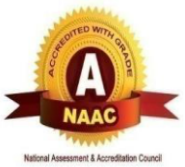
Malla Reddy College Engineering (Autonomous)



Maisammaguda, Dhulapally (Post Via. Hakimpet), Secunderabad, Telangana-500100 www.mrec.ac.in

Department of Information Technology

II B. TECH I SEM (A.Y.2018-19)

80509 - OPERATING SYSTEMS LAB

2018-19 Onwards (MR-18)	MALLA REDDY ENGINEERING COLLEGE (Autonomous)	B.Tech. III Semester		
Code: 80509	OPERATING SYSTEMS LAB (Common for CSE and IT)	L	T	P
Credits: 2		-	1	2

Prerequisite: 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 Dhamdhere**,"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 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		

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

EXPERIMENT 1

OBJECTIVE

Write a C program to simulate the following non-preemptive CPU scheduling algorithms to find turnaround time and waiting time for the above problem.

a) FCFS b) SJF

DESCRIPTION

Assume all the processes arrive at the same time.

FCFS CPU SCHEDULING ALGORITHM

For FCFS scheduling algorithm, read the number of processes/jobs in the system, their CPU burst times. The scheduling is performed on the basis of arrival time of the processes irrespective of their other parameters. Each process will be executed according to its arrival time. Calculate the waiting time and turnaround time of each of the processes accordingly.

SJF CPU SCHEDULING ALGORITHM

For SJF scheduling algorithm, read the number of processes/jobs in the system, their CPU burst times. Arrange all the jobs in order with respect to their burst times. There may be two jobs in queue with the same execution time, and then FCFS approach is to be performed. Each process will be executed according to the length of its burst time. Then calculate the waiting time and turnaround time of each of the processes accordingly.

FCFS

```
#include<stdio.h>
#include<conio.h>
main()
{
int bt[20], wt[20], tat[20], i, n;
float wtavg, tatavg;
clrscr();
    printf("\nEnterthenumberofprocesses-- ");
    scanf("%d", &n);
        for(i=0;i<n;i++)
        {
        printf("\nEnter Burst Time for Process%d-- ", i);
       scanf("%d", &bt[i]);
        }
        wt[0] = wtavg = 0;
        tat[0] = tatavg =
        bt[0];
        for(i=1;i<n;i++)
        {
                wt[i] = wt[i-1]+bt[i-1];
                tat[i] = tat[i-1]
                +bt[i]; wtavg =
                wtavg + wt[i];
                tatavg= tatavg+
                tat[i];
        }
        printf("\t PROCESS \tBURST TIME \t WAITING TIME\t TURNAROUND TIME\n");

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

```
        printf("\n\t P%d \t\t %d \t\t %d \t\t %d", i, bt[i], wt[i],
        tat[i]); printf("\nAverage Waiting Time -- %f", wtavg/n);
  printf("\nAverage Turnaround Time -- %f", tatavg/n);
        getch();
}
```

INPUT

```
Enterthe number of processes-- 3
Enter Burst TimeforProcess 0-- 24
Enter Burst TimeforProcess 1-- 3
Enter Burst TimeforProcess 2-- 3
```

OUTPUT

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

```
AverageWaitingTime-- 17.000000
Average Turnaround Time-- 27.000000
```

SJF CPU SCHEDULING ALGORITHM

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

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

                                temp=p[i];
                                p[i]=p[k];
                                p[k]=temp;
                        }
        wt[0] = wtavg = 0;
        tat[0] = tatavg = bt[0];
        for(i=1;i<n;i++)
        {
```

```
                wt[i] = wt[i-1]+bt[i-1];
                tat[i] = tat[i-1] +bt[i];
                wtavg = wtavg + wt[i];
                tatavg= tatavg+tat[i];
        }
        printf("\n\t PROCESS \tBURST TIME \t WAITING TIME\t TURNAROUND TIME\n");
        for(i=0;i<n;i++)

        printf("\n\t P%d \t\t %d \t\t %d \t\t %d", p[i], bt[i], wt[i], tat[i]);
        printf("\nAverage Waiting Time -- %f", wtavg/n);
        printf("\nAverage Turnaround Time -- %f", tatavg/n);
        getch();
}
```

INPUT

```
Enter thenumber of processes -- 4
Enter Burst Timefor Process 0 -- 6
Enter Burst Timefor Process 1 -- 8
Enter Burst Timefor Process 2 -- 7
Enter Burst Timefor Process 3 -- 3
```

OUTPUT

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

EXPERIMENT 2

OBJECTIVE

Write a C program to simulate the following non-preemptive CPU scheduling algorithms to find turnaround time and waiting time for the above problem.

a) Round Robin b) Priorit

ROUND ROBIN CPU SCHEDULING ALGORITHM:

For round robin scheduling algorithm, read the number of processes/jobs in the system, their CPU burst times, and the size of the time slice. Time slices are assigned to each process in equal portions and in circular order, handling all processes execution. This allows every process to get an equal chance. Calculate the waiting time and turnaround time of each of the processes accordingly

PRIORITY CPU SCHEDULING ALGORITHM

For priority scheduling algorithm, read the number of processes/jobs in the system, their CPU burst times, and the priorities. Arrange all the jobs in order with respect to their priorities. There may be two jobs in queue with the same priority, and then FCFS approach is to be performed. Each process will be executed according to its priority. Calculate the waiting time and turnaround time of each of the processes accordingly.

ROUND ROBIN CPU SCHEDULING ALGORITHM

```
#include<stdio.h>
main()
{
        int i,j,n,bu[10],wa[10],tat[10],t,ct[10],max;
        float awt=0,att=0,temp=0;
        clrscr();
        printf("Entertheno of processes-- ");
        scanf("%d",&n);

        for(i=0;i<n;i++)
        {
                printf("\nEnterBurstTimeforprocess%d--",i+1);
                scanf("%d",&bu[i]);
                ct[i]=bu[i];
        }
                printf("\nEnter the size of time slice -- ");
                scanf("%d",&t);
                max=bu[0];
                for(i=1;i<n;i++)
                        if(max<bu[i])
                        max=bu[i];
      for(j=0;j<(max/t)+1;j++)
      for(i=0;i<n;i++)
      if(bu[i]!=0)
      if(bu[i]<=t)
      {
      tat[i]=temp+bu[i];
      temp=temp+bu[i];
      bu[i]=0;
      }
      else
```

```
        {
        bu[i]=bu[i]-t;
        temp=temp+t;
        }
        for(i=0;i<n;i++)
        {
        wa[i]=tat[i]-ct[i];.
        att+=tat[i];
        awt+=wa[i];
        }
 printf("\nThe Average Turnaround time is -- %f",att/n);
  printf("\nThe Average Waiting time is -- %f ",awt/n);
        printf("\n\tPROCESS\t BURST TIME \t WAITING TIME\tTURNAROUND TIME\n");
        for(i=0;i<n;i++)
        printf("\t%d \t %d \t\t %d \t\t %d \n",i+1,ct[i],wa[i],tat[i]);
        getch();
        }
```

