :");
        scanf("%d",&n);
        for(i=1;i<=n;i++)
        {
                printf("enter the service time of process %d: \n",i);
                scanf("%d", &st[i]);
        }
        wt[i]=0;
        printf("waiting time of processor 1 is : 0 \n");
        for(i=2;i<=n;i++)
        {
                 wt[i]=wt[i-1]+st[i-1];
                 printf("waiting time of processor %d : is %d \n",i,wt[i]);
        }
        for(i=1;i<=n;i++)
        {
                 twt=twt+wt[i];
                 awt=twt/n;
```

```
        }
        printf("total waiting time is :%d \n",twt);
        printf("avg waiting time is : %f\n",awt);
        for(i=1;i<=n;i++)
        {
                tat[i]=wt[i]+st[i];
            printf("turnaround time of processor %d is :%d
\n",i,tat[i]);
        }
        getch();
```

OUTPUT

Enter the no. of process: 3

Enter service time of processor 1: 24

Enter service time of processor 2: 3

Enter service time of processor 3: 3

Waiting time of processor 1 is: 0

Waiting time of processor 1 is: 24

Waiting time of processor 1 is: 27

Total waiting time is: 51

Avg waiting time is: 17

Turnaround time of processor 1 is: 24

Turnaround time of processor 2 is: 27

Turnaround time of processor 3 is: 30

*** ----------****----------***

2. SJF

AIM: - Program to implement Shortest Job first scheduling algorithm

Description: - Consider the following set of processes with the length of the CPU burst given in milliseconds

Process	Burst time
P1	6
P2	8
P3	7
P4	3

Using SJF scheduling we would schedule these processes according to the following Gantt chart.

P4	**P1**	**P3**	**P2**
0	3	9	16 24

The average waiting time is $\frac{3+16+9+0}{4}$ = **7ms**

SOURCE CODE

// * Shortest Job first *//

```
#include<stdio.h>
#include<conio.h>
#include<process.h>
void main()
{
        char p[10][5],temp[5];
        int tot=0,wt[10],pt[10],i,j,n,temp1;
        float avg;
        clrscr();
        printf("enter no.of
        processors:"); scanf("%s",&n);
        for(i=0;i<n;i++)
        {
                printf("Enter the process %d name :" , i++);
                scanf("%s",&p[i]);
                printf("Enter processor time: ");
                scanf("%d",&pt[i]);
        }
        for(i=0;i<n-1;i++)
        {
```

```
        for(j=i+1;j<n;j++)
        {
                if(pt[i]>pt[j])
                {
                        temp1=pt[i];
                        pt[i]=pt[j];
                        pt[j]=temp1;
                        strcpy(temp,p[i]);
                        strcpy(p[i],temp);
                }
        }
}
wt[0]=0;
for(i=1;i<n;i++)
{
        wt[i]=wt[i-1]+pt[i-1];
        tot=tot+w[t];
}
avg=(float)tot/n;
printf("p-name\t p-name \t w-name \n");
for(i=o;i<n;i++)
        printf("%s \t %d \t %d \n",p[i],pt[i],wt[i]);
```

```
    printf("Total waiting time = %d \n avg waiting time =
%f",tot,avg);

        getch();

}
```

OUTPUT

Enter the no. of process: 3

Enter the process 1 name: p1

Enter the process time: 6

Enter the process 2 name: p2

Enter the process time: 8

Enter the process 3 name: p3

Enter the process time: 7

Enter the process 4 name: p4

Enter the process time: 3

P-name	P-time	W-time
P4	3	0
P1	6	3
P3	7	9
P2	8	16

Total waiting time=28

Avg waiting time=7.000

*** ----------****----------***

3. ROUND ROBIN

AIM: - Program to implement Round Robin job scheduling algorithm

Description: - The avg waiting time under RR policy is often long. Consider the following set of processes that arrive at time 0, with
the length of the CPU burst given in milliseconds

Process	Burst time
P1	24
P2	3
P3	3

Using Round Robin scheduling we would schedule these processes according to the following Gantt chart.

P1	**P2**	**P3**	**P1**	**P1**	**P1**	**P1**	**P1**
0	4	7	10	14	18	22	2630

P1 waits for 6ms, p2 waits for 4ms, p3 waits for 7ms.

The average waiting time is **~~6+4+7~~ = 17ms = 5.66ms**

3

SOURCE CODE

// * Round Robin * //

```
#include<stdio.h>

#include<conio.h>

#include<process.h>
```

```
#include<string.h>
void main()
{
        char p[10][5];
    int
et[10],wt[10],timer=3,count,pt[10],rt,i,j,totwat=0,n=0,found=0,m;
        float avgwt;
        clrscr();
        printf("enetr no.of processess:");
        scanf("%d",&n);
        for(i=0;i<n;i++)
        {
                printf("enter the process name:");
                scanf("%s",&p[i]);
                printf("enter the processing
                time:"); scanf("%d", &pt[i]);
        }
        m=n;
        wt[0]=0;
        i=0;
        do
        {
                if(pt[i]>timer)
```

```
        {
                rt=pt[i]-timer;
                strcpy(p[n],p[i]);
                pt[n]=rt;
                et[i]=timer;
                n++;
        }
        else
        {
                et[i]=pt[i];
        }
        i++; wt[i]=wt[i-
        1]+et[i-1];
}
while(i<n);
count=0;
for(i=0;i<m;i++)
{
        for(j=i+1;j<=n;j++)
        {
                if(strcmp(p[i],p[j])==0)
                {
```

```
                        count++;
                        found=j;
                }
        }
        if(found!=0)
        {
                wt[i]=wt[found]-(count * timer);
                count=0;
                found=0;
        }
}
for(i=0;i<m;i++)
{
        totwt=wt[i];
}
avgwt=(float)totwt/m;
printf("process NAME \t BURST TIME \t WAITING TIME
\t"); for(i=0;i<m;i++)
{
        printf("\n %s \t %d \t %d",p[i],pt[i],wt[i]);
}
printf("\n total waiting time %d \n", totwt);
```

