DELHI TECHNICAL CAMPUS
GREATER NOIDA
Affiliated to GGSIPU and approved by AICTE & COA

PRACTICAL FILE

SESSION: 2023-24

Operating Systems Lab (CIC-353)

III Year, V Sem

Submitted to:

Name: Mr. Upendra Pratap Pandey
Designation: Asst. Professor

Submitted by:

Name: Adityan Verma
Enrollment No. 07818002721

Department of Computer Science and Engineering
Delhi Technical Campus, Greater Noida

INDEX

EXPERIMENT NO. 1

Aim: Write a program to implement CPU scheduling for first come first serve (FCFS).

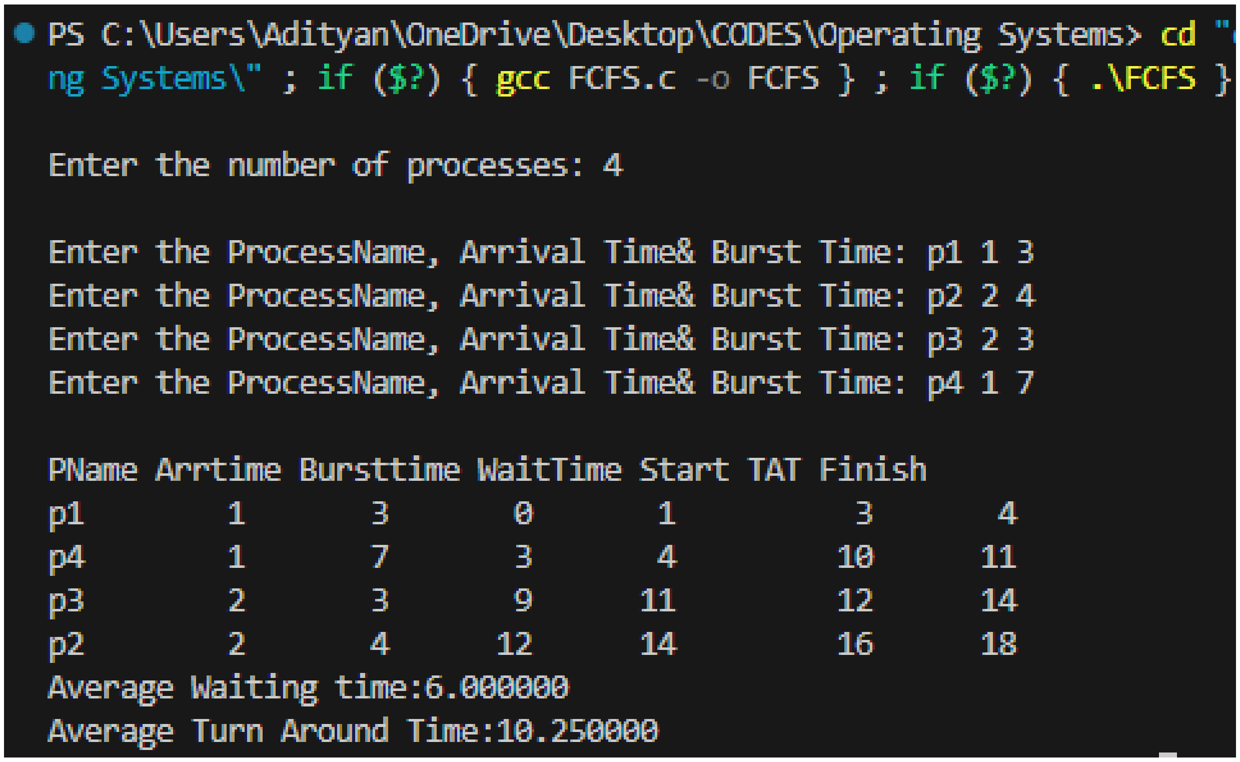
FCFS CPU SCHEDULING ALGORITHM

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.

Code:

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

int main() {
    char pn[10][10],t[10];
    int arr[10],burst[10],star[10],finish[10],tat[10],wt[10],i,j,n,temp;
    int totwt=0,tottat=0;
    printf("\nEnter the number of processes:");
    scanf("%d",&n);
    for(i=0; i<n; i++)
    {
        printf("Enter the ProcessName, Arrival Time& Burst Time:");
        scanf("%s%d%d",&pn[i],&arr[i],&burst[i]);
    }
    for(i=0; i<n; i++)
    {
        for(j=0; j<n; j++)
        {
            if(arr[i]<arr[j])
            {
                temp=arr[i];
                arr[i]=arr[j];
                arr[j]=temp;

                temp= burst[i];
                burst[i]=burst[j];
                burst[j]=temp;

                strcpy(t,pn[i]);
                strcpy(pn[i],pn[j]);
                strcpy(pn[j],t);
            }

        }
    }
    for(i=0; i<n; i++)
    {
        if(i==0)
            star[i]=arr[i];
        else
            star[i]=finish[i-1];
        wt[i]=star[i]-arr[i];
        finish[i]=star[i]+ burst[i];
        tat[i]=finish[i]-arr[i];
    }
    printf("\nPName Arrtime Bursttime WaitTime Start TAT Finish");
    for(i=0; i<n; i++)
    {
   printf("\n%s\t%3d\t%3d\t%3d\t%3d\t%6d\t%6d",pn[i],arr[i],
burst[i],wt[i],star[i],tat[i],finish[i]);
        totwt+=wt[i];
        tottat+=tat[i];
    }
    printf("\nAverage Waiting time:%f",(float)totwt/n);
    printf("\nAverage Turn Around Time:%f",(float)tottat/n);
    return 0;
}
```

OUTPUT:



```
● PS C:\Users\Adityan\OneDrive\Desktop\CODES\Operating Systems> cd "
  ng Systems\" ; if ($?) { gcc FCFS.c -o FCFS } ; if ($?) { .\FCFS }

  Enter the number of processes: 4

  Enter the ProcessName, Arrival Time& Burst Time: p1 1 3
  Enter the ProcessName, Arrival Time& Burst Time: p2 2 4
  Enter the ProcessName, Arrival Time& Burst Time: p3 2 3
  Enter the ProcessName, Arrival Time& Burst Time: p4 1 7

  PName Arrtime Bursttime WaitTime Start TAT Finish
  p1        1       3       0       1           3       4
  p4        1       7       3       4          10      11
  p3        2       3       9      11          12      14
  p2        2       4      12      14          16      18
  Average Waiting time:6.000000
  Average Turn Around Time:10.250000
```

EXPERIMENT NO. 2

Aim: Write a program to implement CPU scheduling for shortest job first (SJF).

SJF CPU SCHEDULING ALGORITHM

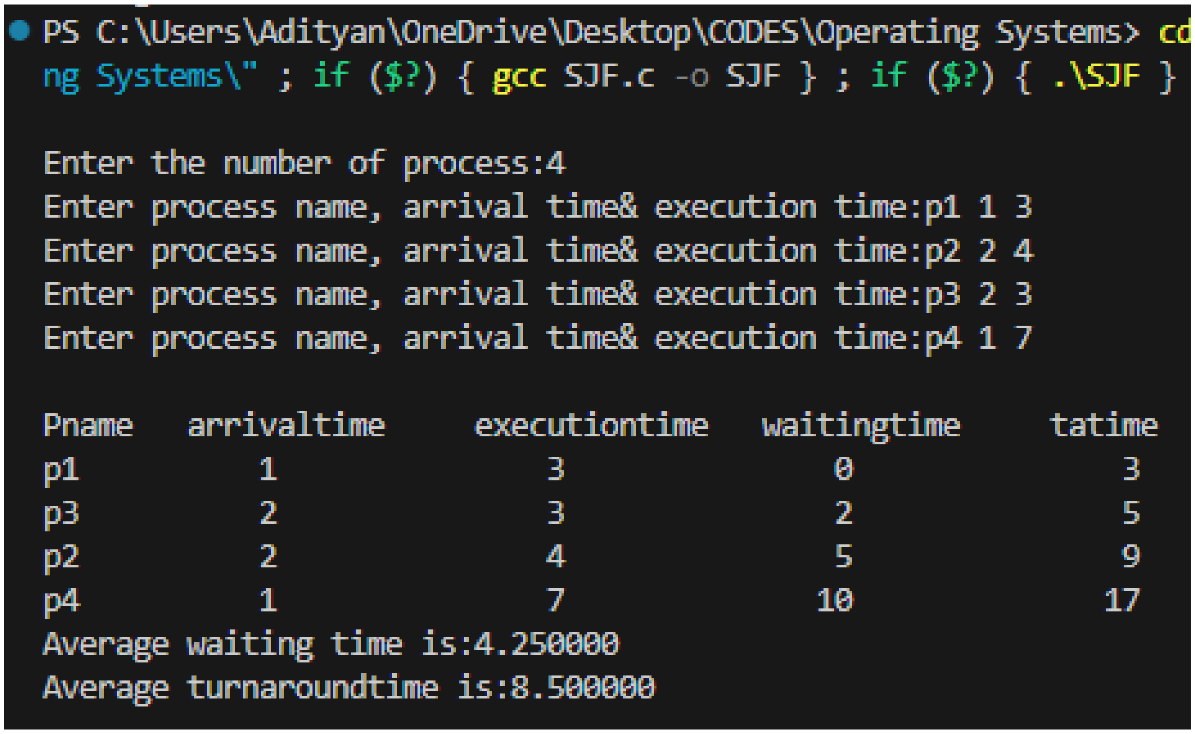
For SJF scheduling algorithm, read the number of processes/jobs in the system, their CPU burst times. Arrange all the jobs in order with respect to their burst times. There may be two jobs in queue with the same execution time, and then FCFS approach is to be performed. Each process will be executed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

Code:

```
#include<stdio.h>
#include<string.h>
int main()
{
   int et[20],at[10],n,i,j,temp,st[10],ft[10],wt[10],ta[10];
   int totwt=0,totta=0;
   float awt,ata;
   char pn[10][10],t[10];
   //clrscr();
   printf("Enter the number of process:");
   scanf("%d",&n);
   for(i=0; i<n; i++)
   {
      printf("Enter process name, arrival time& execution time:");
      //flushall();
      scanf("%s%d%d",pn[i],&at[i],&et[i]);
   }
   for(i=0; i<n; i++)
      for(j=0; j<n; j++)
      {
         if(et[i]<et[j])
         {
            temp=at[i];
            at[i]=at[j];
            at[j]=temp;
            temp=et[i];
            et[i]=et[j];
            et[j]=temp;
```

```
                strcpy(t,pn[i]);
                strcpy(pn[i],pn[j]);
                strcpy(pn[j],t);
            }
        }
    for(i=0; i<n; i++)
    {
        if(i==0)
            st[i]=at[i];
        else
            st[i]=ft[i-1];
        wt[i]=st[i]-at[i];
        ft[i]=st[i]+et[i];
        ta[i]=ft[i]-at[i];
        totwt+=wt[i];
        totta+=ta[i];
    }
    awt=(float)totwt/n;
    ata=(float)totta/n;
    printf("\nPname\tarrivaltime\texecutiontime\twaitingtime\ttatime");
    for(i=0; i<n; i++)
        printf("\n%s\t%5d\t\t%5d\t\t%5d\t\t%5d",pn[i],at[i],et[i],wt[i],ta[i]);
    printf("\nAverage waiting time is:%f",awt);
    printf("\nAverage turnaroundtime is:%f",ata);
    return 0;
}
```

OUTPUT:



```
PS C:\Users\Adityan\OneDrive\Desktop\CODES\Operating Systems> cd
ng Systems\" ; if ($?) { gcc SJF.c -o SJF } ; if ($?) { .\SJF }

Enter the number of process:4
Enter process name, arrival time& execution time:p1 1 3
Enter process name, arrival time& execution time:p2 2 4
Enter process name, arrival time& execution time:p3 2 3
Enter process name, arrival time& execution time:p4 1 7

Pname   arrivaltime     executiontime   waitingtime     tatime
p1          1               3               0               3
p3          2               3               2               5
p2          2               4               5               9
p4          1               7              10              17
Average waiting time is:4.250000
Average turnaroundtime is:8.500000
```

EXPERIMENT NO. 3

AIM: Write a program to perform priority scheduling.

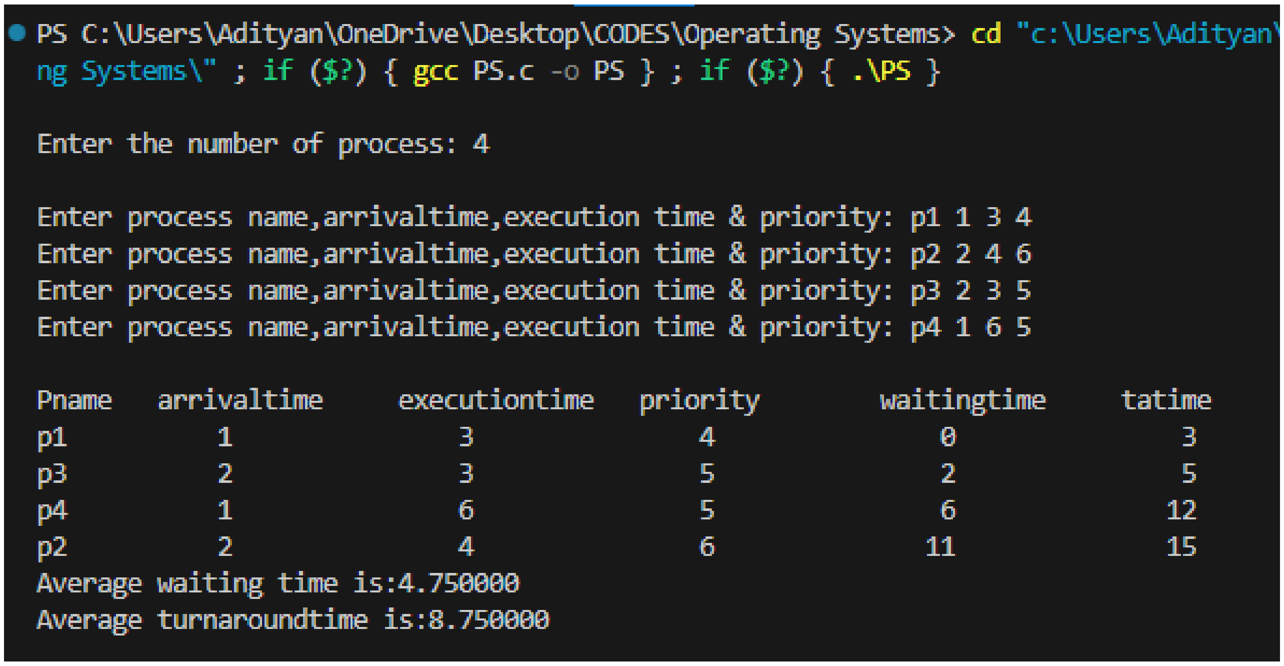
PRIORITY CPU SCHEDULING ALGORITHM

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.

Code:

```
#include<stdio.h>
#include<string.h>
int main()
{
    int et[20],at[10],n,i,j,temp,p[10],st[10],ft[10],wt[10],ta[10];
    int totwt=0,totta=0;
    float awt,ata;
    char pn[10][10],t[10];
    //clrscr();
    printf("Enter the number of process:");
    scanf("%d",&n);
    for(i=0; i<n; i++)
    {
        printf("Enter process name,arrivaltime,execution time & priority:");
        //flushall();
        scanf("%s%d%d%d",pn[i],&at[i],&et[i],&p[i]);
    }
    for(i=0; i<n; i++)
        for(j=0; j<n; j++)
        {
            if(p[i]<p[j])
            {
                temp=p[i];
                p[i]=p[j];
                p[j]=temp;
                temp=at[i];
                at[i]=at[j];
                at[j]=temp;
                temp=et[i];
                et[i]=et[j];
                et[j]=temp;
                strcpy(t,pn[i]);
                strcpy(pn[i],pn[j]);
                strcpy(pn[j],t);
            }
        }
    for(i=0; i<n; i++)

    {

        if(i==0)
        {
            st[i]=at[i];
            wt[i]=st[i]-at[i];
            ft[i]=st[i]+et[i];
            ta[i]=ft[i]-at[i];
        }
        else
        {
            st[i]=ft[i-1];
            wt[i]=st[i]-at[i];
            ft[i]=st[i]+et[i];
            ta[i]=ft[i]-at[i];
        }
        totwt+=wt[i];
        totta+=ta[i];
    }
    awt=(float)totwt/n;
    ata=(float)totta/n;
    printf("\nPname\tarrivaltime\texecutiontime\tpriority\twaitingtime\ttatime");
    for(i=0; i<n; i++)
        printf("\n%s\t%5d\t\t%5d\t\t%5d\t\t%5d\t\t%5d",pn[i],at[i],et[i],p[i],wt[i],ta[i]);
    printf("\nAverage waiting time is:%f",awt);
    printf("\nAverage turnaroundtime is:%f",ata);
    return 0;
}
```

OUTPUT:



```
PS C:\Users\Adityan\OneDrive\Desktop\CODES\Operating Systems> cd "c:\Users\Adityan\
ng Systems\" ; if ($?) { gcc PS.c -o PS } ; if ($?) { .\PS }

Enter the number of process: 4

Enter process name,arrivaltime,execution time & priority: p1 1 3 4
Enter process name,arrivaltime,execution time & priority: p2 2 4 6
Enter process name,arrivaltime,execution time & priority: p3 2 3 5
Enter process name,arrivaltime,execution time & priority: p4 1 6 5