INPUT

Enter the no of processes – 3
Enter Burst Timeforprocess1 – 24
Enter Burst Timeforprocess2 -- 3
Enter Burst Timeforprocess3 -- 3

Enter the size of time slice – 3

OUTPUT

The Average Turnaround time is– 15.666667
The Average Waitingtimeis-- 5.666667

PROCESS	BURST TIME	WAITING TIME	TURNAROUND TIME
1	24	6	30
2	3	4	7
3	3	7	10

PRIORITY CPU SCHEDULING ALGORITHM

```
#include<stdio.h>
main()
{
        int p[20],bt[20],pri[20], wt[20],tat[20],i, k, n, temp;
        float wtavg, tatavg;
        clrscr();
        printf("Enter the number of processes --- ");
        scanf("%d",&n);

        for(i=0;i<n;i++)
        {
                p[i] = i;
                printf("Enterthe BurstTime&Priority of Process%d --- ",i);
                scanf("%d %d",&bt[i], &pri[i]);
        }
        for(i=0;i<n;i++)
                for(k=i+1;k<n;k++)
                        if(pri[i] > pri[k])
                        {
                                temp=p[i];
                                p[i]=p[k];
                                p[k]=temp;

                                temp=bt[i];
                                bt[i]=bt[k];
                                bt[k]=temp;

                                temp=pri[i];
                                pri[i]=pri[k];
                                pri[k]=temp;

                        }
        wtavg = wt[0] = 0;
        tatavg = tat[0] = bt[0];
        for(i=1;i<n;i++)
        {
                wt[i] = wt[i-1] + bt[i-1];
                tat[i] = tat[i-1] + bt[i];

                wtavg = wtavg + wt[i];
                tatavg= tatavg+ tat[i];
        }

        printf("\nPROCESS\t\tPRIORITY\tBURST TIME\tWAITING TIME\tTURNAROUND  TIME");
        for(i=0;i<n;i++)
                printf("\n%d \t\t %d \t\t %d \t\t %d \t\t %d ",p[i],pri[i],bt[i],wt[i],tat[i]);

        printf("\nAverage Waiting Time is --- %f",wtavg/n);
        printf("\nAverage Turnaround Time is --- %f",tatavg/n);
        getch();
}
```

INPUT

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

OUTPUT

PROCESS	PRIORITY	BURST TIME	WAITING TIME	TURNAROUND TIME
1	1	1	0	1
4	2	5	1	6
0	3	10	6	16
2	4	2	16	18
3	5	1	18	19

Average Waiting Time is --- 8.200000
Average Turnaround Timeis---12.000000

EXPERIMENT 3

OBJECTIVE

*****Write a C program to simulate producer-consumer problem using semaphores.

DESCRIPTION

Producer-consumer problem, is a common paradigm for cooperating processes. A producer process produces information that is consumed by a consumer process. One solution to the producer-consumer problem uses shared memory. To allow producer and consumer processes to run concurrently, there must be available a buffer of items that can be filled by the producer and emptied by the consumer. This buffer will reside in a region of memory that is shared by the producer and consumer processes. A producer can produce one item while the consumer is consuming another item. The producer and consumer must be synchronized, so that the consumer does not try to consume an item that has not yet been produced.

PROGRAM

```
#include<stdio.h>
void main()
{
        int buffer[10], bufsize, in, out, produce, consume, choice=0;
        in = 0;
        out = 0;
        bufsize = 10;
        while(choice !=3)
        {
                printf("\n1. Produce \t 2. Consume\t3. Exit");
                printf("\nEnter your choice: ");
                scanf("%d", &choice);
                switch(choice)  {
                        case 1: if((in+1)%bufsize==out)
                                        printf("\nBuffer is Full");
                                else
                                {
                                        printf("\nEnter the value: ");
                                        scanf("%d", &produce);
                                        buffer[in] = produce;
                                        in = (in+1)%bufsize;
                                }
                                Break;
                        case 2: if(in == out)
                                        printf("\nBuffer is Empty");
                                else
                                {
                                        consume = buffer[out];
                                        printf("\nThe consumed value is %d", consume);
                                        out = (out+1)%bufsize;
                                }
                                break;

        }

    }
}
```

OUTPUT

```
1. Produce      2. Consume      3.Exit
Enter your choice:2
Buffer is Empty
1. Produce      2. Consume      3.Exit
Enter your choice:1
Enter the value: 100
1. Produce      2. Consume      3.Exit
Enter your choice:2
The consumed value is 100
1. Produce      2. Consume      3.Exit
Enter your choice:3
```

EXPERIMENT 4

OBJECTIVE

Write a C program to simulate Bankers algorithm for the purpose of deadlock avoidance.

DESCRIPTION

In a multiprogramming environment, several processes may compete for a finite number of resources. A process requests resources; if the resources are not available at that time, the process enters a waiting state. Sometimes, a waiting process is never again able to change state, because the resources it has requested are held by other waiting processes. This situation is called a deadlock. Deadlock avoidance is one of the techniques for handling deadlocks. This approach requires that the operating system be given in advance additional information concerning which resources a process will request and use during its lifetime. With this additional knowledge, it can decide for each request whether or not the process should wait. To decide whether the current request can be satisfied or must be delayed, the system must consider the resources currently available, the resources currently allocated to each process, and the future requests and releases of each process. Banker's algorithm is a deadlock avoidance algorithm that is applicable to a systemwith multiple instances of each resource type.