```
        printf("total avgtime %f", avgwt);
}
```

OUTPUT

Enter the no. of process: 3

Enter the process name: p1

Enter the processing time: 24

Enter process name : p2

Enter process time : 3

Enter process name: p3

Enter process time: 3

PROCESS NAME	BURST TIME	WAITING TIME
P1	24	6
P2	3	4
P3	3	7

Total waiting time: 17

Total average time: 5.666667

*** ----------****----------***

4. CPU Priority scheduling

AIM: - Program to implement CPU priority scheduling algorithm.

Description: - SJF algorithm is a special case of the priority scheduling algorithm. We have scheduling in terms of high priority and low priority.

Set of process assumed to have arrived at time 0 in order p1, p2 -------- p5 with the length of the CPU burst given in ms.

Process	Burst time	priority
P1	10	3
P2	1	1
P3	2	4
P4	1	5
P5	5	2

P2	P5	P1	P3	P4
0	1	6	16	18 19

Average waiting time is 8.2 ms

SOURCE CODE

```
// * CPU Priority Scheduling * //

#include<stdio.h>

#include<conio.h>

void main()

{

    char p[10][5],temp[5];
```

```
int i,j,pt[10],wt[10],totwt=0,pr[10],temp1,n;
float avgwt;
clrscr();
printf("enter no.of processes:");
scanf("%d",&n); for(i=0;i<n;i++)

{
        printf("enter process%d name : ",i+1);
        scanf("%s",&p[i]);
        printf("enter process time:");
        scanf("%d", &pt[i]);
        printf("enter priority");
        scanf("%d", &pr[i]);
}
for(i=0;i<n-1;i++)
{
        for(j=i+1;j<n;j++)
        {
                if(pr[i] > pr[j])
                {
                        temp1=pr[i];
                        pr[i]=pr[j];
```

```
                        pr[j]=temp1;
                        temp1=pt[i];
                        pt[i]=pt[j];
                        pt[j]=temp1;
                        strcpy(temp,p[i]);
                        strcpy(p[i],p[j]);
                        strcpy(p[j],temp);
                    }
                }
        }
        wt[0]=0;
        for(i=1;i<n;i++)
        {
                wt[i]=wt[i-1]+pt[i-1];
                totwt=totwt+wt[i];
        }
        avgwt=(float)totwt/n;
        printf("%s \t %d \t %d \t %d \t %d \n", p[i],pt[i],pr[i],wt[i]);
}
printf("total waiting time= %d \n avg waiting time= %f",tot,avg);
getch();
}
```

OUTPUT

Enter no. of process: 5

Enter process 1 name: p1

Enter process time: 9

Enter priority : 7

Enter process 2 name: p2

Enter process time: 0

Enter priority: 1

Enter process 3 name: p3

Enter process time: 2

Enter priority : 2

Enter process 4 name: p4

Enter process time: 1

Enter priority : 2

P-name	P-time	Priority	W-time
P2	0	1	0
P3	2	2	0
P4	1	2	2
P1	9	7	3

Total wait time = 5

Avg waiting time = 1.25

*** ----------****----------***

5. FIRST FIT

AIM: - Program to implement First-Fit algorithm.

Description: - Allocate the first hole that is big enough searching can start either at the beginning of the set of holes or at the location where the previous first-fit search ended. We can stop searching as soon as we find a free hole that is large enough.

SOURCE CODE

```
// * First- Fit * //

#include<stdio.h>

#include<conio.h>

void main()

{

    int m[5],n,mr[10],i,j,t,rm=0,pr[5]={1,2,3,4,5},q;

    char p[10][10];

    clrscr();

    printf("enter the memory for partition %d \t ",

    i+1); scanf("%d",&m[i]);

}

printf("enter the no.of process\n");

scanf("%d", &n);

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

```
{
        printf("enter process name \t");
        scanf("%s", &p[i]);
        printf("enetr memory required \t");
        scanf("%d",&mr[i]);
}
for(i=0;i<n;i++)
{
           for(j=0;j<q;j++)
           if(m[j]>mr[i])
          {
        printf("proces %s has been allocated partition %d with
memory %d kb \n", p[i],m[j],pr[j]);
                m[j]=m[j]=mr[i];
                break;
            }
}
for(i=0;i<q;i++)
rm=rm+m[i];
printf("Remaining free space is %d kb \n ",rm);
getch();
}
```

OUTPUT

Enter the no. of memory partitions : 5

Enter the memory for partition 1 : 100

Enter the memory for partition 2 : 500

Enter the memory for partition 3 : 200

Enter the memory for partition 4 : 300

Enter the memory for partition 5 :

600 Enter the number of process: 4

Enter process name : p1

Enter memory required : 212

Enter process name : p2

Enter memory required : 417

Enter process name : p3

Enter memory required : 112

Enter process name : p4

Enter memory required : 426

Process p1 has been allocated partition 2 with memory 500kb.

Process p2 has been allocated partition 5 with memory 600kb.

Process p3 has been allocated partition 2 with memory 288kb.

Remaining free space is 959kb.

*** ----------****----------***

6. BEST FIT

AIM: - Program to implement Best-Fit algorithm.

Description: - Allocate the smallest hole that is big enough. We must search the entire list, unless the list is ordered by size. This strategy produces the smallest left over hole.