Pname   arrivaltime     executiontime   priority        waitingtime     tatime
p1          1               3               4               0               3
p3          2               3               5               2               5
p4          1               6               5               6              12
p2          2               4               6              11              15
Average waiting time is:4.750000
Average turnaroundtime is:8.750000
```

EXPERIMENT NO. 4

Aim: Write a program to implement CPU scheduling for Round Robin.

ROUND ROBIN CPU SCHEDULING ALGORITHM

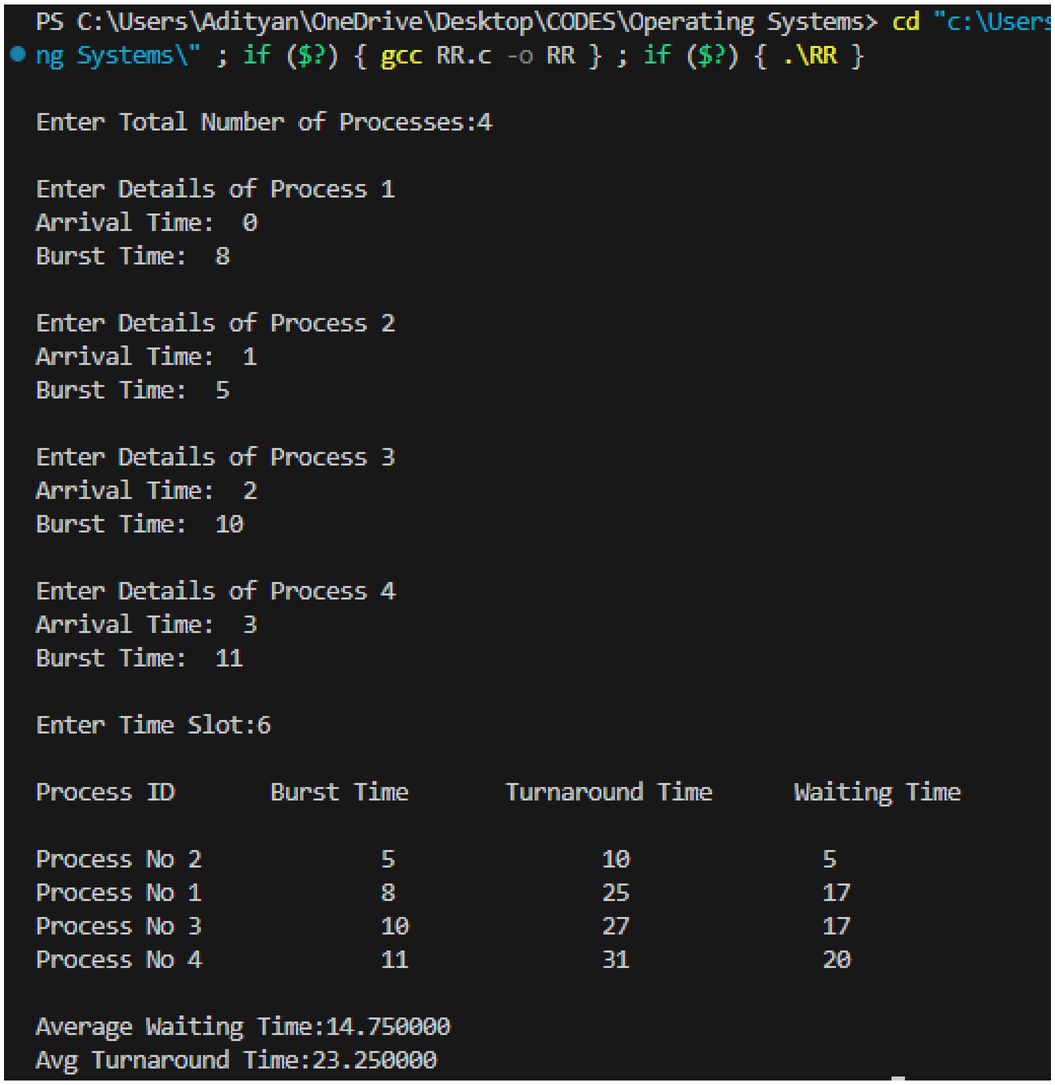
For round robin scheduling algorithm, read the number of processes/jobs in the system, their CPU burst times, and the size of the time slice. Time slices are assigned to each process in equal portions and in circular order, handling all processes execution. This allows every process to get an equal chance.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

Code:

```
#include <stdio.h>
int main()
{
   int n;
   printf("\nEnter Total Number of Processes:");
   scanf("%d", &n);
   int wait_time = 0, ta_time = 0, arr_time[n], burst_time[n],
      temp_burst_time[n];
   int x = n;
   for (int i = 0; i < n; i++)
   {
      printf("\nEnter Details of Process %d \n", i + 1);
      printf("Arrival Time:  ");
      scanf("%d", &arr_time[i]);
      printf("Burst Time:  ");
      scanf("%d", &burst_time[i]);
      temp_burst_time[i] = burst_time[i];
   }
   int time_slot;
   printf("\nEnter Time Slot:");
   scanf("%d", &time_slot);

   int total = 0, counter = 0, i;
   printf("\nProcess ID      Burst Time      Turnaround Time      Waiting Time\n");
   for (total = 0, i = 0; x != 0;)
   {
      if (temp_burst_time[i] <= time_slot && temp_burst_time[i] > 0)
      {
         total = total + temp_burst_time[i];
         temp_burst_time[i] = 0;
```

```
            counter = 1;
        }
        else if (temp_burst_time[i] > 0)
        {
            temp_burst_time[i] = temp_burst_time[i] - time_slot;
            total += time_slot;
        }
        if (temp_burst_time[i] == 0 && counter == 1)
        {
            x--;
            printf("\nProcess No %d  \t\t %d\t\t %d\t\t %d", i + 1, burst_time[i],
                   total - arr_time[i], total - arr_time[i] - burst_time[i]);
            wait_time = wait_time + total - arr_time[i] - burst_time[i];
            ta_time += total - arr_time[i];
            counter = 0;
        }
        if (i == n - 1)
        {
            i = 0;
        }
        else if (arr_time[i + 1] <= total)
        {
            i++;
        }
        else
        {
            i = 0;
        }
    }
    float average_wait_time = wait_time * 1.0 / n;
    float average_turnaround_time = ta_time * 1.0 / n;
    printf("\n\nAverage Waiting Time:%f", average_wait_time);
    printf("\nAvg Turnaround Time:%f", average_turnaround_time);
    return 0;
}
```

OUTPUT:



```
PS C:\Users\Adityan\OneDrive\Desktop\CODES\Operating Systems> cd "c:\Users
ng Systems\" ; if ($?) { gcc RR.c -o RR } ; if ($?) { .\RR }

Enter Total Number of Processes:4

Enter Details of Process 1
Arrival Time:  0
Burst Time:  8

Enter Details of Process 2
Arrival Time:  1
Burst Time:  5

Enter Details of Process 3
Arrival Time:  2
Burst Time:  10

Enter Details of Process 4
Arrival Time:  3
Burst Time:  11

Enter Time Slot:6

Process ID        Burst Time        Turnaround Time      Waiting Time

Process No 2              5                10               5
Process No 1              8                25               17
Process No 3              10               27               17
Process No 4              11               31               20

Average Waiting Time:14.750000
Avg Turnaround Time:23.250000
```

EXPERIMENT NO. 5

Aim: Write a program for page replacement policy by using a) LRU b) FIFO c) optimal.

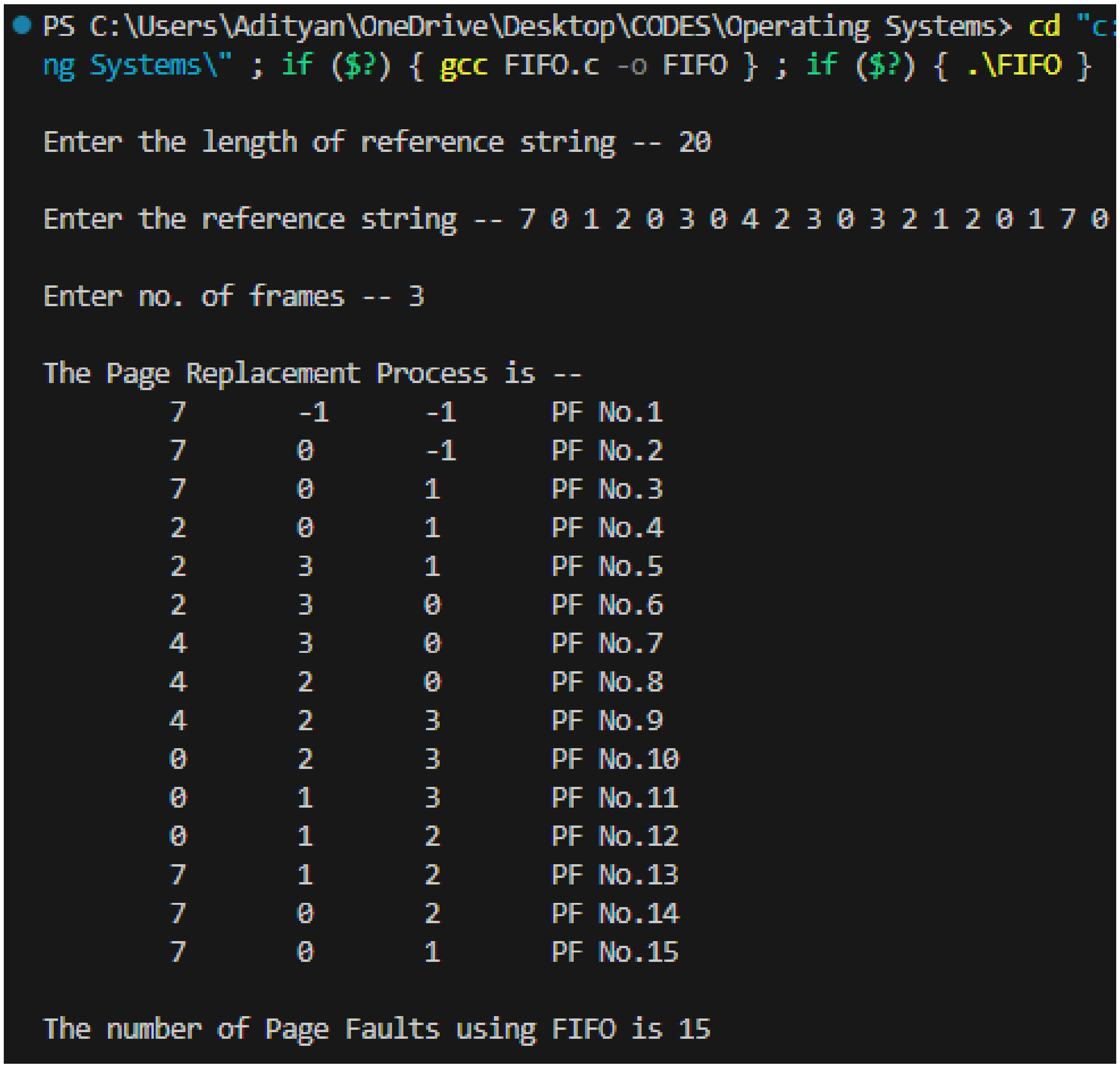
Page replacement is basic to demand paging. It completes the separation between logical memory and physical memory. With this mechanism, an enormous virtual memory can be provided for programmers on a smaller physical memory. There are many different page-replacement algorithms. Every operating system 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.

FIFO PAGE REPLACEMENT ALGORITHM

Code:

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

    printf("\nEnter the length of reference string -- ");
    scanf("%d", &n);

    printf("\nEnter the reference string -- ");
    for (i = 0; i < n; i++) {
        scanf("%d", &rs[i]);
    }

    printf("\nEnter no. of frames -- ");
    scanf("%d", &f);
    for (i = 0; i < f; i++) {
        m[i] = -1;
    }

    printf("\nThe Page Replacement Process is --\n");
    for (i = 0; i < n; i++)
    {
        flag = 0;
        for (j = 0; j < f; j++)
        {
            if (m[j] == rs[i]) {
                flag = 1; // Page found in memory
                break;
            }
        }

        if (flag == 0) // Page fault
        {
            pf++;
            m[count] = rs[i];
            count = (count + 1) % f;

            for (j = 0; j < f; j++)
                printf("\t%d", m[j]);

            printf("\tPF No.%d\n", pf);
        }
    }

    printf("\nThe number of Page Faults using FIFO is %d\n\n", pf);
    return 0;
}
```

