1. A.) **FCFS**.

CODE:

#include<stdio.h>

void main(){

    int bt[20],wt[20],tat[20],i,n;

    float wtavg =0,tatavg=0;

    int tottime =0;

    printf("Enter the number of processes: ");

    scanf("%d",&n);

    for(i=0;i<n;i++){

        printf("\nEnter burst time for processes %d: ",i);

        scanf("%d",&bt[i]);

        }

        for(i=0;i<n;i++){

            tottime+=bt[i];

            tat[i]=tottime;

            wt[i]=tat[i]-bt[i];

            wtavg=wtavg+wt[i];

            tatavg=tatavg+tat[i];

        }

        printf("\t PROCESS \t BURST TIME\t WATING 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 Wating Time: %f\n",wtavg/n);

            printf("\n Average Turnaround Time: %f\n",tatavg/n);

}

OUTPUT:

Enter the number of processes: 3

Enter burst time for processes 0: 24

Enter burst time for processes 1: 3

Enter burst time for processes 2: 3

PROCESS BURST TIME WATING TIME TURNAROUND TIME

p0 24 0 24

p1 3 24 27

p2 3 27 30

Average Wating Time: 17.000000

Average Turnaround Time: 27.000000

1. B.) **SJF**.

CODE:

#include<stdio.h>

void **main**(){

    int p[20],bt[20],wt[20],tat[20],i,k,n,temp;

    float wtavg=0,tatavg=0;

    int tottime=0;

**printf**("Enter the number of processes:");

**scanf**("%d",&n);

    for(i=0;i<n;i++) {

        p[i]=i;

**printf**("Enter burst time for process %d:",i);

**scanf**("%d",&bt[i]);

    }

    for(i=0;i<n;i++) {

        for(k=i+1;k<n;k++) {

            if(bt[i]>bt[k]) {

                temp=bt[i];

                bt[i]=bt[k];

                bt[k]=temp;

                temp=p[i];

                p[i]=p[k];

                p[k]=temp;

            }

        }

    }

    for(i=0;i<n;i++) {

        tottime+=bt[i];

        tat[i]=tottime;

        wt[i]=tottime-bt[i];

        wtavg+=wt[i];

        tatavg+=tat[i];

    }

**printf**("\tPROCESS\tBURST TIME\tWAITING TIME\tTURNAROUND TIME\n");

    for(i=0;i<n;i++) {

**printf**("\n\tp%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);

}

OUTPUT:

Enter the number of processes:3

Enter burst time for process 0:5

Enter burst time for process 1:2

Enter burst time for process 2:3

PROCESS BURST TIME WAITING TIME TURNAROUND TIME

p1 2 0 2

p2 3 2 5

p0 5 5 10

Average Waiting Time=2.333333

Average Turnaround Time=5.666667

2.) **Producer-Consumer**.

CODE:

#include<stdio.h>

#include<stdlib.h>

#define **buffsize** 3

int in=0,out=0,buff[10],nextP,nextC,mutex=1,full=0,empty=**buffsize**;

int **wait**(int s){

    return (--s);

}

int **signal**(int s){

    return (++s);

}

void **producer**(){

**printf**("Enter the item produced:");

**scanf**("%d",&nextP);

    empty=**wait**(empty);

    mutex=**wait**(mutex);

    buff[in]=nextP;

    in=(in+1)%**buffsize**;

    mutex=**signal**(mutex);

    full=**signal**(full);

}

void **consumer**() {

    full=**wait**(full);

    mutex=**wait**(mutex);

    nextC=buff[out];

**printf**("\nConsumer consumes item %d",nextC);

    out=(out+1)%**buffsize**;

    mutex=**signal**(mutex);

    empty=**signal**(empty);

}

void **main**() {

    int n;

**printf**("\n1.Producer\n2.Consumer\n3.Exit");

    while(1) {

**printf**("\nEnter your choice:");

**scanf**("%d",&n);

        switch(n) {

            case 1:

                if((mutex==1)&&(empty!=0))

**producer**();

                else

**printf**("Buffer is full");

                break;

            case 2:

                if((mutex==1)&&(full!=0))

**consumer**();

                else

**printf**("Buffer is empty!");

                break;

            case 3:

                return 0;

        }

    }

}

OUTPUT:

1.Producer

2.Consumer

3.Exit

Enter your choice:1

Enter the item produced:10

Enter your choice:1

Enter the item produced:20

Enter your choice:1

Enter the item produced:30

Enter your choice:2

Consumer consumes item 10

Enter your choice:2

Consumer consumes item 20

Enter your choice:2

Consumer consumes item 30

Enter your choice:2

Buffer is empty!