SOURCE CODE

```
// * Best- Fit * //

#include<stdio.h>

#include<conio.h>

void main()

{

        int m[5],n,mr[10],i,j,t,rm=0,pr[5]={1,2,3,4,5},q;

        char p[10][10];

        clrscr();

        printf("Enter the no.of memory partitions \n");

        scanf("%d",&q);

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

            {

          printf("
                  Enter the memory for partition %d \t",i++);
```

```
        scanf("%d",&m[i]);
}
      printf("Enter the no.of process \n:");
      scanf("%d",&n);
      for(i=0;i<n;i++)
       {
         printf(" Enter process name \t
          "); scanf("%s",&p[i]);
         printf("Enter memory required
         \t"); scanf("%d",&mr[i]);
       }
     for(i=0;i<q;i++)
      {
        for(j=i+1;j<q;j++)
         {
            if(m[i]>m[j])
             {
               t=m[i];
               m[i]=m[j];
```

```
                m[j]=t;

                t=pr[i];

                pr[i]=pr[j];

                pr[j]=t;



              }

        }

}

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

 {

     for(j=0;j<q;j++)

     if(m[j]>mr[i])

      {

        printf(" Process %s has been allocated partitio %d with space

                        %d kb \n ", p[i],pr[j],m[j]);
```

```
                    m[j]=m[j]-mr[i];
                     break;
                  }
               }
            for(i=0;i<q;i++)
            rm=rm+m[i];
         printf("remaining free space is %d kb \n", rm);
}
```

OUTPUT

Process p1 has been allocated partition 4 with space 300kb

Process p2 has been allocated partition 2 with space 500kb

Process p3 has been allocated partition 3 with space 200kb

Process p4 has been allocated partition 5 with space 600kb

Remaining free space is 533kb.

*** ----------****----------***

7. WORST FIT

AIM: - Program to implement Worst-Fit algorithm.

Description: - Allocate the largest hole. Again, we must search the entire list, unless it is sorted by size. This strategy produces the largest left over hole, which may be more useful than the smaller leftover hole from a best-fit approach.

SOURCE CODE

```
// * Worst- Fit * //

#include<stdio.h>

#include<conio.h>

void main()

{

        int m[5],n,mr[10],i,j,t,rm=0,pr[5]={1,2,3,4,5},q;

        char p[10][10];

        clrscr();

        printf(" Enter the no.of memory partitions \n");

        scanf("%d",&q);

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

        {

                Printf("Enter the memory for partition %d \t:", i+1);

                Scanf("%d",&m[i]);
```

```
        }
        Printf(" Enter the no.of process \n");
        Scanf("%d",&n);
        for( i=0;i<n;i++)
        {
                Printf(" Enter process name \t");
               Scanf("%s", &p[i]);
           Printf(" Enter memory required \t");
           Scanf("%d", &mr[i]);
}
for(i=0;i<q;i++)
{
          for(j=i+1;j<q;j++)
          {
                  if(m[i]<m[j])
                  {
                          t=m[i];
                          m[i]= m[j];
                          m[j]= t;
                          t=pr[i];
                          pr[i]=pr[j];
                          pr[j]=t;
```

```
                    }
                }
            }
    for(i=o;i<n;i++)
    {
            for(j=0;j<q;j++)
            if(m[j]>mr[i])
     {Printf(" Process %s has been allocated partition %d with
memory %d kb \n ", p[i],pr[j],m[j]);
                    m[j]=m[j]-
                    mr[i]; break;
            }
} for(i=0;i<q;i++)
rm=rm+m[i];
printf(" Remaining free space is % d kb \n", rm);
getch(); }
```

OUTPUT

Process p1 has been allocated partition 5 with memory 600kb

Process p2 has been allocated partition 2 with memory 500kb

Process p3 has been allocated partition 5 with memory

300kb Remaining free space is 959kb.

*** ----------****----------***

8.Page Replacement Using FIFO

AIM: - Program to implement Page Replacement Using FIFO

Description: - In FIFO page replacement algorithm, when a page must be replaced, the oldest page is chosen. Notice that it is not strictly necessary to record the time when a page is brought in. We can create a FIFO queue to hold all pages. We replace the page at the head of the queue when a page is brought into memory; we insert it at the tail of the queue.

SOURCE CODE

```
// * Page Replacement Using FIFO * //

#include<stdio.h>

#include<conio.h>

Int fr[3];

void main()

{

        Void display();

        int     i,j,page[12]={2,3,2,1,5,2,4,5,3,2,5,2};

        int    flag1=0,flage2=0,pf=0,frsize=3,top=0;

        clrscr();

        for(i=0;i<3;i++)

        {
```

```
            fr[i]=-1;
        }
        for(j=0;j<12;j++)
        {
            flag1=0;
            flag2=0;
            for(i=0;i<3;i++)
            {
                if(fr[i]==page[j])
                {
                    flag1=1;
                    flag2=1;
                    break;
                }
            }
            if(flag1==0)
            {
                for(i=0;i<frsize;i++)
                {
                    if(fr[i]==-1)
                    {
                        fr[i]=page[j];
```

```
                        flag2=1;
                        break;
                    }
                }
        }
        if(flag2==0)
        {
                    fr[top]=page[j];
                    top++;
                    pf++;
                    if(top>=frsize)
                       top=0;
        }
        display();
        }
        printf(" number of pages faults: %d",pf);
        getch();
        }
        void display()
        {
                    int i;
                    printf("\n");
```

```
        for(i=0;i<3;i++)
            printf("%d \t", fr[i]);
    }
```

OUTPUT

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

Number of page faults: 9

*** ----------****----------***

9. Page Replacement Using LRU (Least recently used)

AIM: - Program to implement least recently used using page Replacement algorithm.

Description: - In LRU, if we use the recent past as an approximation of the near future, then we can replace the page that has not been used for the longest period of time. This approach is the least recently used algorithm.