OUTPUT:



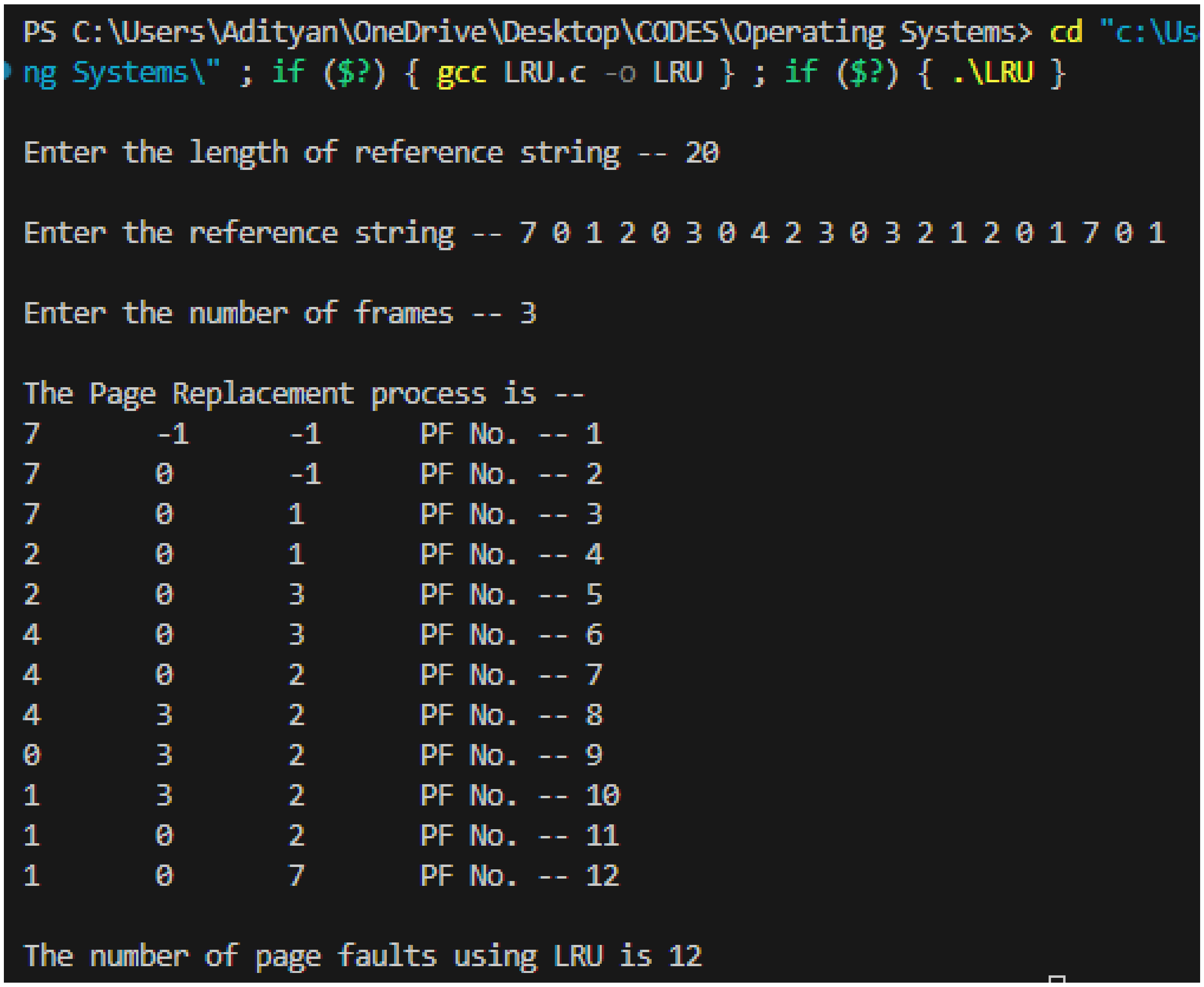
```
● PS C:\Users\Adityan\OneDrive\Desktop\CODES\Operating Systems> cd "c:
  ng Systems\" ; if ($?) { gcc FIFO.c -o FIFO } ; if ($?) { .\FIFO }

  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

  Enter no. of frames -- 3

  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       3       1       PF No.5
          2       3       0       PF No.6
          4       3       0       PF No.7
          4       2       0       PF No.8
          4       2       3       PF No.9
          0       2       3       PF No.10
          0       1       3       PF No.11
          0       1       2       PF No.12
          7       1       2       PF No.13
          7       0       2       PF No.14
          7       0       1       PF No.15

  The number of Page Faults using FIFO is 15
```

LRU PAGE REPLACEMENT ALGORITHM

Code:

```
#include <stdio.h>

int main() {
    int i, j, min, rs[25], m[10], count[10], flag[25], n, f, pf = 0, next = 1;

    printf("\nEnter the length of reference string -- ");
    scanf("%d", &n);

    printf("\nEnter the reference string -- ");
    for (i = 0; i < n; i++) {
        scanf("%d", &rs[i]);
        flag[i] = 0;
    }

    printf("\nEnter 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++) {
        flag[i] = 0;

        // Check if page is already in memory
        for (j = 0; j < f; j++) {
            if (m[j] == rs[i]) {
                flag[i] = 1;
                count[j] = next;
                next++;
                break;
            }
        }

        if (flag[i] == 0) {
            int replace = 0;
            min = count[0];

            // Finding the page with minimum count to replace
            for (j = 1; j < f; j++) {
                if (count[j] < min) {
                    min = count[j];
                    replace = j;
                }
            }

            m[replace] = rs[i];
            count[replace] = next;
            next++;
            pf++;

            for (j = 0; j < f; j++)
                printf("%d\t", m[j]);

            printf("PF No. -- %d\n", pf);
        }
    }

    printf("\nThe number of page faults using LRU is %d\n", pf);
    return 0;
}
```

OUTPUT:



```
PS C:\Users\Adityan\OneDrive\Desktop\CODES\Operating Systems> cd "c:\Us
ng Systems\" ; if ($?) { gcc LRU.c -o LRU } ; if ($?) { .\LRU }

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

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

The number of page faults using LRU is 12
```

OPTIMAL PAGE REPLACEENT ALGORITHM

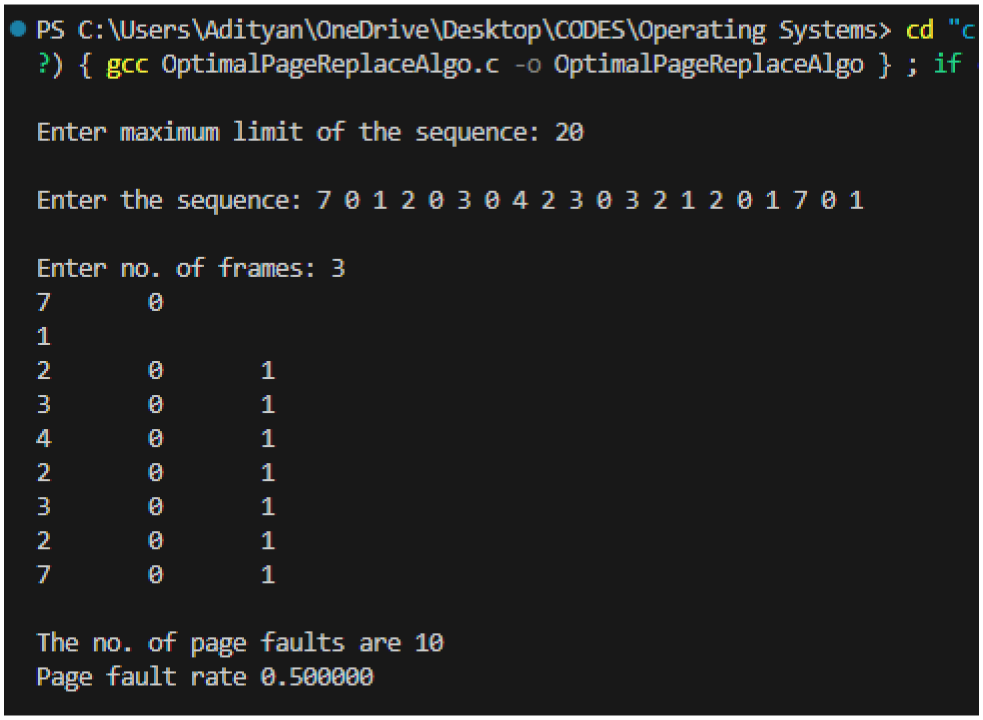
Optimal page replacement algorithm has the lowest page-fault rate of all algorithms and will never suffer from Belady's anomaly. The basic idea is to replace the page that will not be used for the longest period of time. Use of this page-replacement algorithm guarantees the lowest possible page fault rate for a fixed number of frames. Unfortunately, the optimal page-replacement algorithm is difficult to implement, because it requires future knowledge of the reference string.