PROGRAM

```
#include<stdio.h>
struct file
{
        int all[10];
        int max[10];
        int need[10];
        int flag;
};
void main()
{
        struct file f[10];
        int fl;
        int i, j, k, p, b, n, r, g, cnt=0, id, newr;
        int avail[10],seq[10];
        clrscr();
        printf("Enter number of processes -- ");
        scanf("%d",&n);
        printf("Enter number of resources -- ");
        scanf("%d",&r);
        for(i=0;i<n;i++)
        {
                printf("Enter detailsfor P%d",i);
                printf("\nEnter allocation\t -- \t");
                for(j=0;j<r;j++)
                        scanf("%d",&f[i].all[j]);
                printf("Enter Max\t\t -- \t");
                for(j=0;j<r;j++)
                        scanf("%d",&f[i].max[j]);
                f[i].flag=0;
         }
         printf("\nEnter Available Resources\t -- \t");
         for(i=0;i<r;i++)
```

```
            scanf("%d",&avail[i]);

printf("\nEnter New Request Details -- ");
printf("\nEnter pid \t -- \t");
scanf("%d",&id);
printf("EnterRequestforResources\t--\t");
for(i=0;i<r;i++)
{
        scanf("%d",&newr);
        f[id].all[i] += newr;
        avail[i]=avail[i] - newr;
}

for(i=0;i<n;i++)
{
        for(j=0;j<r;j++)
        {
                f[i].need[j]=f[i].max[j]-f[i].all[j];
                if(f[i].need[j]<0)
                        f[i].need[j]=0;
        }
}
cnt=0;
fl=0;
while(cnt!=n)
{
        g=0;
        for(j=0;j<n;j++)
        {
                if(f[j].flag==0)
                {
                        b=0;
                        for(p=0;p<r;p++)
                        {
                                if(avail[p]>=f[j].need[p])
                                        b=b+1;
                                else
                                        b=b-1;
                        }
                        if(b==r)
                        {
                                printf("\nP%d is visited",j);
                                seq[fl++]=j;
                                f[j].flag=1;
                                for(k=0;k<r;k++)
                                        avail[k]=avail[k]+f[j].all[k];
                                cnt=cnt+1;
                                printf("(");
                                for(k=0;k<r;k++)
                                        printf("%3d",avail[k]);
                                printf(")");
                                g=1;

                        }
                }
        }
        if(g==0)
        {
                printf("\n REQUEST NOT GRANTED -- DEADLOCK OCCURRED");
                printf("\n SYSTEM IS IN UNSAFE STATE");
                goto y;
        }
```

```
        }
        printf("\nSYSTEM IS IN SAFE STATE");
        printf("\nThe Safe Sequence is -- (");
        for(i=0;i<fl;i++)
                printf("P%d ",seq[i]);
        printf(")");
        printf("\nProcess\t\tAllocation\t\tMax\t\t\tNeed\n");
        for(i=0;i<n;i++)
        {
                printf("P%d\t",i);
                for(j=0;j<r;j++)

                }

                getch();

                }

}
```

INPUT

```
Enter number of processes        –      5
Enter number of resources        --     3
Enter details for P0
Enter allocation          --     0      1      0
Enter Max                 --     7      5      3

Enter details for P1
Enter allocation          --     2      0      0
Enter Max                 --     3      2      2

Enter details for P2
Enter allocation          --     3      0      2
Enter Max                 --     9      0      2

Enter details for P3
Enter allocation          --     2      1      1
Enter Max                 --     2      2      2

Enter details for P4
Enter allocation          --     0      0      2
Enter Max                 --     4      3      3

Enter Available Resources -- 3    3      2
Enter New Request Details--
Enter pid      --   1
Enter Request for Resources  --   1      0      2
```

OUTPUT

```
P1 is visited( 5 3 2)
P3 is visited( 7 4 3)
P4 isvisited( 7 4 5)
P0 isvisited( 7 5 5)
P2 is visited( 10 5 7)
SYSTEM IS IN SAFE STATE
The Safe Sequence is-- (P1 P3  P4 P0 P2 )
```

Process	Allocation	Max	Need
P0	0 1 0	7 5 3	7 4 3
P1	3 0 2	3 2 2	0 2 0
P2	3 0 2	9 0 2	6 0 0
P3	2 1 1	2 2 2	0 1 1
P4	0 0 2	4 3 3	4 3 1

EXPERIMENT 5

OBJECTIVE

Write a C program to simulate the MVT and MFT memory management techniques

DESCRIPTION

MFT (Multiprogramming with a Fixed number of Tasks) is one of the old memory management techniques in which the memory is partitioned into fixed size partitions and each job is assigned to a partition. The memory assigned to a partition does not change. MVT (Multiprogramming with a Variable number of Tasks) is the memory management technique in which each job gets just the amount of memory it needs. That is, the partitioning of memory is dynamic and changes as jobs enter and leave the system. MVT is a more ``efficient" user of resources. MFT suffers with the problem of internal fragmentation and MVT suffers with external fragmentation.

PROGRAM

MEMORY MANAGEMENT TECHNIQUE

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

main()
{
int ms, bs, nob, ef,n, mp[10],tif=0; int i,p=0;

clrscr();
printf("Enter the totalmemoryavailable (in Bytes) -- ");
scanf("%d",&ms);
printf("Enter theblock size (in Bytes) -- ");
scanf("%d", &bs);
nob=ms/bs; ef=ms - nob*bs;
printf("\nEnterthe numberofprocesses -- ");
```

```
 scanf("%d",&n);
for(i=0;i<n;i++)
{

    printf("Enter memory required for process %d (in Bytes)--",i+1);
   scanf("%d",&mp[i]);
}
printf("\nNo. of Blocks available in memory -- %d",nob);
printf("\n\nPROCESS\tMEMORY REQUIRED\t ALLOCATED\tINTERNAL
FRAGMENTATION");

for(i=0;i<n && p<nob;i++)
{
  printf("\n d\t\t%d",i+1,mp[i]); if(mp[i] > bs)

  printf("\t\tNO\t\t---");

else
{
printf("\t\tYES\t%d",bs-mp[i]);
tif = tif + bs-mp[i];
p++;

}
}

 if(i<n)

       printf("\nMemory is Full, Remaining Processes cannot be accomodated");

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

}
```

.

INPUT

Enter thetotalmemory available (inBytes 1000
Enterthe blocksize (inBytes)-- 300

Enter the number ofprocesses 5

Entermemory required forprocess1 (in Bytes) -- 275
Entermemory required forprocess2 (in Bytes) -- 400
Entermemory required forprocess3 (in Bytes) -- 290
Entermemory required forprocess4 (in Bytes) -- 293
Entermemory required forprocess5 (in Bytes) -- 100

No. of Blocksavailableinmemory -- 3

OUTPUT

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

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

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

main()
{
int ms, bs, nob, ef,n, mp[10],tif=0; int i,p=0;

clrscr();
printf("Enter the total memory available (in Bytes) -- "); scanf("%d",&ms);
printf("Enter the block size (in Bytes) -- "); scanf("%d", &bs);
nob=ms/bs; ef=ms - nob*bs;
printf("\nEnter the number of processes -- "); scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("Enter memory required for process %d (in Bytes)-- ",i+1);
scanf("%d",&mp[i]);
}

printf("\nNo. of Blocks available in memory -- %d",nob);
printf("\n\nPROCESS\tMEMORY REQUIRED\t ALLOCATED\tINTERNAL FRAGMENTATION");
for(i=0;i<n && p<nob;i++)
{
printf("\n %d\t\t%d",i+1,mp[i]);
 if(mp[i] > bs)
printf("\t\tNO\t\t---");

else
{
printf("\t\tYES\t%d",bs-mp[i]);
 tif = tif + bs-mp[i];
p++;
}}
if(i<n)
 printf("\nMemory is Full, Remaining Processes cannot be accomodated");

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

INPUT

Enter the total memory available (in Bytes) -- 1000
Enter the block size (in Bytes)-- 300 Enter the number of processes 5
Enter memory required for process 1 (in Bytes) -- 275
Enter memory required for process 2 (in Bytes) -- 400
Enter memory required for process 3 (in Bytes) -- 290
Enter memory required for process 4 (in Bytes) -- 293
Enter memory required for process 5 (in Bytes) -- 100
No. of Blocks available in memory -- 3

OUTPUT

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

MVT MEMORY MANAGEMENT TECHNIQUE

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

main()
{
int ms,mp[10],i, temp,n=0;
char ch = 'y';

clrscr();
printf("\nEnter the total memory available (in Bytes)-- ");
scanf("%d",&ms);
temp=ms;
for(i=0;ch=='y';i++,n++)
{
     printf("\nEnter memory required for process %d (in Bytes) -- ",i+1);
     scanf("%d",&mp[i]);
if(mp[i]<=temp)
   {

printf("\nMemory is allocated for Process %d ",i+1);
temp = temp - mp[i];

}
```

```
            else
            {
                printf("\nMemory is Full");
                break;

            }

printf("\nDo you want to continue(y/n) -- ");
scanf(" %c", &ch);
}
printf("\n\nTotal Memory Available -- %d", ms);

printf("\n\n\tPROCESS\t\t MEMORY ALLOCATED ");
for(i=0;i<n;i++)
     printf("\n \t%d\t\t%d",i+1,mp[i]);
printf("\n\nTotal Memory Allocated is %d",ms-temp);
printf("\nTotal External Fragmentation is %d",temp);

getch();
}
```