SOURCE CODE

```
// * Page Replacement Using LRU * //


#include<stdio.h>

#include<conio.h>

int fr[3];

void main()

{

                    void desplay();

                    int     p[12]={2,3,2,1,5,2,4,5,3,2,5,2},i,j,fs[3];

                    int    index,k,l,flag1=0,flag2=0,pf=0,frsize=3;

                    for(i=0;i<3;i++)

                    {

                       fr[i]=-1;
```

```
}
for(j=0;j<12;j++)
{
   flag1=0,flag2=0;
   for(i=0;i<3;i++)
   {
         if(fr[i]==p[j])
         {
               flag1=1;
               flag2=1;
               break;
         }
   }
   if(flag1==0)
   {
         for(i=0;i<3;i++)
         {
               if(fr[i]==-1)
               {
                     fr[i]=p[j];
                     flag2=1;
                     pf++;
```

```
                        break;
                }
        }
}
if(flag2==0)
{
        for(i=0;i<3;i++)
                fs[i]=0; for(k=j-
        1,l=1;l<=frsize-1;l++,k--)
        {
                for(i=0;i<3;i++)
                {
                        if(fr[i]==p[k])
                                fs[i]=1;
                }
        }
        for(i=0;i<3;i++)
        {
                if(fs[i]==0)
                        index=i;
        }
        fr[index]=p[j];
```

```
                pf++;
            }
            display();
        }
        printf("\n no.of page faults: %d",pf); getch();
}
void display()
{
        int i; printf("\n"); for(i=0;i<3;i++)
            printf("\t %d",fr[i]);
}
```

OUTPUT

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

2 5 4

2 5 4

3 5 4

3 5 2

3 5 2

3 5 2

Number of page faults: 7

*** ----------****----------***

10. OPTIMAL Page Replacement

AIM: - Program to implement optimal page replacement algorithm.

Description: - It is given as, which has the lowest page fault rate of all algorithms and will never suffer from the Beledy's anomaly simply given as replace the page that will not be used as the longest period of time.

SOURCE CODE

// * Optimal Page Replacement * //

```
#include<stdio.h>

#include<conio.h>
```

```
int fr[3];
void main()
{
        void desplay();
        int p[12]={2,3,2,1,5,2,4,5,3,2,5,2},i,j,fs[3];
        int max,found=0,lg[3],index,k,l,flag1=0,flag2=0,pf=0,frsize=3;
        clrscr();
        for(i=0;i<3;i++)
        {
                fr[i]=-1;
        }
        for(j=0;j<12;j++)
        {
                flag1=0;
                flag2=0;
                for(i=0;i<3;i++)
                {
                        if(fr[i]==p[j])
                        {
                                flag1=1;
                                flag2=1;
                                break;
```

```
            }
        }
        if(flag1==0)
        {
            for(i=0;i<3;i++)
            {
                if(fr[i]==-1)
                {
                    fr[i]=p[j];
                    flag2=1;
                    break;
                }
            }
        }
        if(flag2==0)
        {
            for(i=0;i<3;i++)
                lg[i]=0;
            for(i=0;i<frsize;i++)
                {
                    for(k=j+1;k<12;k++)
                    {
```

```
                        if(fr[i]==p[k])
                        {
                                lg[i]=k-j;
                                break;
                        }
                }
        }
found=0;
for(i=0;i<frsize;i++)
{
        if(lg[i]==0)
        {
                index=i;
                found=1;
                break;
        }
}
if(found==0)
{
        max=lg[0];
        index=0;
        for(i=1;i<frsize;i++)
```

```
                        {
                                if(max<lg[i])
                                {
                                        max=lg[i];
                                        index=i;
                                }
                        }
                }
                fr[index]=p[j];
                pf++;
        }
        display();
    }
    printf("\n no.of page faults: %d",pf);
    getch();
}
void display()
{ int i; printf("\n");

    for(i=0;i<3;i++)

        printf("\t %d", fr[i]);
}
```

OUTPUT

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

Number of page faults: 6

*** ----------****----------***

11. Deadlock Avoidence

AIM: - Program to implement Deadlock avoidance.

SOURCE CODE

```
// * Deadlock Avoidance * //
#include<stdio.h>
#include<conio.h>
void main()
{
        int work[3],sum,flag[3],set,finish[5],alloc[5][3],max[5][3];
        int need[5][3],i,j,k,count=0,available[]={10,5,7};
        clrscr();
        for(i=0;i<3;i++)
        {
                flag[i]=-1;
                work[i]=available[i];
        }
        for(i=0;i<5;i++)
        {
                finish[i]=-1;
        }
        for(i=0;i<5;i++)
                for(j=0;j<3;j++)
```

```
                {

                printf("\n enter no.of allocation resource %d of
process %d: : ", j,i);

                        scanf("%d",&alloc[i][j]);

                printf("\n enter max no.of allocation resource %d to
process %d: : ", j,i);

                        scanf("%d",&max[i][j]);

                }

        for(i=0;i<5;i++)

                for(j=0;j<3;j++)

                        need[i][j]=max[i][j]-alloc[i][j];

        for(j=0;j<3;j++)

        {

                sum=0;

                for(i=0;i<5;i++)

                        sum=sum+alloc[i][j];

                work[j]=work[j]-sum;

        }

        while(count!=5&&n<=15)

        {

                for(i=0;i<5;i++)

                {

                        if(finish[i]==-1)
```

```
                {
                        for(j=0;j<3;j++)
                        {
                                flag[j]=-1;
                                if(need[i][j]<=work[j])
                                        flag[j]=0;
                                else
                                        break;
                        }
                        if(flag[0]==0&&flag[1]=0&&flag[2]==0)
                        {
                                for(k=0;k<3;k++)
                                        work[k]=work[k]+alloc[i][k];
                                finish[i]=0;
                                count=count+1;
                        }
                        printf(" \n process %d is finished ",i);
                }
        }
        n++;
}
if(count==5)
```

```
                printf("\n system will be in safe state");
}
else
printf("system is in deadlock state");
printf(" \n process causing deadlock are:");
for(i=0;i<5;i++)
if(finish[i]==-1)
        printf("\n process %d",i);
}
getch();
}
```