Code:

```
#include <stdio.h>

int findmax(int a[], int n) {
    int max, i, k = 0;
    max = a[0];
    for (i = 0; i < n; i++) {
        if (max < a[i]) {
            max = a[i];
            k = i;
        }
    }
    return k;
}

int main() {
    int seq[30], fr[5], pos[5];
    int count = 0, pf = 0, max, n, i, j, m, k, t, s, flag, p = 0;
    float pfr;

    printf("\nEnter maximum limit of the sequence: ");
    scanf("%d", &max);

    printf("\nEnter the sequence: ");
    for (i = 0; i < max; i++)
        scanf("%d", &seq[i]);

    printf("\nEnter no. of frames: ");
    scanf("%d", &n);

    fr[0] = seq[0];
    pf++;
```

```
printf("%d\t", fr[0]);
count = 1;
i = 1;

while (count < n) {
    flag = 1;
    p++;
    for (j = 0; j < i; j++) {
        if (seq[i] == seq[j])
            flag = 0;
    }
    if (flag != 0) {
        fr[count] = seq[i];
        printf("%d\t", fr[count]);
        count++;
        pf++;
        printf("\n");
    }
    i++;
}

for (i = p; i < max; i++) {
    flag = 1;
    for (j = 0; j < n; j++) {
        if (seq[i] == fr[j])
            flag = 0;
    }
    if (flag != 0) {
        for (j = 0; j < n; j++) {
            m = fr[j];
            for (k = i; k < max; k++) {
                if (seq[k] == m)
                    pos[j] = k;
                else
                    pos[j] = 0;
            }
        }
        for (k = 0; k < n; k++) {
            if (pos[k] == 0)
                flag = 0;
        }
        if (flag != 0)
            s = findmax(pos, n);
        if (flag == 0) {
            for (k = 0; k < n; k++) {
                if (pos[k] == 0) {
                        s = k;
                        break;
                    }
                }
            }
            fr[s] = seq[i];
            for (k = 0; k < n; k++)
                printf("%d\t", fr[k]);
            pf++;
            printf("\n");
        }
    }
    pfr = (float)pf / (float)max;
    printf("\nThe no. of page faults are %d", pf);
    printf("\nPage fault rate %f\n", pfr);

    printf("\n\n");
    return 0;
}
```

OUTPUT:



```
PS C:\Users\Adityan\OneDrive\Desktop\CODES\Operating Systems> cd "c
?) { gcc OptimalPageReplaceAlgo.c -o OptimalPageReplaceAlgo } ; if

Enter maximum limit of the sequence: 20

Enter the sequence: 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
7       0
1
2       0       1
3       0       1
4       0       1
2       0       1
3       0       1
2       0       1
7       0       1

The no. of page faults are 10
Page fault rate 0.500000
```

EXPERIMENT NO. 6

Aim: Write a program to implement first fit, best fit, worst fit algorithm for memory management.

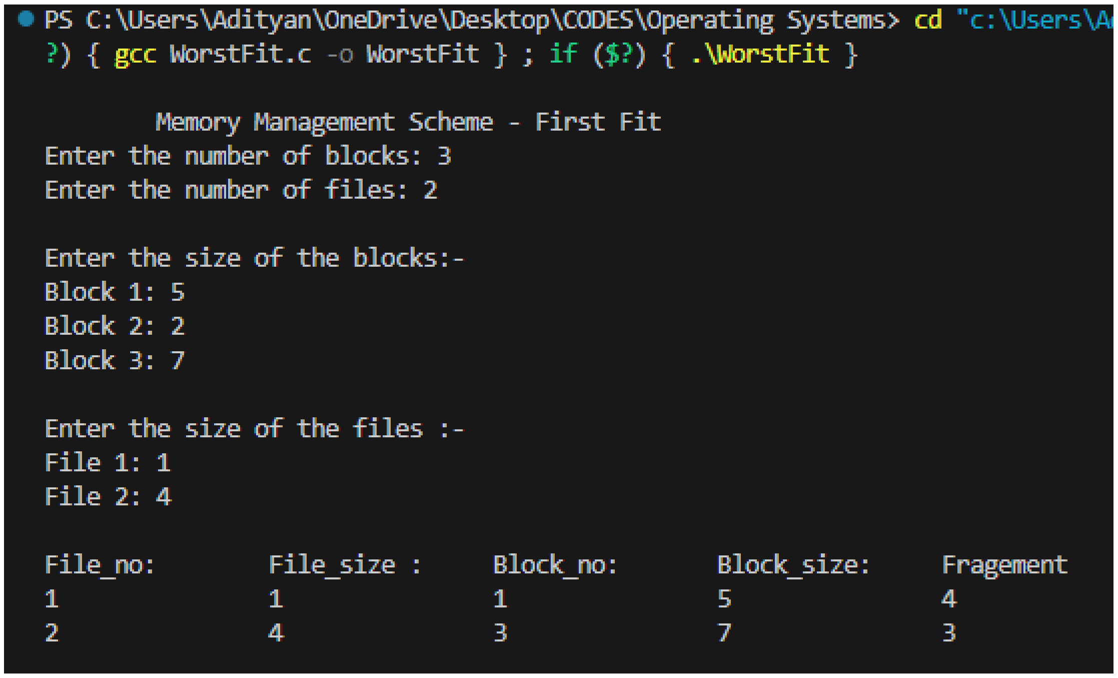
One of the simplest methods for memory allocation is to divide memory into several fixed-sized partitions. Each partition may contain exactly one process. In this multiple-partition method, when a partition is free, a process is selected from the input queue and is loaded into the free partition. When the process terminates, the partition becomes available for another process. The operating system keeps a table indicating which parts of memory are available and which are occupied. Finally, when a process arrives and needs memory, a memory section large enough for this process is provided. When it is time to load or swap a process into main memory, and if there is more than one free block of memory of sufficient size, then the operating system must decide which free block to allocate. Best-fit strategy chooses the block that is closest in size to the request. First-fit chooses the first available block that is large enough. Worst-fit chooses the largest available block.

WORST-FIT

Code:

```
#include <stdio.h>
#define max 25

void main()
{
    int frag[max], b[max], f[max], i, j, nb, nf, temp;
    static int bf[max], ff[max];

    printf("\n\tMemory Management Scheme - First Fit");
    printf("\nEnter the number of blocks: ");
    scanf("%d", &nb);
    printf("Enter the number of files: ");
    scanf("%d", &nf);
    printf("\nEnter the size of the blocks:-\n");
    for (i = 1; i <= nb; i++)
    {
        printf("Block %d: ", i);
        scanf("%d", &b[i]);
    }
    printf("\nEnter the size of the files :-\n");
    for (i = 1; i <= nf; i++)
    {
        printf("File %d: ", i);
        scanf("%d", &f[i]);
    }
    for (i = 1; i <= nf; i++)
    {
        for (j = 1; j <= nb; j++)
        {
            if (bf[j] != 1)
            {
                temp = b[j] - f[i];
                if (temp >= 0)
                {
                    ff[i] = j;
                    break;
                }
            }
        }
        frag[i] = temp;
        bf[ff[i]] = 1;
    }
    printf("\nFile_no:\tFile_size :\tBlock_no:\tBlock_size:\tFragement");
    for (i = 1; i <= nf; i++)
        printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d", i, f[i], ff[i], b[ff[i]], frag[i]);

    printf("\n\n");
}
```

OUTPUT:



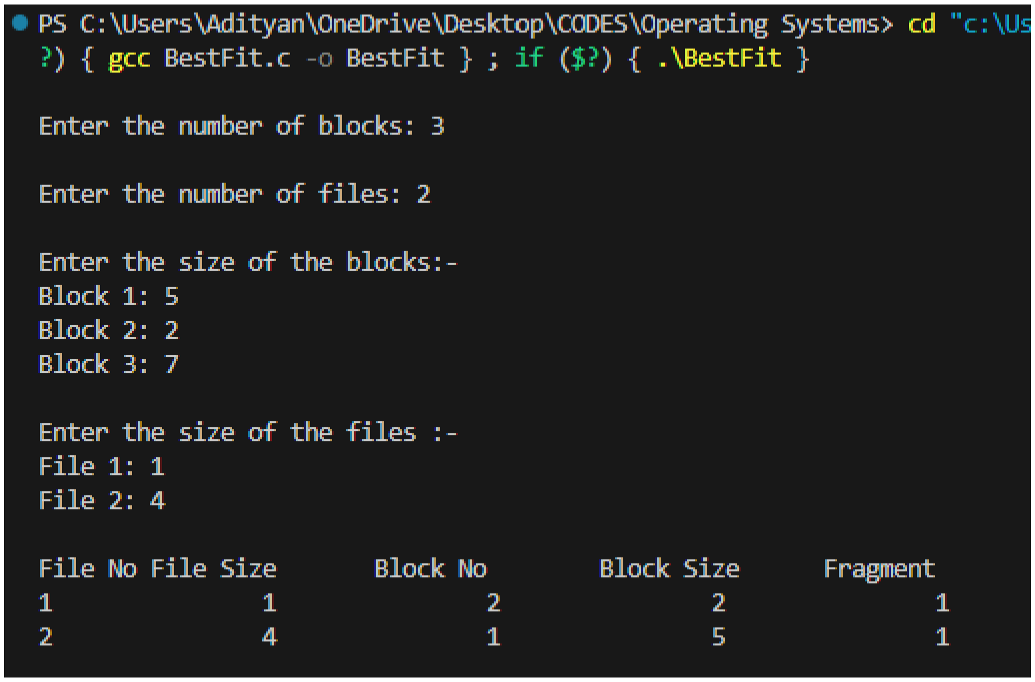
```
● PS C:\Users\Adityan\OneDrive\Desktop\CODES\Operating Systems> cd "c:\Users\A
  ?) { gcc WorstFit.c -o WorstFit } ; if ($?) { .\WorstFit }

          Memory Management Scheme - First Fit
  Enter the number of blocks: 3
  Enter the number of files: 2

  Enter the size of the blocks:-
  Block 1: 5
  Block 2: 2
  Block 3: 7

  Enter the size of the files :-
  File 1: 1
  File 2: 4

  File_no:        File_size :     Block_no:       Block_size:     Fragement
  1               1               1               5               4
  2               4               3               7               3
```