INPUT
Enter the total memory available (in Bytes) -- 1000
Enter memory required for process 1 (in Bytes) -- 400

Memory is allocated for Process 1
Do you want to continue(y/n) -- y
Enter memory required for process 2 (in Bytes) -- 275
Memory is allocated for Process 2
Do you want to continue(y/n) -- y

Enter memory required for process 3 (in Bytes) -- 550
OUTPUT
Memory is Full
Total Memory Available -- 1000

PROCESS	MEMORY ALLOCATED
1	400
2	275

Total Memory Allocated is 675
Total External Fragmentation is 325

EXPERIMENT 6

OBJECTIVE

Write a C program to simulate paging technique of memory management.

DESCRIPTION

In computer operating systems, paging is one of the memory management schemes by which a computer stores and retrieves data from the secondary storage for use in main memory. In the paging memory-management scheme, the operating system retrieves data from secondary storage in same-size blocks called pages. Paging is a memory-management scheme that permits the physical address space a process to be noncontiguous. The basic method for implementing paging involves breaking physical memory into fixed-sized blocks called frames and breaking logical memory into blocks of the same size called pages. When a process is to be executed, its pages are loaded into any available memory frames from their source.

PROGRAM ::

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


main()
{
int ms, ps, nop, np, rempages, i, j, x, y, pa, offset;
int s[10], fno[10][20];

clrscr();

printf("\nEnter the memory size -- "); scanf("%d",&ms);

printf("\nEnter the page size -- "); scanf("%d",&ps);

nop = ms/ps;
printf("\nThe no. of pages available in memory are -- %d ",nop);

printf("\nEnter number of processes -- "); scanf("%d",&np);
rempages = nop;
for(i=1;i<=np;i++)
{

printf("\nEnter no. of pages required for p[%d]-- ",i);
scanf("%d",&s[i]);

if(s[i] >rempages)
{
```

```
printf("\nMemory is Full");
break;
}
rempages = rempages - s[i];

printf("\nEnter pagetable for p[%d] --- ",i);
for(j=0;j<s[i];j++)
scanf("%d",&fno[i][j]);
}
printf("\nEnter Logical Address to find Physical Address ");
printf("\nEnter process no. and pagenumber and offset -- ");
scanf("%d %d %d",&x,&y, &offset);
    if(x>np || y>=s[i] || offset>=ps)
       printf("\nInvalid Process or Page Number or offset");
else
{
         pa=fno[x][y]*ps+offset;
         printf("\nThe Physical Address is -- %d",pa);
  }
getch();
}
```

INPUT
Enter the memory size – 1000
Enter the page size -- 100
The no. of pages available in memory are -- 10
Enter number of processes -- 3
Enter no. of pages required for p[1]-- 4
Enter pagetable for p[1] --- 8 6 9 5

Enter no. of pages required for p[2] - 5
Enter pagetable for p[2] --- 1 4 5 7 3

Enter no. of pages required for p[3] - 5

OUTPUT
Memory is Full
Enter Logical Address to find Physical Address
Enter process no. and pagenumber and offset -- 2 3 60
The Physical Address is--- 760

EXPERIMENT 7

OBJECTIVE

Write a C program to simulate page replacement algorithms

a) FIFO b) LRU c)optimal

DESCRIPTION

Page replacement is basic to demand paging. It completes the separation between logical memory and physical memory. With this mechanism, an enormous virtual memory can be provided for programmers on a smaller physical memory. There are many different page-replacement algorithms. Every operating system probably has its own replacement scheme. A FIFO replacement algorithm associates with each page the time when that page was brought into memory. When a page must be replaced, the oldest page is chosen. If the recent past is used as an approximation of the near future, then the page that has not been used for the longest period of time can be replaced. This approach is the Least Recently Used (LRU) algorithm. LRU replacement associates with each page the time of that page's last use. When a page must be replaced, LRU chooses the page that has not been used for the longest period of time. Least frequently used (LFU) page-replacement algorithm requires that the page with the smallest count be replaced. The reason for this selection is that an actively used page should have a large reference count

PROGRAM

FIFO PAGE REPLACEMENT ALGORITHM

PROGRAM

FIFO PAGE REPLACEMENT ALGORITHM

```
#include<stdio.h>
#include<conio.h>
main()
{
int i, j, k, f, pf=0, count=0, rs[25], m[10], n;

clrscr();

printf("\n Enter the length of reference string -- ");

scanf("%d",&n);

printf("\n Enter the reference string -- ");

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

```
scanf("%d",&rs[i]);
printf("\n Enter no. of frames -- ");
scanf("%d",&f);
      for(i=0;i<f;i++)
         m[i]=-1;
printf("\n The Page Replacement Process is -- \n");
for(i=0;i<n;i++)
{
     for(k=0;k<f;k++)
      {
        if(m[k]==rs[i])
        break;
    }
if(k==f)
{
  m[count++]=rs[i];.
   p++;
}
for(j=0;j<f;j++)
   printf("\t%d",m[j]);
if(k==f)
   printf("\tPF No. %d",pf);
 printf("\n");
if(count==f)
     count=0;
}
printf("\n The number of Page Faults using FIFO are %d",pf);
getch();
```

INPUT
Enter the length 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
Enter no. of frames -- 3