OUTPUT

```
Enter no. of allocation resource 0 to process 0: 0
Enter max of allocation resource 0 to process 0: 7
Enter no. of allocation resource 1 to process 0: 1
Enter max of allocation resource 1 to process 0: 5
Enter no. of allocation resource 2 to process 0: 0
Enter max of allocation resource 2 to process 0: 3
Enter no. of allocation resource 0 to process 1: 2
Enter max of allocation resource 0 to process 1: 3
```

Enter no. of allocation resource 1 to process 1: 0

Enter max of allocation resource 1 to process 1: 2

Enter no. of allocation resource 2 to process 1: 0

Enter max of allocation resource 2 to process 1: 2

Enter no. of allocation resource 0 to process 2: 3

Enter max of allocation resource 0 to process 2: 9

Enter no. of allocation resource 1 to process 2: 0

Enter max of allocation resource 1 to process 2: 0

Enter no. of allocation resource 2 to process 2: 2

Enter max of allocation resource 2 to process 2: 2

Enter no. of allocation resource 0 to process 3: 2

Enter max of allocation resource 0 to process 3: 2

Enter no. of allocation resource 1 to process 3: 1

Enter max of allocation resource 1 to process 3: 2

Enter no. of allocation resource 2 to process 3: 1

Enter max of allocation resource 2 to process 3: 2

Allocation resource ----------

Something is to be filed above.....

Work 0: 3

Work 1: 3

Work 2: 2

Process 1 is finished

Process 3 is finished

Process 4 is finished

Process 0 is finished

Process 2 is finished

System is in safe state.

*** ----------****----------***

12. Deadlock Detection

AIM: - Program to show Deadlock detection.

SOURCE CODE

```
// * Deadlock Detection * //


#include<stdio.h>

#include<conio.h>

void main()

{
```

```
int work[3],sum,flag[3],set,finish[5],alloc[5][3],max[5][3],n=0;
int need[5][3],i,j,k,count=0,available[]={10,5,7};
clrscr();
for(i=0;i<3;i++)
{
        flag[i]=-1;
        work[i]=available[i];
}
for(i=0;i<5;i++)
{
        finish[i]=-1;
}
for(i=0;i<5;i++)
        for(j=0;j<3;j++)
        {
       printf("\n enter no.of allocation resource %d of
process %d: : ", j,i);
                scanf("%d",&alloc[i][j]);
       printf("\n enter max no.of allocation resource %d to
process %d: : ", j,i);
                scanf("%d",&max[i][j]);
        }
for(i=0;i<5;i++)
```

```
        for(j=0;j<3;j++)
                need[i][j]=max[i][j]-alloc[i][j];
for(j=0;j<3;j++)
{
        sum=0;
        for(i=0;i<5;i++)
                sum=sum+alloc[i][j];
        work[j]=work[j]-sum;
}
for(i=0;i<5;i++)
        {
                if(finish[i]==-1)
                {
                        for(j=0;j<3;j++)
                        {
                                flag[j]=-1;
                                if(need[i][j]<=work[j])
                                        flag[j]=0;
                                else
                                        break;
                        }
                        if(flag[0]==0&&flag[1]=0&&flag[2]==0)
```

```
                        {
                                for(k=0;k<3;k++)
                                        work[k]=work[k]+alloc[i][k];
                                finish[i]=0;
                                count=count+1;
                        }
                        printf(" \n process %d is finished ",i);
                    }
                }
                n++;
        }
        if(count==5)
                printf("\n system is in safe state");
}
else
printf("system is not safe");
printf(" \n process causing deadlock are:");
for(i=0;i<5;i++)
if(finish[i]==-1)
        printf("\n process %d",i);
        }
getch();
```

}

OUTPUT

```
Enter no. of allocation resource 0 to process 0: 0
Enter max of allocation resource 0 to process 0: 7
Enter no. of allocation resource 1 to process 0:        3
Enter max of allocation resource 1 to process 0: 0
Enter no. of allocation resource 2 to process 0:        7
Enter max of allocation resource 2 to process 0: 2
Enter no. of allocation resource 0 to process 1: 3
Enter max of allocation resource 0 to process 1: 0
Enter no. of allocation resource 1 to process 1:        2
Enter max of allocation resource 1 to process 1: 0
Enter no. of allocation resource 2 to process 1:        2
Enter max of allocation resource 2 to process 1: 3
Enter no. of allocation resource 0 to process 2:        9
Enter max of allocation resource 0 to process 2: 0
Enter no. of allocation resource 1 to process 2:        0
Enter max of allocation resource 1 to process 2: 2
Enter no. of allocation resource 2 to process 2: 2
Enter max of allocation resource 2 to process 2: 2
Enter no. of allocation resource 0 to process 3: 2
Enter max of allocation resource 0 to process 3: 2
```

Enter no. of allocation resource 1 to process 3: 1

Enter max of allocation resource 1 to process 3: 2

Enter no. of allocation resource 2 to process 3: 1

Enter max of allocation resource 2 to process 3: 2

Allocation resource ----------

Something is to be filed above.....

Work 0: 3

Work 1: 1

Work 2: 2

Process 3 is finished

Process 1 is finished

Process 2 is finished

System is not in safe state.

Process that cause deadlock are:

Process 0

Process 4

*** ----------****----------***

13. Sequential

AIM: - Write a Program to implement sequential.