BEST-FIT

Code:

```
#include <stdio.h>

#define max 25

void main()
{
    int frag[max], b[max], f[max], i, j, nb, nf, temp, lowest = 10000;
    static int bf[max], ff[max];
    printf("\nEnter the number of blocks: ");
    scanf("%d", &nb);
    printf("\nEnter the number of files: ");
    scanf("%d", &nf);
    printf("\nEnter the size of the blocks:-\n");
    for (i = 1; i <= nb; i++) {
        printf("Block %d: ", i);
        scanf("%d", &b[i]);
    }
    printf("\nEnter the size of the files :-\n");
    for (i = 1; i <= nf; i++) {
        printf("File %d: ", i);
        scanf("%d", &f[i]);
    }
    for (i = 1; i <= nf; i++)
    {
        for (j = 1; j <= nb; j++)
        {
            if (bf[j] != 1)
            {
                temp = b[j] - f[i];
                if (temp >= 0)
                    if (lowest > temp)
                    {
                        ff[i] = j;
                        lowest = temp;
                    }
            }
        }
        frag[i] = lowest;
        bf[ff[i]] = 1;
        lowest = 10000;
    }
    printf("\nFile No\tFile Size \tBlock No\tBlock Size\tFragment");
    for (i = 1; i <= nf && ff[i] != 0; i++)
        printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d", i, f[i], ff[i], b[ff[i]], frag[i]);

    printf("\n\n");
}
```

OUTPUT:



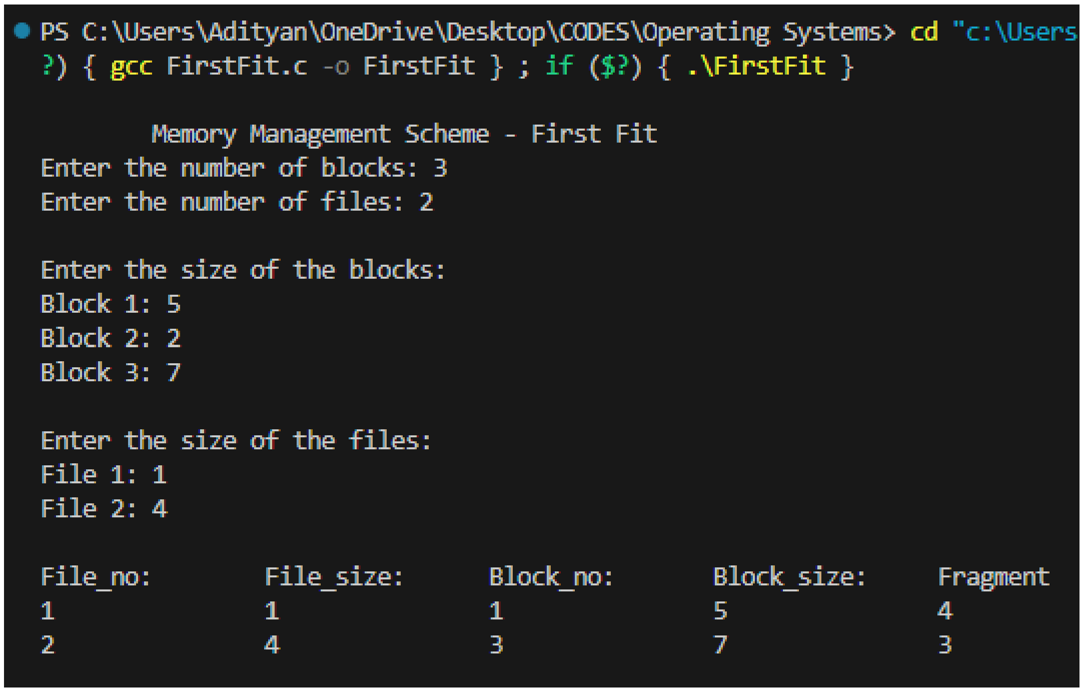
```
PS C:\Users\Adityan\OneDrive\Desktop\CODES\Operating Systems> cd "c:\Us
?) { gcc BestFit.c -o BestFit } ; if ($?) { .\BestFit }

Enter the number of blocks: 3

Enter the number of files: 2

Enter the size of the blocks:-
Block 1: 5
Block 2: 2
Block 3: 7

Enter the size of the files :-
File 1: 1
File 2: 4

File No File Size       Block No        Block Size      Fragment
1               1               2               2               1
2               4               1               5               1
```

FIRST-FIT

Code:

```
#include <stdio.h>
#define MAX 25
int main() {
    int frag[MAX], b[MAX], f[MAX], bf[MAX], ff[MAX];
    int i, j, nb, nf, temp;

    printf("\n\tMemory Management Scheme - First Fit\n");
    printf("Enter the number of blocks: ");
    scanf("%d", &nb);

    printf("Enter the number of files: ");
    scanf("%d", &nf);

    printf("\nEnter the size of the blocks:\n");
    for (i = 1; i <= nb; i++) {
        printf("Block %d: ", i);
        scanf("%d", &b[i]);
        bf[i] = 0; // Initialize allocation flag for blocks
    }

    printf("\nEnter the size of the files:\n");
    for (i = 1; i <= nf; i++) {
        printf("File %d: ", i);
        scanf("%d", &f[i]);
        ff[i] = -1; // Initialize file allocation block to -1 (unallocated)
    }

    for (i = 1; i <= nf; i++) {
        for (j = 1; j <= nb; j++) {
            if (bf[j] == 0 && b[j] >= f[i]) { // if block j is not allocated and fits file i
                ff[i] = j;
                frag[i] = b[j] - f[i]; // Calculate fragmentation
                bf[j] = 1; // Mark block as allocated
                break; // Move to next file
            }
        }
    }

    printf("\nFile_no:\tFile_size:\tBlock_no:\tBlock_size:\tFragment\n");
    for (i = 1; i <= nf; i++) {
        printf("%d\t\t%d\t\t", i, f[i]);
        if (ff[i] != -1) {
            printf("%d\t\t%d\t\t%d\n", ff[i], b[ff[i]], frag[i]);
        } else {
            printf("Not allocated\tNot allocated\tNot allocated\n");
        }
    }
    printf("\n");
    return 0;
}
```

OUTPUT:



```
● PS C:\Users\Adityan\OneDrive\Desktop\CODES\Operating Systems> cd "c:\Users
  ?) { gcc FirstFit.c -o FirstFit } ; if ($?) { .\FirstFit }

          Memory Management Scheme - First Fit
  Enter the number of blocks: 3
  Enter the number of files: 2

  Enter the size of the blocks:
  Block 1: 5
  Block 2: 2
  Block 3: 7

  Enter the size of the files:
  File 1: 1
  File 2: 4

  File_no:        File_size:      Block_no:       Block_size:     Fragment
  1               1               1               5               4
  2               4               3               7               3
```

EXPERIMENT NO. 7

Aim: Write a program to implement reader writer problem using semaphore.

Code:

```
#include <semaphore.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
sem_t x, y;
pthread_t writerthreads[100], readerthreads[100];
int readercount = 0;
void *reader(void *param) {
    sem_wait(&x);
    readercount++;
    if (readercount == 1)
        sem_wait(&y);
    sem_post(&x);
    printf("%lu Reader is inside\n", pthread_self());
    usleep(100000); // Simulating reader reading time
    sem_wait(&x);
    readercount--;
    if (readercount == 0) {
        sem_post(&y);
    }
    sem_post(&x);
    printf("%lu Reader is leaving\n", pthread_self());
    return NULL;
}
void *writer(void *param) {
    sem_wait(&y);
    printf("%lu Writer is inside\n", pthread_self());
    usleep(100000); // Simulating writer writing time
    sem_post(&y);
    printf("%lu Writer is leaving\n", pthread_self());
    return NULL;
}
int main() {
    int n2, i;
    printf("Enter the number of readers and writers: ");
    scanf("%d", &n2);
    printf("\n");
    sem_init(&x, 0, 1);
    sem_init(&y, 0, 1);
    for (i = 0; i < n2; i++) {
        pthread_create(&readerthreads[i], NULL, reader, NULL);
        pthread_create(&writerthreads[i], NULL, writer, NULL);
    }
    for (i = 0; i < n2; i++) {
        pthread_join(readerthreads[i], NULL);
        pthread_join(writerthreads[i], NULL);
    }
    return 0;
}
```

OUTPUT:

```
1 reader is inside

Writer is trying to enter

2 reader is inside

2 Reader is leaving

2 reader is inside

Writer is trying to enter

2 Reader is leaving

1 Reader is leaving
```