3.) **Dining-Philosopher**.

CODE:

#include<stdio.h>

#include<pthread.h>

#include<unistd.h>

#include<stdint.h>

#define **NUM\_PHILOSOPHERS** 5

#define **EATING** 0

#define **THINKING** 1

#define **HUNGRY** 2

pthread\_mutex\_t forks[**NUM\_PHILOSOPHERS**];

int philosophers[**NUM\_PHILOSOPHERS**] ={**THINKING**};

void\* **philosopher**(void\* arg) {

    int id=\*(int\*)arg;

    int left\_fork=id;

    int right\_fork=(id+1)%**NUM\_PHILOSOPHERS**;

    while(1){

**printf**("Philosopher %d is thinking.\n",id);

**sleep**(1);

        philosophers[id]=**HUNGRY**;

**printf**("Philosopher %d is hungry.\n",id);

**pthread\_mutex\_lock**(&forks[left\_fork]);

**printf**("Philosopher %d picked up left fork.\n",id);

        if (**pthread\_mutex\_trylock**(&forks[right\_fork])==0) {

**printf**("Philosopher %d picked up right fork.\n",id);

            philosophers[id]=**EATING**;

**printf**("Philosopher %d is eating.\n",id);

**sleep**(2);

**pthread\_mutex\_unlock**(&forks[right\_fork]);

**pthread\_mutex\_unlock**(&forks[left\_fork]);

        } else{

**printf**("Philosopher %d couldn't pick up right fork. Putting down left fork.\n",id);

**pthread\_mutex\_unlock**(&forks[left\_fork]);

        }

    }

}

int **main**(){

    pthread\_t tid[**NUM\_PHILOSOPHERS**];

    int ids[**NUM\_PHILOSOPHERS**],i;

    for(i=0;i<**NUM\_PHILOSOPHERS**;i++){

**pthread\_mutex\_init**(&forks[i],**NULL**);

    }

    for(i=0;i<**NUM\_PHILOSOPHERS**;i++){

        ids[i]=i;

**pthread\_create**(&tid[i],**NULL**,**philosopher**,&ids[i]);

    }

    for(i=0;i<**NUM\_PHILOSOPHERS**;i++){

**pthread\_join**(tid[i], **NULL**);

    }

    for(i=0;i<**NUM\_PHILOSOPHERS**;i++){

**pthread\_mutex\_destroy**(&forks[i]);

    }

    return 0;

}

OUTPUT:

Philosopher 0 is thinking.

Philosopher 2 is thinking.

Philosopher 4 is thinking.

Philosopher 3 is thinking.

Philosopher 1 is thinking.

Philosopher 0 is hungry.

Philosopher 4 is hungry.

Philosopher 2 is hungry.

Philosopher 4 picked up left fork.

Philosopher 4 couldn't pick up right fork. Putting down left fork.

Philosopher 2 picked up left fork.

Philosopher 3 is hungry.

Philosopher 4 is thinking.

Philosopher 3 picked up left fork.

Philosopher 3 picked up right fork.

Philosopher 3 is eating.

Philosopher 1 is hungry.

Philosopher 1 picked up left fork.

Philosopher 1 couldn't pick up right fork. Putting down left fork.

Philosopher 1 is thinking.

Philosopher 2 couldn't pick up right fork. Putting down left fork.

Philosopher 0 picked up left fork.

Philosopher 0 picked up right fork.

Philosopher 0 is eating.

Philosopher 2 is thinking.

4.) A.) **MFT**.