SOURCE CODE

```
// * Sequential * //
#include<Mstdio.h>
#include<graphics.h>
#include<stdlib.h>
#include<conio.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:");
        scanf("%d",&n); printf("enter
        the details \n");
        for(i=0;i<n;i++)
        {
                printf("file name:");
                fflish(stdin);
                gets(F[i].name);
                printf("starting block:");
                scanf("%d",&F[i].st-blk);
                printf("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-dist()
{
        int i,j,x1,y1;
        char s[5];
        setcolor(LIGHTGREEN);
        settextstyle(1,0,3);
        settextjustify(1,1);
        outtextxy(320,20,"contiguous File allocatio");
        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);
                        printf(s, "%d", i*7+j);
                        outtextxy(x1+BL/2,y1+BW+10,s);
                }
        rectangle(sx-20,sy-20,sx+350,sy+280);
        settextstyle(1,0,2);
        for(i=0;i<n;i++)
        {
                setcolor(F[i].color);
                outtextxy(100,400+i*20,F[i].name);
        }
        setcolor(14);
}
void main()
{
        int gd,gm;
        gd=DETECT;
        initgraph(&gd,&gm,"c:\tc\\BGI");
        randomize();
```

```
            read-files();
            cleardevice();
            design-disk();
            fillspace();
            getch();
            closegrapgh();
        }
}
```

OUTPUT

How many files: 2

Enter the details

Filename: a

Starting block: 12

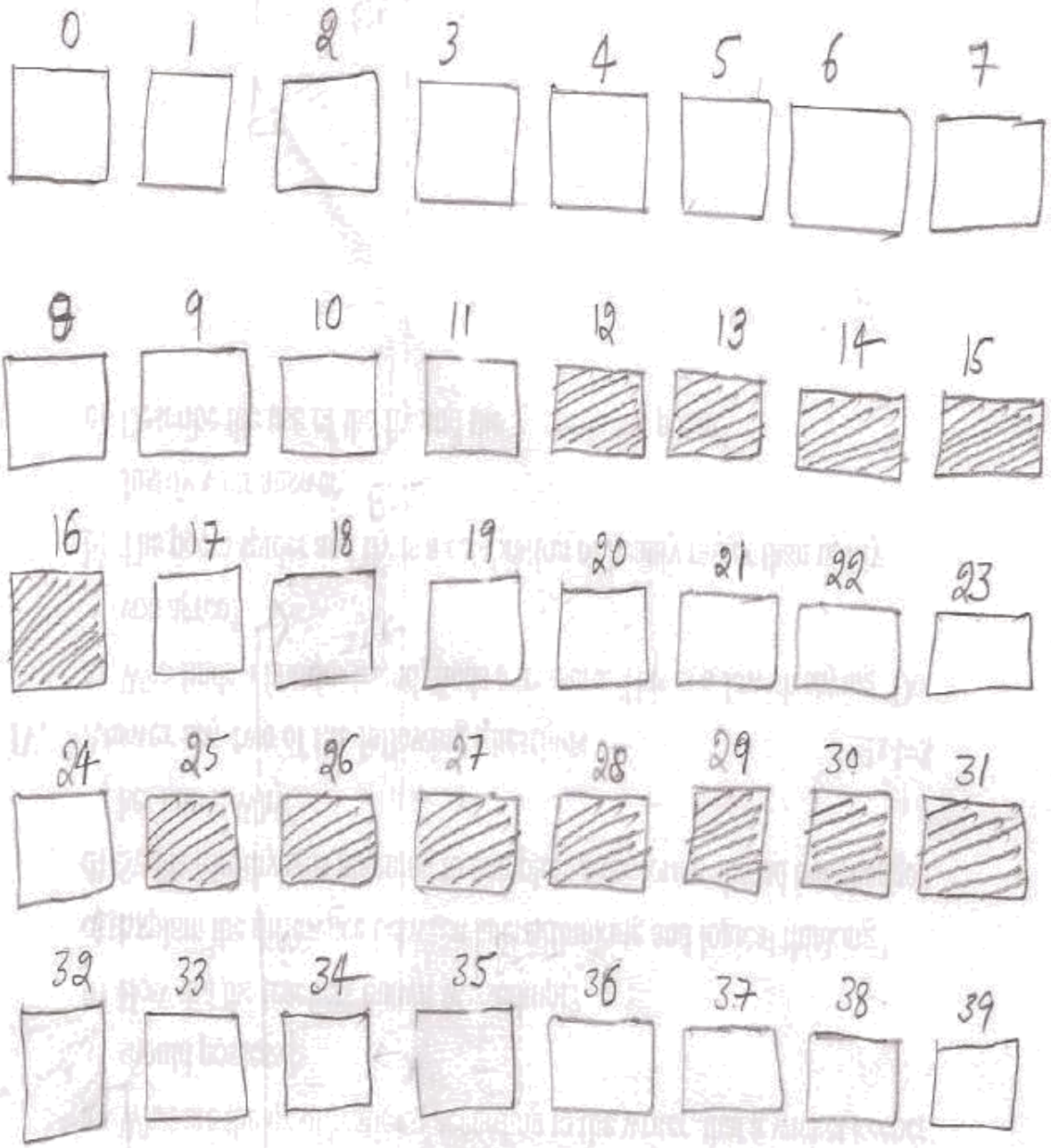
Length: 5

Filename: b

Starting block: 25

Length: 8

Contiguous file allocation:-



*** ----------****----------***

14. Simulation Linked allocation

AIM: - Write a Program for simulation linked allocation.