OUTPUT
The Page Replacement Process is –

7	-1	-1 PF No. 1
7	0-1	PF No. 2
7	01	PF No. 3
2	01	PF No. 4
2	01	
2	31	PF No. 5
2	30	PF No. 6
4	30	PF No. 7
4	20	PF No. 8
4	23	PF No. 9
0	23	PF No. 10
0	23	
0	23	
0	13	PF No. 11
0	12	PF No. 12
0	12	
0	12	
7	12	PF No. 13
7	02	PF No. 14
7	01	PF No. 15

The number of Page Faults using FIFO are 15

LRU PAGE REPLACEMENT ALGORITHM

```
#include<stdio.h>
#include<conio.h>
main()
{
int i, j , k, min, rs[25], m[10], count[10], flag[25], n, f, pf=0, next=1;
clrscr();
printf("Enter the length of reference string -- ");
scanf("%d",&n);
printf("Enter the reference string -- ");
for(i=0;i<n;i++)
{
        scanf("%d",&rs[i]);
                 flag[i]=0;
                }
printf("Enter the number of frames -- ");
scanf("%d",&f);
for(i=0;i<f;i++)
                {
                count[i]=0;
                m[i]=-1;
                }
printf("\nThe Page Replacement process is -- \n"); for(i=0;i<n;i++)
                     {
                    for(j=0;j<f;j++)
                      {
                    if(m[j]==rs[i])
                     {
                flag[i]=1;
                count[j]=next;
                next++;
                }
                }
                if(flag[i]==0)
{
if(i<f)
{
   m[i]=rs[i];
```

```
    count[i]=next;
    next++;

}
else
{
  min=0;
  for(j=1;j<f;j++)
      if(count[min] > count[j])
      min=j;
                m[min]=rs[i];
                count[min]=next;
                next++;
                }
                pf++;
                }
     for(j=0;j<f;j++)
printf("%d\t", m[j]);
                if(flag[i]==0)
                printf("PF No. -- %d" , pf);
                printf("\n");
                }
printf("\nThe number of page faults using LRU are %d",pf);
getch();
}
```

INPUT

Enter the length 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
Enter the number of frames -- 3

OUTPUT

The Page Replacement process is --

7	-1	-1	PF No. -- 1
7	0	-1	PF No. -- 2
7	0	1	PF No. -- 3
2	0	1	PF No. -- 4
2	0	1	
2	0	3	PF No. -- 5
2	0	3	
4	0	3	PF No. -- 6
4	0	2	PF No. -- 7
4	3	2	PF No. -- 8
0	3	2	PF No. -- 9
0	3	2	
0	3	2	
1	3	2	PF No. -- 10
1	3	2	
1	0	2	PF No. -- 11
1	0	2	
1	0	7	PF No. -- 12
1	0	7	
1	0	7	

The number of page faults using LRU are 12

Optimal page replacement algorithms

Optimal page replacement algorithm says that if page fault occurs then that page should be removed that will not be used for **maximum** time in future. It is also known as clairvoyant replacement algorithm or Be lady's optimal page replacement policy

```
#include<stdio.h>

int main()
{
   int no_of_frames, no_of_pages, frames[10], pages[30], temp[10], flag1,
flag2, flag3, i, j, k, pos, max, faults = 0;
   printf("Enter number of frames: ");
   scanf("%d", &no_of_frames);

   printf("Enter number of pages: ");
   scanf("%d", &no_of_pages);

   printf("Enter page reference string: ");

   for(i = 0; i < no_of_pages; ++i){
      scanf("%d", &pages[i]);
   }

   for(i = 0; i < no_of_frames; ++i){
      frames[i] = -1;
   }

   for(i = 0; i < no_of_pages; ++i){
      flag1 = flag2 = 0;

      for(j = 0; j < no_of_frames; ++j){
         if(frames[j] == pages[i]){
               flag1 = flag2 = 1;
               break;
           }
      }

      if(flag1 == 0){
```

```
    for(j = 0; j < no_of_frames; ++j){
       if(frames[j] == -1){
          faults++;
          frames[j] = pages[i];
          flag2 = 1;
          break;
       }
    }
}

if(flag2 == 0){
        flag3 =0;

    for(j = 0; j < no_of_frames; ++j){
        temp[j] = -1;

        for(k = i + 1; k < no_of_pages; ++k){
        if(frames[j] == pages[k]){
          temp[j] = k;
          break;
        }
        }
    }

    for(j = 0; j < no_of_frames; ++j){
        if(temp[j] == -1){
        pos = j;
        flag3 = 1;
        break;
        }
    }

    if(flag3 ==0){
        max = temp[0];
        pos = 0;

        for(j = 1; j < no_of_frames; ++j){
        if(temp[j] > max){
          max = temp[j];
          pos = j;
```

```
                }
                }
            }

                frames[pos] = pages[i];
                faults++;
        }

        printf("\n");

        for(j = 0; j < no_of_frames; ++j){
            printf("%d\t", frames[j]);
        }
    }

    printf("\n\nTotal Page Faults = %d", faults);

    return 0;
}
```

Output

```
Enter number of frames: 3
Enter number of pages: 10
Enter page reference string: 2 3 4 2 1 3 7 5 4 3

2 -1 -1
2 3  -1
2 3  4
2 3  4
1 3  4
1 3  4
7 3  4
5 3  4
5 3  4
5 3  4
```

EXPERIMENT 8

OBJECTIVE

Write a C program to simulate disk scheduling algorithms:

One of the responsibilities of the operating system is to use the hardware efficiently. For the disk drives, meeting this responsibility entails having fast access time and large disk bandwidth. Both the access time and the bandwidth can be improved by managing the order in which disk I/O requests are serviced which is called as disk scheduling. The simplest form of disk scheduling is, of course, the first-come, first-served (FCFS) algorithm. This algorithm is intrinsically fair, but it generally does not provide the fastest service. In the SCAN algorithm, the disk arm starts at one end, and moves towards the other end, servicing requests as it reaches each cylinder, until it gets to the other end of the disk. At the other end, the direction of head movement is reversed, and servicing continues. The head continuously scans back and forth across the disk. C-SCAN is a variant of SCAN designed to provide a more uniform wait time. Like SCAN, C-SCAN moves the head from one end of the disk to the other, servicing requests along the way. When the head reaches the other end, however, it immediately returns to the beginning of the disk without servicing any requests on the return trip

PROGRAM

```
#include<conio.h>
#include<stdio.h>
int main()
{
 int i,j,sum=0,n;
 int ar[20],tm[20];
 int disk;
 clrscr();
 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",&ar[i]);
        tm[i]=disk-ar[i];
 if(tm[i]<0)
        {
        tm[i]=ar[i]-disk;
        }
 disk=ar[i];
 sum=sum+tm[i];
 }
                /*for(i=0;i<n;i++)
```

```
                {
                printf("\n%d",tm[i]);
                }   */
 printf("\nmovement of total cylinders %d",sum);
 getch();
 return 0;
}
```