CODE:

#include <stdio.h>

int **main**(){

    int tm,om,n,i,block\_size,internal\_frag=0,ex\_frag,waste;

**printf**("Enter total memory size, memory for OS, No. of processes: ");

**scanf**("%d %d %d",&tm,&om,&n);

    int process[n];

    tm-=om;

    block\_size=tm/n;

    ex\_frag=tm-block\_size\*n;

    for(i=0;i<n;i++) {

**printf**("Enter size of process %d: ",i+1);

**scanf**("%d",&process[i]);

        if(process[i]<= block\_size){

            waste=block\_size-process[i];

            internal\_frag+=waste;

**printf**("Memory allocated to process %d is: %d\n",i+1,process[i]);

        }else{

**printf**("Process %d is blocked, no sufficient memory available\n",i+1);

            ex\_frag+=block\_size;

        }

    }

**printf**("Total memory for user space = %d.\n",tm);

**printf**("External Fragmentation is %d.\n",ex\_frag);

**printf**("Internal Fragmentation is %d\n",internal\_frag);

    return 0;

}

OUTPUT:

Enter total memory size, memory for OS, No. of processes: 100 50 3

Enter size of process 1: 10

Memory allocated to process 1 is: 10

Enter size of process 2: 20

Process 2 is blocked, no sufficient memory available

Enter size of process 3: 15

Memory allocated to process 3 is: 15

Total memory for user space = 50.

External Fragmentation is 18.

Internal Fragmentation is 7

4.) B.) **MVT**.

CODE:

#include<stdio.h>

#include<string.h>

int **main**(){

    int tm,mp[10],i,pid[50],temp,n=0,osm,allotedpc=0,allotedp[50],flag[50];

    char ch='y';

**printf**("Enter the total memory available (in Bytes): ");

**scanf**("%d",&tm);

**printf**("Enter the memory required by OS (in Bytes): ");

**scanf**("%d",&osm);

    tm-=osm;

    temp=tm;

    for(i=0;i<50;i++) {

        flag[i]=0;

    }

    for(i=0;ch=='y';i++,n++){

**printf**("Enter memory required for process %d (in Bytes): ",i+1);

**scanf**("%d",&mp[i]);

        if(mp[i]<=temp){

**printf**("Memory is allocated for Process %d\n",i+1);

            temp-=mp[i];

            flag[i]=1;

        }else{

**printf**("No sufficient memory available\n");

            flag[i]=0;

        }

**printf**("Do you want to continue (y/n): ");

**scanf**(" %c",&ch);

    }

**printf**("\nTotal Memory Available: %d",tm);

**printf**("\n\n\tPROCESS\t\tMEMORY ALLOCATED\n");

    for(i=0;i<n;i++) {

        if(flag[i]==0) continue;

**printf**("\t%d\t\t%d\n",i+1,mp[i]);

    }

**printf**("Total Memory Allocated: %d\n",tm-temp);

**printf**("Total External Fragmentation: %d\n",temp);

    return 0;

}

OUTPUT:

Enter the total memory available (in Bytes): 100

Enter the memory required by OS (in Bytes): 50

Enter memory required for process 1 (in Bytes): 3

Memory is allocated for Process 1

Do you want to continue (y/n): y

Enter memory required for process 2 (in Bytes): 10

Memory is allocated for Process 2

Do you want to continue (y/n): y

Enter memory required for process 3 (in Bytes): 20

Memory is allocated for Process 3

Do you want to continue (y/n): y

Enter memory required for process 4 (in Bytes): 15

Memory is allocated for Process 4

Do you want to continue (y/n): n

Total Memory Available: 50

PROCESS MEMORY ALLOCATED

1 3

2 10

3 20

4 15

Total Memory Allocated: 48

Total External Fragmentation: 2

5.) **BANKER’S ALGORITHM.**

CODE:

#include<stdio.h>

#define **PROCESSES** 10

#define **RESOURCES** 10

int available[**RESOURCES**],work[**RESOURCES**];

int max[**PROCESSES**][**RESOURCES**];

int allocation[**PROCESSES**][**RESOURCES**];

int need[**PROCESSES**][**RESOURCES**];

int p,r,finish[**PROCESSES**];

int **safety**(){

    int safeSequence[**PROCESSES**];

    int count=0,j,i,k,flag=0;

    for (i=0;i<r;i++) {

        work[i]=available[i];

    }

    for(i=0;i<p;i++){

        finish[i]=0;

    }

    while(count<p){

        flag=0;

        for(i=0;i<p;i++){

            if(!finish[i]){

                for(j=0;j<r;j++){

                    if(need[i][j]>work[j])

                        break;

                }

                if(j==r){

                    for(k=0;k<r;k++){

                        work[k]+=allocation[i][k];

                    }

                    safeSequence[count++]=i;

                    finish[i]=1;

                    flag=1;

                }

            }

        }

        if(!flag){

**printf**("System is in unsafe state!\n");

            return 0;

        }

    }

**printf**("Safe sequence: ");

    for(i=0;i<p-1;i++)

**printf**("P%d -> ",safeSequence[i]);

**printf**("P%d\n",safeSequence[p-1]);

    return 1;

}

int **requestResources**(int process,int request[]){

    for(int i=0;i<r;i++){

        if(request[i]>need[process][i]||request[i]>available[i])

            return 0;

    }

    for (int i=0;i<r;i++){

        available[i]-=request[i];

        allocation[process][i]+=request[i];

        need[process][i]-=request[i];

    }

    if(!**safety**()){

        for(int i=0;i<r;i++){

            available[i]+=request[i];

            allocation[process][i]-=request[i];

            need[process][i]+=request[i];

        }

        return 0;

    }

    return 1;

}

int **main**(){

    int process;

    int request[**RESOURCES**];

**printf**("Enter the number of processes: ");

**scanf**("%d",&p);

**printf**("Enter the number of resources: ");

**scanf**("%d",&r);

**printf**("Enter the maximum demand of each process:\n");

    for(int i=0;i<p;i++){

**printf**("For process %d: ",i);

        for(int j=0;j<r;j++){

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

        }

    }

**printf**("Enter the allocation for each process:\n");

    for(int i=0;i<p;i++){

**printf**("For process %d: ",i);

        for(int j=0;j<r;j++){

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

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

        }

    }

**printf**("Enter the available resources: ");

    for(int i=0;i<r;i++){

**scanf**("%d",&available[i]);

    }

**safety**();

**printf**("\nEnter the process number making the request: ");

**scanf**("%d",&process);

**printf**("Enter the request for each resource: ");

    for(int i=0;i<r;i++){

**scanf**("%d",&request[i]);

    }

    if(**requestResources**(process,request)){

**printf**("Request granted!\n");

    }else{

**printf**("Request denied! System will be in unsafe state.\n");

    }

    return 0;  
}

OUTPUT:

Enter the number of processes: 5

Enter the number of resources: 3

Enter the maximum demand of each process:

For process 0: 7 5 3

For process 1: 3 2 2

For process 2: 9 0 2

For process 3: 2 2 2

For process 4: 4 3 3

Enter the allocation for each process:

For process 0: 0 1 0

For process 1: 2 0 0

For process 2: 3 0 2

For process 3: 2 1 1

For process 4: 0 0 2

Enter the available resources: 3 3 2

Safe sequence: P1 -> P3 -> P4 -> P0 -> P2

Enter the process number making the request: 1

Enter the request for each resource: 1 0 2

Safe sequence: P1 -> P3 -> P4 -> P0 -> P2

Request granted!