SOURCE CODE

```
// * Simulation linked allocation * //

#include<Mstdio.h>
#include<graphics.h>
#include<conio.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",28,7,CYAN};

intblocks-
allocated[63]={28,16,31,33,38,55,56,29,1,30,48,12,19,20,21,22,39,
40,18,2,10,50,51,52,54,58,17,35,43,13,3,36,49,24,6,4,14,16,25,37,
15,7,56,11,26,9,27,45,55,23,34,41,42,59,8,53};


block b[63];

read-file-info()

{

        ptrintf("enter the file-name:");

        fflish(stdin);

        gets(f.name);

        printf("length:");

        scanf("y.d",&f.len);

}

create-info-table()

 {

        int i;

        char s;

        settextstyle(2,0,4);

        setcolor(GREEN);
```

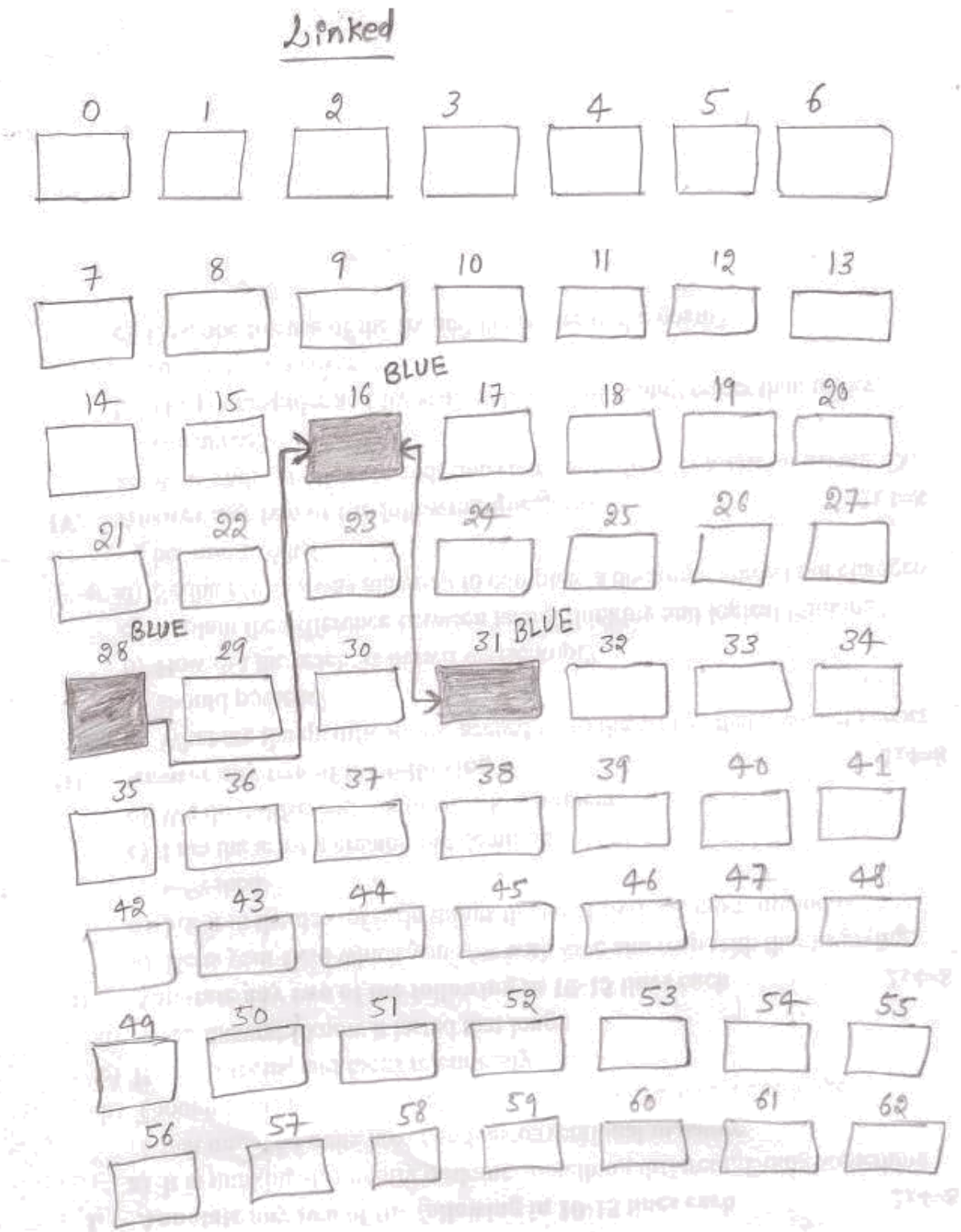
```
        rectangle(500,200,639,300);
        line(580,200,580,300);
        line(500,230,639,230);
        line(500,260,639,260);
        settextxy(505,220,"filename");
        outtextxy(505,250,"st.Block");
        sprintf(s,"d",f.len);
        outtextxy(595,280,s);
}
fill-space()
{
        int i,n;
        for(i=0;i<f.len;i++)
        {
                n=blocked-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++]);
                sleep[1];
        }
}
```

```
void crea-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;
}
design-disk()
{
        int i,j,x1,v1;
        char s[5];
        setcolor(LIGHTGREEN);
        settextstyle(1,0,3);
        settextjustify(1,1);
        outtextjustify(320,20,"LINKEN ALLOCATION");
        setcolor(14);
        settextstyle(2,0,3);
        settextjustify(1,1);
        for(i=0;i<63;i++)
        {
```

```
                crea-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);
        }
        rectangle(sx-20,sx-20,sx+350,sx+360);
        settextstyle(1,0,2);
        setcolor(14);
}
void draw-link-lines(int st, int end)
{
        int row,col;
        move to(b[st].x2,b[st].y1+7);
        linerel(10,0);
        lineto(getx(),b[end],y1-10);
        lineto(b[end],x1-10,gety());
        lineto(get(),b[end].y1+10);
        lineto(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,"D://TCPP//BGI");
        cleardevice();
        sleep(2);
        read-file-info();
        design-disk();
        fillspace();
        create-info-table();
        getch();
        close graph();
}
```

OUTPUT


Linked
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
BLUE
17
18
19
20
21
22
23
24
25
26
27
28
BLUE
29
30
31
BLUE
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62

15. Index allocation

AIM: - Write a Program for index allocation.