EXPERIMENT NO. 8

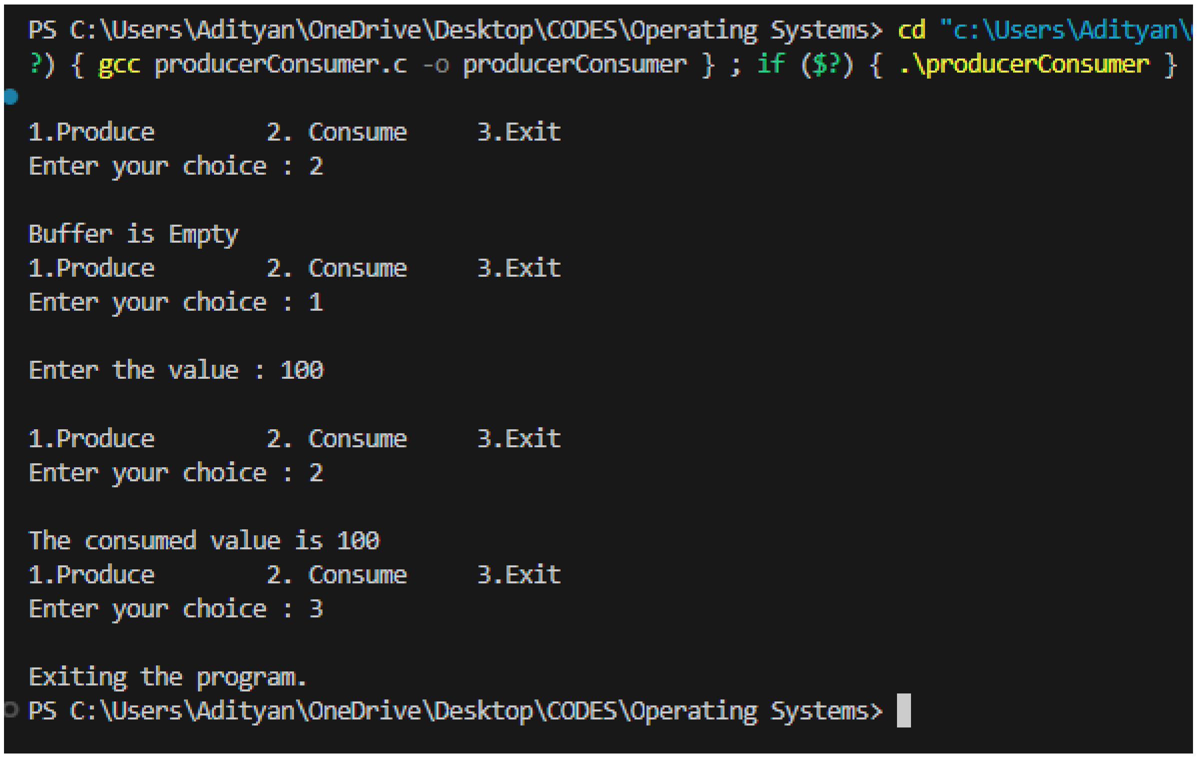
Aim: Write a program to implement Producer-consumer problem using semaphores.

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.

Code:

```
#include <stdio.h>

int main() {
    int buffer[10], bufsize = 10, in = 0, out = 0, produce, consume, choice = 0;

    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;

            case 3:
                printf("\nExiting the program.\n");
                break;

            default:
                printf("\nInvalid choice. Enter 1, 2, or 3.\n");
                break;
        }
    }

    return 0;
}
```

OUTPUT:



```
PS C:\Users\Adityan\OneDrive\Desktop\CODES\Operating Systems> cd "c:\Users\Adityan\
?) { gcc producerConsumer.c -o producerConsumer } ; if ($?) { .\producerConsumer }

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

Exiting the program.
PS C:\Users\Adityan\OneDrive\Desktop\CODES\Operating Systems>
```

EXPERIMENT NO. 9

Aim: Write a program to implement Banker’s algorithm for deadlock avoidance.

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.

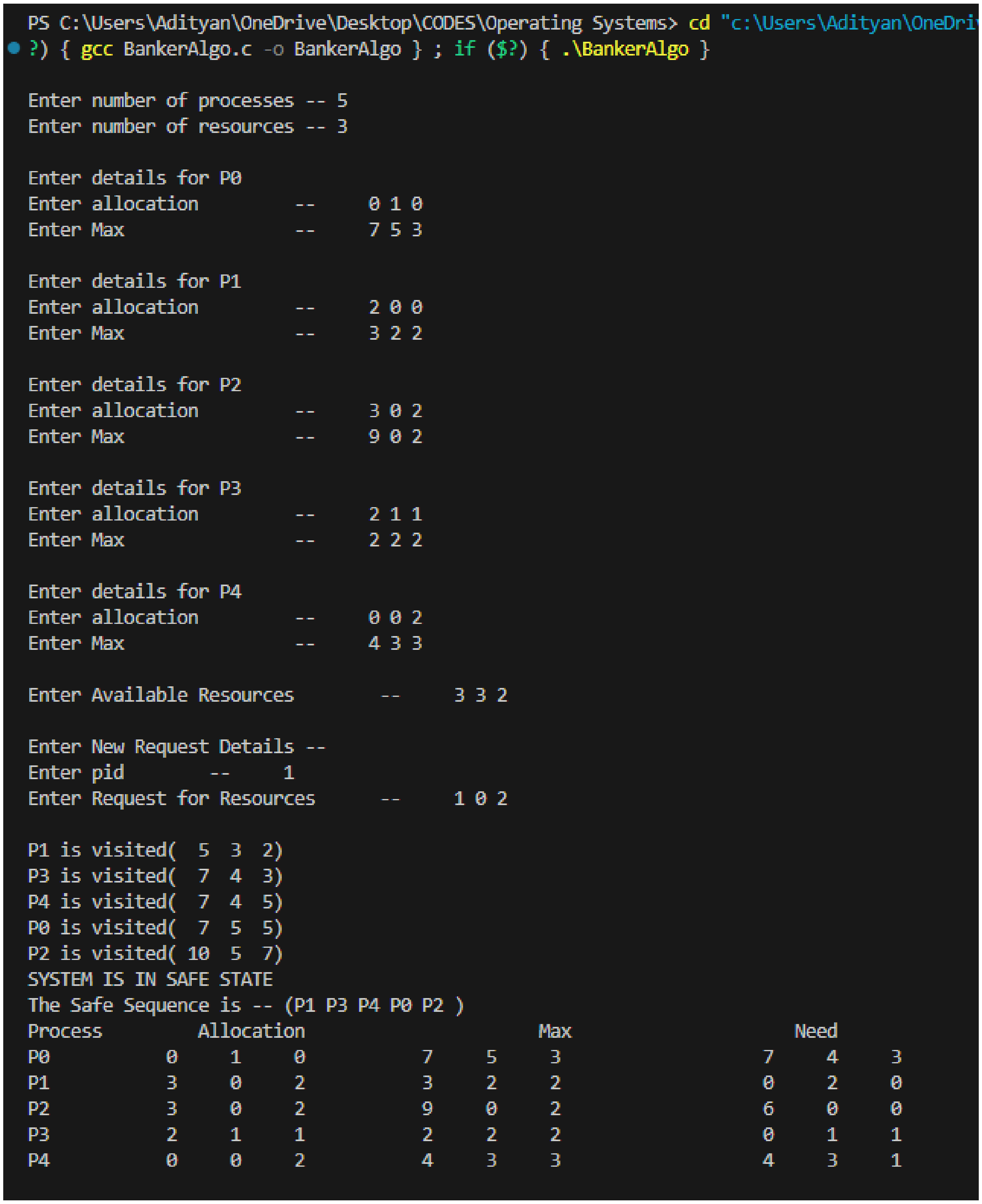
Banker s algorithm is a deadlock avoidance algor each resource type.

Code:

```
#include <stdio.h>

struct file {
    int all[10];
    int max[10];
    int need[10];
    int flag;
};

int main() {
    struct file f[10];
    int i, j, p, b, n, r, g, cnt = 0, id, newr;
    int avail[10], seq[10];

    printf("\nEnter number of processes -- ");
    scanf("%d", &n);
    printf("Enter number of resources -- ");
```

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

for (i = 0; i < n; i++) {
    printf("\nEnter details for 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("Enter Request for Resources \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;
int 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;
            }
            if (b == r) {
                printf("\nP%d is visited", j);
                seq[fl++] = j;
                f[j].flag = 1;
                for (int k = 0; k < r; k++)
                    avail[k] = avail[k] + f[j].all[k];
                cnt = cnt + 1;
                printf("(");
                for (int k = 0; k < r; k++)
                    printf("%3d", avail[k]);
                printf(")");
                g = 1;
            }
        }
    }
    if (g == 0) {
        printf("\nREQUEST NOT GRANTED -- DEADLOCK OCCURRED");
        printf("\nSYSTEM IS IN UNSAFE STATE");
        return 0;
    }
}

printf("\nSYSTEM IS IN SAFE STATE");
printf("\nThe Safe Sequence is -- (");
for (i = 0; i < fl; i++)
    printf("P%d ", seq[i]);
printf(")\n");

printf("Process\t\tAllocation\t\t\tMax\t\t\tNeed\n");
for (i = 0; i < n; i++) {
    printf("P%d\t", i);
    for (j = 0; j < r; j++)
        printf("%6d", f[i].all[j]);

    printf("\t");

    for (j = 0; j < r; j++)
        printf("%6d", f[i].max[j]);

    printf("\t\t");

    for (j = 0; j < r; j++)
        printf("%6d", f[i].need[j]);
        printf("\n");
    }

    printf("\n\n");
    return 0;
}
```

OUTPUT:



```
PS C:\Users\Adityan\OneDrive\Desktop\CODES\Operating Systems> cd "c:\Users\Adityan\OneDri
?) { gcc BankerAlgo.c -o BankerAlgo } ; if ($?) { .\BankerAlgo }

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

P1 is visited(  5  3  2)
P3 is visited(  7  4  3)
P4 is visited(  7  4  5)
P0 is visited(  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 NO. 10

Aim: Write a program to implement the various file organization techniques.