6.) **PAGING.**

CODE:

#include<stdio.h>

void **main**(){

    int ms,fs,nof,np,rempages,i,j,x,y,pa,offset;

    int s[10],fno[10][20];

**printf**("\nenter the memory size--");

**scanf**("%d",&ms);

**printf**("\nenter the page size--");

**scanf**("%d",&fs);

    nof=ms/fs;

**printf**("\nno. of pages available in memory are--%d",nof);

**printf**("\nenter number of processes--");

**scanf**("%d",&np);

    rempages=nof;

    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 page table for p[%d] (frame numbers <%d)---",i,nof);

        for(j=0;j<s[i];j++){

**printf**("\nenter frame number for page number %d:",j);

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

        }

    }

**printf**("\nenter logical address to find physical address");

**printf**("\nenter process no. and page number and offset--");

**scanf**("%d%d%d",&x,&y,&offset);

    if(x<np){

        for(i=1;i<=np;i++){

            if(y<s[i] && offset<=fs){

                pa=fno[x][y]\*fs+offset;

**printf**("\nThe Physical Address is--%d",pa);

                break;

            }else{

**printf**("\nInvalid Process or Page Number or offset");

                break;

            }

        }

    }

}

OUTPUT:

enter the memory size--1000

enter the page size--100

no. of pages available in memory are--10

enter number of processes--2

enter no. of pages required for p[1]--3

enter page table for p[1] (frame numbers <10)---

enter frame number for page number 0:4

enter frame number for page number 1:3

enter frame number for page number 2:5

enter no. of pages required for p[2]—4

enter page table for p[2] (frame numbers <10)---

enter frame number for page number 0:1

enter frame number for page number 1:2

enter frame number for page number 2:3

enter frame number for page number 3:4

enter logical address to find physical address

enter process no. and page number and offset—1 2 23

The Physical Address is—523

7.) A.) **SINGLE LEVEL DIRECTORY.**

CODE:

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

struct Directory{

    char dname[10];

    char fname[10][10];

    int fcnt;

};

void **main**(){

    int i,ch,k,flag=0;

    char f[30];

    struct Directory dir;

    dir.fcnt=0;

**printf**("\nEnter name of directory -- ");

**scanf**("%s",dir.dname);

    while(1){

**printf**("\n\n1. Create File\t 2. Delete File\t 3. Search File\t 4. Display Files\t 5. Exit\n");

**printf**("Enter 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);

                        for(k=i;k<dir.fcnt-1;k++)

**strcpy**(dir.fname[k],dir.fname[k+1]);

                        dir.fcnt--;

                        flag=1;

                    }

                }

                if(flag==0)

**printf**("File %s not found",f);

                break;

            case 3:

                flag=0;

**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);

                        flag=1;

                        break;

                    }

                }

                if(flag==0)

**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);

        }

    }

}

OUTPUT:

Enter name of directory -- Bhuvan

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

Enter your choice: 1

Enter the name of the file – void

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

Enter your choice: 1

Enter the name of the file -- haha

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

Enter your choice: 4

The Files are -- void haha

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

Enter your choice: 2

Enter the name of the file -- void

File void is deleted

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

Enter your choice: 2

Enter the name of the file -- haha

File haha is deleted

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

Enter your choice: 4

Directory Empty

7.) B.) **SINGLE LEVEL DIRECTORY**

CODE:

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

struct Directory{

    char dname[10];

    char fname[10][10];

    int fcnt;

} dir[10];

void **main**(){

    int i,ch,dcnt,k;

    char f[30],d[30];

    dcnt=0;

    while(1){

**printf**("\n\n1. Create Directory\t2. Create File\t3. Delete File");

**printf**("\n4. Search File\t\t5. Display\t6. Exit\n");

**printf**("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);

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);

        }

    }

}

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 -- void

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 -- order

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 -- void

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: 5

Directory Files

void a1

order

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 -- order

Enter name of the file -- b1

File created

1. Create Directory 2. Create File 3. Delete File

4. Search File 5. Display 6. Exit

Enter your choice: 5

Directory Files

void a1

order b1

1. Create Directory 2. Create File 3. Delete File

4. Search File 5. Display 6. Exit

Enter your choice: 4

Enter name of the directory -- void

Enter the name of the file -- a1

File a1 is found

Directory Files

void a1

order b1

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 -- void

Enter name of the file -- a1

File a1 is deleted

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 -- order

Enter name of the file -- b1

File b1 is deleted

1. Create Directory 2. Create File 3. Delete File

4. Search File 5. Display 6. Exit

Enter your choice: 5

Directory Files

void

order