Output

```
Enter the number of location 8
Enter position of head 53
Enter the elements of disk queue
98
183
37
122
14
124
65
```

Shortest Seek Time First (SSTF)

In **SSTF** (**Shortest Seek Time First**), requests having **shortest seek time** are executed **first**. So, the **seek time** of every request is calculated in advance in queue and then they are scheduled according to their calculated **seek time**. As a result, the request near the disk arm will get executed **first.**

Program:

```
#include<conio.h>
#include<stdio.h>
struct di
{
int num;
int flag;
}
int main()
{
 int i,j,sum=0,n,min,loc,x,y;
 struct di d[20];
 int disk;
 int ar[20],a[20];
 clrscr();
 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].num);
d[i]. flag=0;
 }
 for(i=0;i<n;i++)
 {                x=0;
 min=0;loc=0;
  for(j=0;j<n;j++)
  {
   if(d[j].flag==0)
   {
   if(x==0)
```

```
  {
  ar[j]=disk-d[j].num;
  if(ar[j]<0){ ar[j]=d[j].num-disk;}
  min=ar[j];loc=j;x++; }
  else
  {
  ar[j]=disk-d[j].num;
  if(ar[j]<0){ ar[j]=d[j].num-disk;}
   }
   if(min>ar[j]){ min=ar[j]; loc=j;}
  }
 }
    d[loc].flag=1;
    a[i]=d[loc].num-disk;
    if(a[i]<0){a[i]=disk-d[loc].num;}

    disk=d[loc].num;
}

 for(i=0;i<n;i++)
{
sum=sum+a[i];
}
      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
98
183
37
122
14
124
65
67
movement of total cylinders 236

EXPERIMENT 9

SCAN DISK SCHEDULING ALGORITHM

```
#include<stdio.h>
main()
{
int t[20], d[20], h, i, j, n, temp, k, atr[20], tot, p, sum=0;

clrscr();
printf("enter the no of tracks to be traveresed");
scanf("%d"',&n);
printf("enter the position of head");
scanf("%d",&h);
t[0]=0;
t[1]=h;
printf("enter the tracks");
for(i=2;i<n+2;i++)
   scanf("%d",&t[i]);
for(i=0;i<n+2;i++)
{
for(j=0;j<(n+2)-i-1;j++)
{
if(t[j]>t[j+1])
{
temp=t[j];
t[j]=t[j+1];
t[j+1]=temp;
    }
```

```
  }
}
for(i=0;i<n+2;i++)
if(t[i]==h)
j=i;
k=i;
p=0;
while(t[j]!=0)
{
   atr[p]=t[j];
   j--;
   p++;
}
 atr[p]=t[j];
   for(p=k+1;p<n+2;p++,k++)

atr[p]=t[k+1];
  for(j=0;j<n+1;j++)
 {

if(atr[j]>atr[j+1])
     d[j]=atr[j]-atr[j+1];
else
     d[j]=atr[j+1]-atr[j];
sum+=d[j];
}
printf("\nAverage header movements:%f",(float)sum/n);
getch();
```

}

INPUT

Enter no.of tracks:9

Enter track position:55 58 60 70 18 90 150 160 184

OUTPUT

Tracks traversed	Difference between tracks
160	10
184	24
90	94
70	20
60	10
58	2
55	3
18	37

Average header movements: 27.77

LOOK Disk Scheduling Algorithm

Program

```
#include<math.h>
#include<stdio.h>
int main()
{
   int      i,n,j=0,k=0,x=0,l,req[50],mov=0,cp,ub,end,      lower[50],upper[50],
temp,a[50];
   printf("enter the current position\n");
   scanf("%d",&cp);
   printf("enter the number of requests\n");
   scanf("%d",&n);
   printf("enter the request order\n");
   for(i=0;i<n;i++)
   {
      scanf("%d",&req[i]);
   }
   printf("Enter the upper bound\n");
   scanf("%d",&ub);

   /*break the request array into two arrays : one with requests lower than
current and other with requests higher than current position. Also sort these two
new arrays*/
   for(i=0;i<n;i++)
   {
      if(req[i]<cp)
      {
         lower[j]=req[i];
         j++;
      }
      if(req[i]>cp)
      {
         upper[k]=req[i];
         k++;
      }
   }

   //sort the lower array in reverse order
   for(i=0;i<j;i++)
```

```
{
   for(l=0;l<j-1;l++)
   {
      if(lower[l]<lower[l+1])
      {
         temp=lower[l];
         lower[l]=lower[l+1];
         lower[l+1]=temp;
      }
   }
}

// sort the upper array in ascending order
for(i=0;i<=k;i++)
{
   for(l=0;l<k-1;l++)
   {
      if(upper[l]>upper[l+1])
      {
         temp=upper[l];
         upper[l]=upper[l+1];
         upper[l+1]=temp;
      }
   }
}

printf("Enter the end to which the head is moving (0 - for lower end(zero) and
1 - for upper end\n");
scanf("%d",&end);
switch(end)
{
   case 0:
      for(i=0;i<j;i++)
      {
         a[x]=lower[i];
         x++;
      }

      for(i=0;i<k;i++)
      {
```

```
            a[x]=upper[i];
            x++;
          }
        break;
        case 1:
          for(i=0;i<k;i++)
          {
            a[x]=upper[i];
            x++;
          }

          for(i=0;i<j;i++)
          {
            a[x]=lower[i];
            x++;
          }
        break;
    }

    mov=mov+abs(cp-a[0]);
    printf("%d -> %d",cp,a[0]);
    for(i=1;i<x;i++)
    {
      mov=mov+abs(a[i]-a[i-1]);
      printf(" -> %d",a[i]);
    }
    printf("\n");
    printf("total head movement = %d\n",mov);
}
```

Output:

```
Enter the current position
3
Enter the no of requests 2
Enter the request order 1 2
Enter the upper bound 4
Enter the end to which the head is moving (0- for lower end(zero) and 1- for upper end
5
3-> 1
```

EXPERIMENT 10

OBJECTIVE

Write a C program to simulate the following file allocation strategies.
a)Sequential b) Linked c))Indexed

DESCRIPTION

A file is a collection of data, usually stored on disk. As a logical entity, a file enables to divide data into meaningful groups. As a physical entity, a file should be considered in terms of its organization. The term "file organization" refers to the way in which data is stored inafileand,consequently,themethod(s)bywhich itcan be accessed.

SEQUENTIAL FILE ALLOCATION

In this file organization, the records of the file are stored one after another both physically and logically. Thatis, record with sequence number 16 is located just after the 15th record. Arecordofasequential filecanonly be accessed by reading all the previous records.

LINKED FILE ALLOCATION

With linked allocation, each file is a linked list of disk blocks; the disk blocks may be scattered anywhere onthe disk. Thedirectory contains a pointer to the first and lastblocks ofthe file. Each blockcontains apointer to the next block.

INDEXED FILE ALLOCATION

Indexed file allocation strategy brings all the pointers together into one location: an index block. Each file has its own index block, which is an array of disk-block addresses. The i^{th} entry in the index block points to the i^{th} block of the file. Thedirectory contains the address of the index block. To find and read the i^{th} block, the pointer in the i^{th} index-block entry is used.

PROGRAM

SEQUENTIAL FILE ALLOCATION

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

struct fileTable
{
char name[20];
int sb, nob;
}
ft[30];
void main()
{
int i, j, n;
char s[20];
clrscr();
printf("Enter no of files  :");
```

```
scanf("%d",&n);
for(i=0;i<n;i++)
{
		printf("\nEnter file name %d	:",i+1);
		scanf("%s",ft[i].name);
		printf("Enter starting block of file %d	:",i+1);
		scanf("%d",&ft[i].sb);
		printf("Enter no of blocks in file %d :",i+1);
		scanf("%d",&ft[i].nob);
		}
printf("\nEnter the file name to be searched -- ");
scanf("%s",s);
for(i=0;i<n;i++)
		if(strcmp(s, ft[i].name)==0)
		break;
if(i==n)
printf("\nFile Not Found");
else
{
		printf("\nFILE NAME START BLOCK NO OF BLOCKS
BLOCKS OCCUPIED\n");
		printf("\n%s\t\t%d\t\t%d\t",ft[i].name,ft[i].sb,ft[i].nob);
		for(j=0;j<ft[i].nob;j++)
		  printf("%d, ",ft[i].sb+j);
		}
getch();
}
```

INPUT:
Enter no of files :3
Enter file name 1 :A
Enter starting block of file 1 :85
Enter no of blocks in file 1 :6

Enter file name 2 :B
Enter starting block of file 2 :102
Enter no of blocks in file 2 :4

Enter file name 3 :C
Enter starting block of file 3 :60
Enter no of blocks in file 3 :4
Enter the file name to be searched -- B

OUTPUT:

FILE NAME	START BLOCK	NO OF BLOCKS	BLOCKS OCCUPIED
B	102	4	102, 103, 104, 105

LINKED FILE ALLOCATION

```
#include<stdio.h>
#include<conio.h>
struct fileTable
struct fileTable.
{
char name[20];
int nob;
struct block *sb;
}
ft[30];

struct block
{
        int bno;
        struct block *next;
        };
void main()
{
int i, j, n;
char s[20];
struct block *temp;
clrscr();
printf("Enter no of files    :");
scanf("%d",&n);
for(i=0;i<n;i++)
{
        printf("\nEnter file name %d       :",i+1);
        scanf("%s",ft[i].name);
        printf("Enter no of blocks in file %d :",i+1);
        scanf("%d",&ft[i].nob);
ft[i].sb=(struct block*)malloc(sizeof(struct block));
temp = ft[i].sb;
printf("Enter the blocks of the file :");
scanf("%d",&temp->bno);
temp->next=NULL;

        for(j=1;j<ft[i].nob;j++)
        {
        temp->next = (struct block*)malloc(sizeof(struct block));
```

```
                temp = temp->next;
                scanf("%d",&temp->bno);
                }
                temp->next = NULL;
}
printf("\nEnter the file name to be searched -- ");
scanf("%s",s);
for(i=0;i<n;i++)
if(strcmp(s, ft[i].name)==0)
                break;
if(i==n)
printf("\nFile Not Found");
else
{
                printf("\nFILE NAME NO OF BLOCKS BLOCKS OCCUPIED");
                printf("\n %s\t\t%d\t",ft[i].name,ft[i].nob);
                temp=ft[i].sb;
                for(j=0;j<ft[i].nob;j++)
                {
                printf("%d □ ",temp->bno);
                temp = temp->next;
}
}
getch();
}
```

INPUT:

Enter no of files 2

Enter file 1 : A

Enter no of blocks in file 1 4

Enter the blocks of the file 1 12 23 9 4

Enter file 2 : G

Enter no of blocks in file 2 5

Enter the blocks of the file 2 : 88 77 66 55 44 Enter the file to be searched : G

OUTPUT:

FILE NAME	NO OF BLOCKS	BLOCKS OCCUPIED
G	5	88 → 77 → 66 → 55 → 44

3. INDEXED FILE ALLOCATION

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

struct fileTable
{
char name[20];
int nob, blocks[30];
}ft[30];

void main()
{
int i, j, n;
char s[20];
clrscr();
printf("Enter no of files    :");
scanf("%d",&n);
for(i=0;i<n;i++)
{
        printf("\nEnter file name %d       :",i+1);
        scanf("%s",ft[i].name);
        printf("Enter no of blocks in file %d :",i+1); scanf("%d",&ft[i].nob);
        printf("Enter the blocks of the file :"); for(j=0;j<ft[i].nob;j++)
                scanf("%d",&ft[i].blocks[j]);
}
printf("\nEnter the file name to be searched -- ");
scanf("%s",s);
for(i=0;i<n;i++)
        if(strcmp(s, ft[i].name)==0)
        break;
if(i==n)
        printf("\nFile Not Found");
else
{
printf("\nFILE NAME NO OF BLOCKS BLOCKS OCCUPIED");
printf("\n %s\t\t%d\t",ft[i].name,ft[i].nob);
for(j=0;j<ft[i].nob;j++)
        printf("%d, ",ft[i].blocks[j]);
}
```

```
getch();
}
```

INPUT:
Enter no of files 2
Enter file 1 : A
Enter no of blocks in file 1 4
Enter the blocks of the file 1 12 23 9 4

Enter file 2 : G
Enter no of blocks in file 2 5
Enter the blocks of the file 2 88 77 66 55 44
Enter the file to be searched : G

OUTPUT:

FILENAME	NOOFBLOCKS	BLOCKS OCCUPIED
G	5	88, 77, 66, 55, 44

EXPERIMENT 11

OBJECTIVE

Write a C program to simulate the following file organization techniques

a) Single level directory b) Two level directory

DESCRIPTION

The directory structure is the organization of files into a hierarchy of folders. In a single-level directory system, all the files are placed in one directory. There is a root directory which has all files. It has a simple architecture and there are no sub directories. Advantage of single level directory system is that it is easy to find a file in the directory. In the two-level directory system, each user has own user file directory (UFD). The system maintains a master block that has one entry for each user. This master block contains the addresses of the directory of the users. When a user job starts or a user logs in, the system's master file directory (MFD) is searched. When a user refers to a particular file, only his own UFD is searched. This effectively solves the name collision problem and isolates users from one another. Hierarchical directory structure allows users to create their own subdirectories and to organize their files accordingly. A tree is the most common directory structure. The tree has a root directory, and every file in the system has a unique path name. A directory (or subdirectory) contains a set of files or subdirectories

PROGRAM

SINGLE LEVEL DIRECTORY ORGANIZATION