SOURCE CODE

```
// * Index allocation * //


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

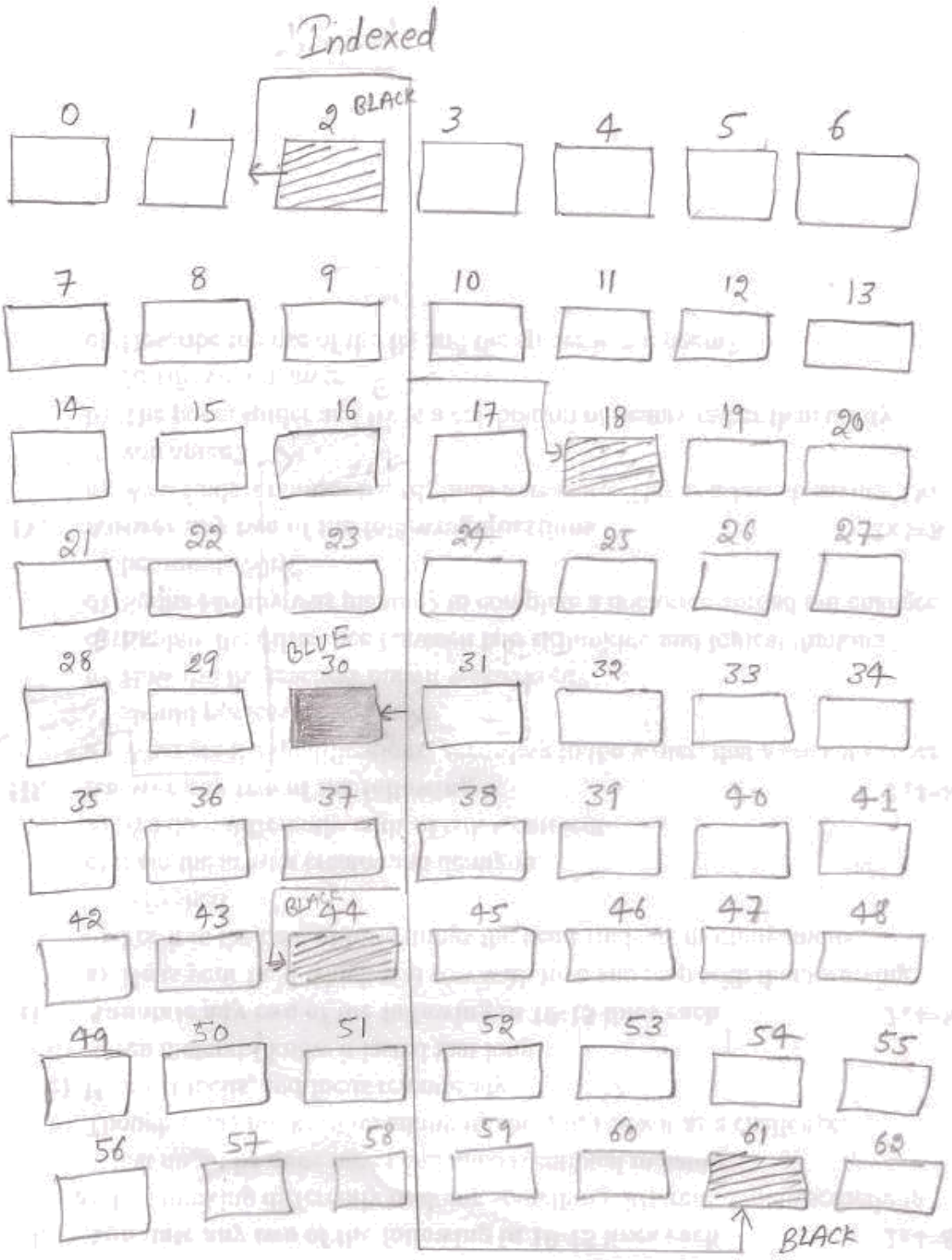
```
    int len,fcolor;
} file-info;
void draw-link-lines(int st, int end);
file-info.f={"ABC.T*T",10,7,RED};
int blocks- allocated[7]={30,2,18,27,44,61};
block b[63];
void fill-space() /* file allocated disk blocks */
{
                    int i,n;
                    setfillstyle(1,LIGHTMAGENTA);
                    N=BLOCKS-ALLOCATED[0];
                    bar3d(b[n].x1,b[n].y1,b[n].x2,b[n].y2,0,0);
                    SET FILLSTYLE(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];
                    }
```

```
}
/* assign attributes for disk blocks
*/ void crea-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,"INDEX
                    ALLOCATION"); setcolor(14);
```

```
                    settextstyle(2,0,3);
                    settextjustify(1,1);
                    for(i=0;i<63;i++)
                    {
                       crea-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,sx-20,sx+350,sy+360);
                    settextstyle(1,0,2);
                    setcolor(14);
}
/* draws link lines between allocated blocks and index blocks */
void draw-link-lines(int st, int end)
{
                    int row,col,x;
                    while((x=random(16))==0)
                       continue;
                    setcolor(x);
                    move to(b[st].x2,b[st].y1+7);
```

```
                    linerel(10,0);
                    lineto(getx(),b[end].y1-10);
                    lineto(b[end],x1-10,gety());
                    lineto(get(),b[end].y1+10);
                    lineto(b[end].x1,b[end].y1+10);
                    line(b[end].x1,b[end].y1+10,b[end].x1-
5,b[end].y1+15);
                    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,"E:\Tc\
                    BGI"); randomize();
                    cleardevice();
                    sleep(2);
                    design-disk();
                    fillspace();
                    getch();
                    close graph();
}
```

OUTPUT


Indexed
0
1
2
BLACK
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
BLUE
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
BLACK