```
#include<stdio.h>
struct
{
char dname[10],fname[10][10];
int fcnt;
}dir;
void main()
{
        int i,ch;
        char f[30];
        clrscr();
        dir.fcnt = 0;
        printf("\nEnter name of directory -- ");
        scanf("%s", dir.dname);
while(1)
{
printf("\n\n1. Create File\t2. Delete File\t3. Search File \n
```

```
                4. Display Files\t5. Exit\nEnter your choice -- ");
scanf("%d",&ch);
switch(ch)
{
                case 1: printf("\nEnter the name of the file -- ");
                scanf("%s",dir.fname[dir.fcnt]); dir.fcnt++;
                break;
                case 2: printf("\nEnter the name of the file -- "); scanf("%s",f);
                for(i=0;i<dir.fcnt;i++)
                {
                        if(strcmp(f, dir.fname[i])==0)
                        {
                           printf("File %s is deleted ",f);
                           strcpy(dir.fname[i],dir.fname[dir.fcnt-1]);
                           break;
                           }
                }

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

                else
                {
```

```
                printf("\nThe Files are -- ");
                for(i=0;i<dir.fcnt;i++)
                printf("\t%s",dir.fname[i]);

            }
            break;
            default: exit(0);
        }
}
getch();
}
```

Output

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

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

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

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

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

Enter your choice – 3
Enter the name of the file – ABC
File ABC not found
```

1. Create File 2. Delete File 3. Search File
4. Display Files 5. Exit

Enter your choice – 2
Enter the name of the file – B
File B is deleted

1. Create File 2. Delete File 3. Search File
4. Display Files 5. Exit

Enter your choice – 5

TWO LEVEL DIRECTORY ORGANIZATION

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

        void main()
        {
        int i,ch,dcnt,k;
        char f[30], d[30];
        clrscr();
        dcnt=0;
while(1)
{
        printf("\n\n1. Create Directory\t2. Create File\t3. Delete File");
        printf("\n4. Search File\t\t5. Display\t6. Exit\t Enter your choice --       ");


scanf("%d",&ch);
switch(ch)
{
        case 1: printf("\nEnter name of directory -- ");
                scanf("%s", dir[dcnt].dname);
                dir[dcnt].fcnt=0;
                dcnt++;
                printf("Directory created");
                break;
        case 2: printf("\nEnter name of the directory -- "); scanf("%s",d);
                for(i=0;i<dcnt;i++)
                if(strcmp(d,dir[i].dname)==0)
                {
        `               printf("Enter name of the file -- ");
                        scanf("%s",dir[i].fname[dir[i].fcnt]);
                        dir[i].fcnt++;
                        printf("File created");
                        break;
                        }
                if(i==dcnt)
```

```
                printf("Directory %s not found",d);
                break;
        case 3: printf("\nEnter name of the directory -- ");
                scanf("%s",d);
                for(i=0;i<dcnt;i++)
                {
                        if(strcmp(d,dir[i].dname)==0)
                        {
                        printf("Enter name of the file -- ");
                        scanf("%s",f);
                        for(k=0;k<dir[i].fcnt;k++)
                                {
                                if(strcmp(f, dir[i].fname[k])==0)
                                {
                                printf("File %s is deleted ",f);
                                dir[i].fcnt--;
                                strcpy(dir[i].fname[k],dir[i].fname[dir[i].fcnt]); goto
jmp;
                                }
                          }
                          printf("File %s not found",f);
                          goto jmp;
                          }
                        }
                        printf("Directory %s not found",d);
                        jmp : break;
        case 4: printf("\nEnter name of the directory -- ");
                scanf("%s",d);
                for(i=0;i<dcnt;i++)
                        {
                        if(strcmp(d,dir[i].dname)==0)
                        {
                        printf("Enter the name of the file -- ");
                        scanf("%s",f);
                        for(k=0;k<dir[i].fcnt;k++)
                        {
                        if(strcmp(f, dir[i].fname[k])==0)
                          {
                          printf("File %s is found ",f); goto jmp1;
                          }
```

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

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

OUTPUT:

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

Enter your choice --        1
Enter name of directory---DIR1
Directory created

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

Enter your choice----- 1
```

Enter name of directory -- DIR2
Directory created

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

Enter name of the directory – DIR1
Enter name of the file -- A1
File created

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

Enter name of the directory – DIR1
Enter name of the file -- A2
File created

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

Enter name of the directory – DIR2
Enter name of the file -- B1
File created

1. Create Directory
4. Search File 2. Create File 5. Display 3. Delete File 6. Exit
3Enter your choice --
5
Directory DIR1 Files A1
A2
DIR2 B1
1. Create Directory
4. Search File 2. Create File 5. Display 3. Delete File
6. Exit
Enter your choice -- 4

Enter name of the directory – DIR Directory not found

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

Enter name of the directory – DIR1 Enter name of the file -- A2
File A2 is deleted

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

EXPERIMENT 12
HIERARCHICAL DIRECTORY RGANIZATION

```
#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]);

}
}
```

INPUT

```
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): USER1
Enter 1 for Dir/2 for File: 1
```

No of subdirectories/files(for USER1): 1

Enter Name of dir/file(under USER1): SUBDIR1 .
Enter 1 for Dir/2 for File: 1
No of subdirectories/files(for SUBDIR1): 2

Enter Name of dir/file(under USER1): JAVA
Enter 1 for Dir/2 for File: 1
No of subdirectories/files(for JAVA): 0 .

Enter Name of dir/file(under SUBDIR1): 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): SUBDIR2
Enter 1 for Dir/2 for File: 1
No of subdirectories/files(for SUBDIR2): 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

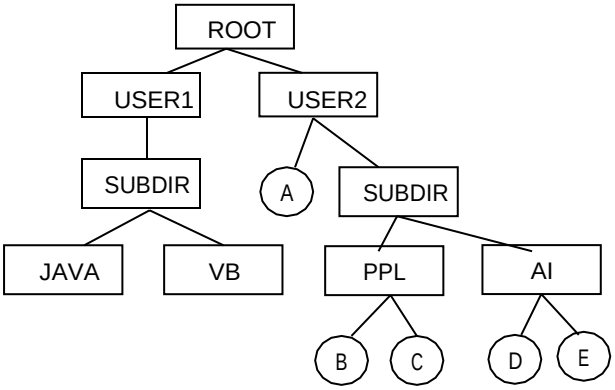
Enter Name of dir/file(under PPL): C
Enter 1 for Dir/2 for File: 2

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



```
#include<stdio.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;
main()
{
        int gd=DETECT,gm;
        root=NULL;
        create(&root,0,"root",0,639,320);
        read_links();
        clrscr();
        initgraph(&gd,&gm,"c:\\tc\\BGI");
        draw_link_lines();
        display(root);
        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]);
}}}
```