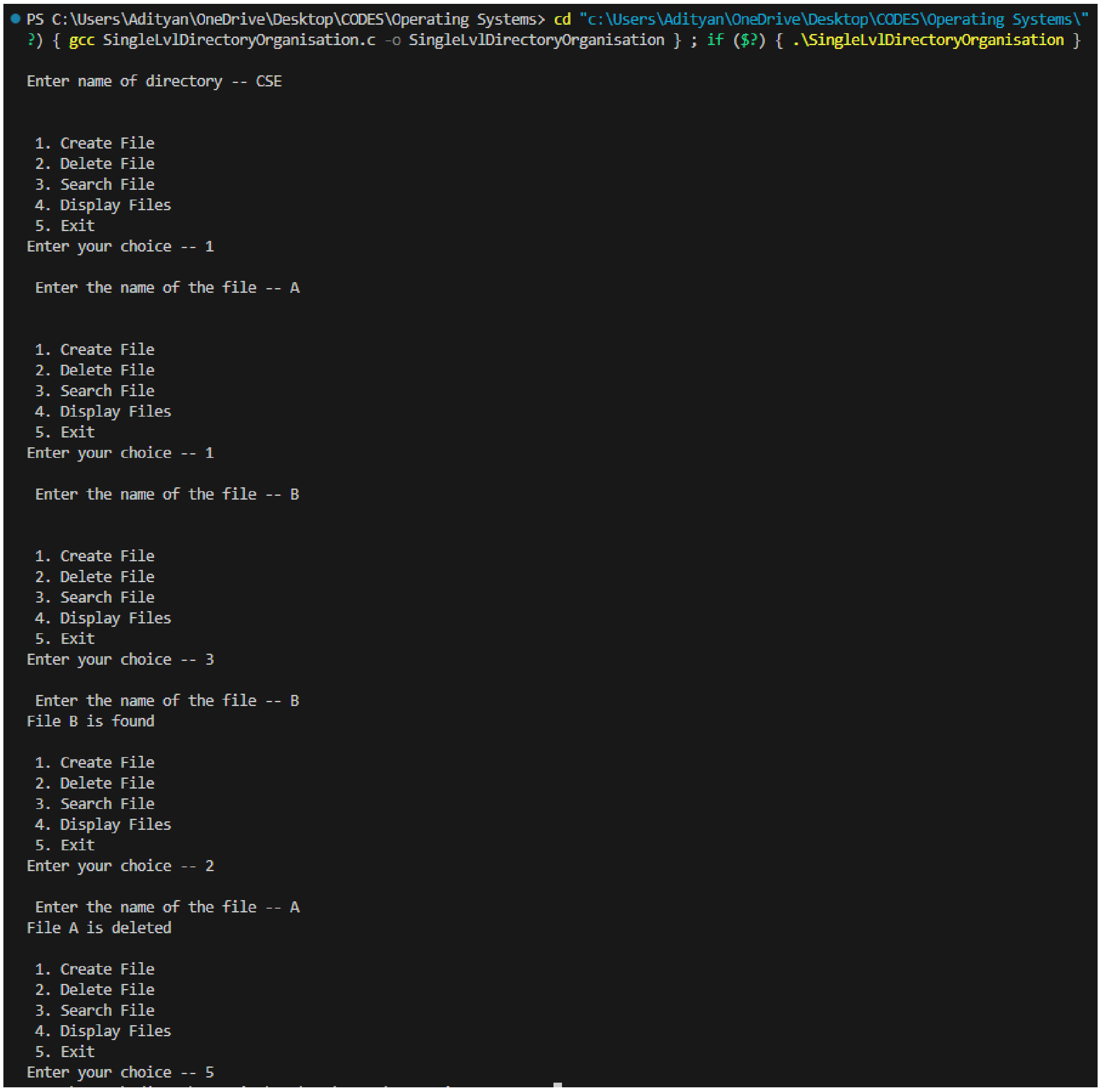
The directory structure is the organization of files into a hierarchy of folders. In a single-level directory system, all the files are placed in one directory. There is a root directory which has all files. It has a simple architecture and there are no sub directories. Advantage of single level directory 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 log 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.

SINGLE LEVEL DIRECTORY ORGANIZATION

Code:

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

struct Directory {
    char dname[10];
    char fname[10][10];
    int fcnt;
};

int main() {
    struct Directory dir;
    int i, ch;
    char f[30];

    dir.fcnt = 0;
    printf("\nEnter name of directory -- ");
    scanf("%s", dir.dname);

    while (1) {
        printf("\n\n 1. Create File\n 2. Delete File\n 3. Search File \n 4. Display Files\n 5.
Exit\nEnter your choice -- ");
        scanf("%d", &ch);

        switch (ch) {
            case 1:
                printf("\n Enter the name of the file -- ");
                scanf("%s", dir.fname[dir.fcnt]);
                dir.fcnt++;
                break;

            case 2:
                printf("\n Enter 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("\n Enter 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("\n Directory Empty");
                else {
                    printf("\n The Files are -- ");
                    for (i = 0; i < dir.fcnt; i++)
                        printf("\t%s", dir.fname[i]);
                }
                break;

            case 5:
                exit(0); // Exiting the program
                break;

            default:
                printf("\nInvalid choice! Please enter a valid option.");
                break;
        }
    }

    return 0;
}
```

OUTPUT:



```
PS C:\Users\Adityan\OneDrive\Desktop\CODES\Operating Systems> cd "c:\Users\Adityan\OneDrive\Desktop\CODES\Operating Systems\"
?) { gcc SingleLvlDirectoryOrganisation.c -o SingleLvlDirectoryOrganisation } ; if ($?) { .\SingleLvlDirectoryOrganisation }

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 -- 3

 Enter the name of the file -- B
File B is 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 -- A
File A is deleted

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

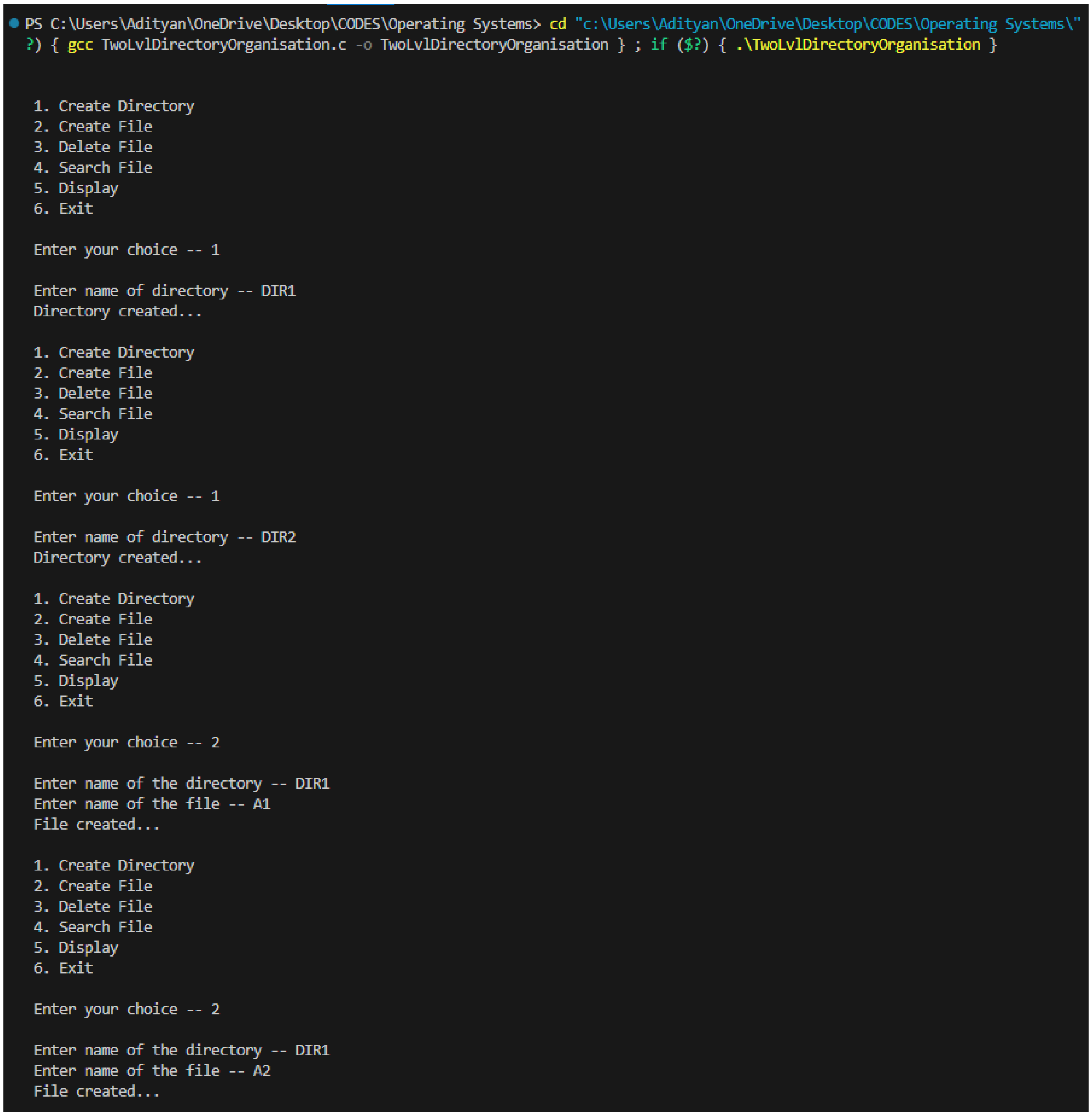
TWO LEVEL DIRECTORY ORGANIZATION

Code:

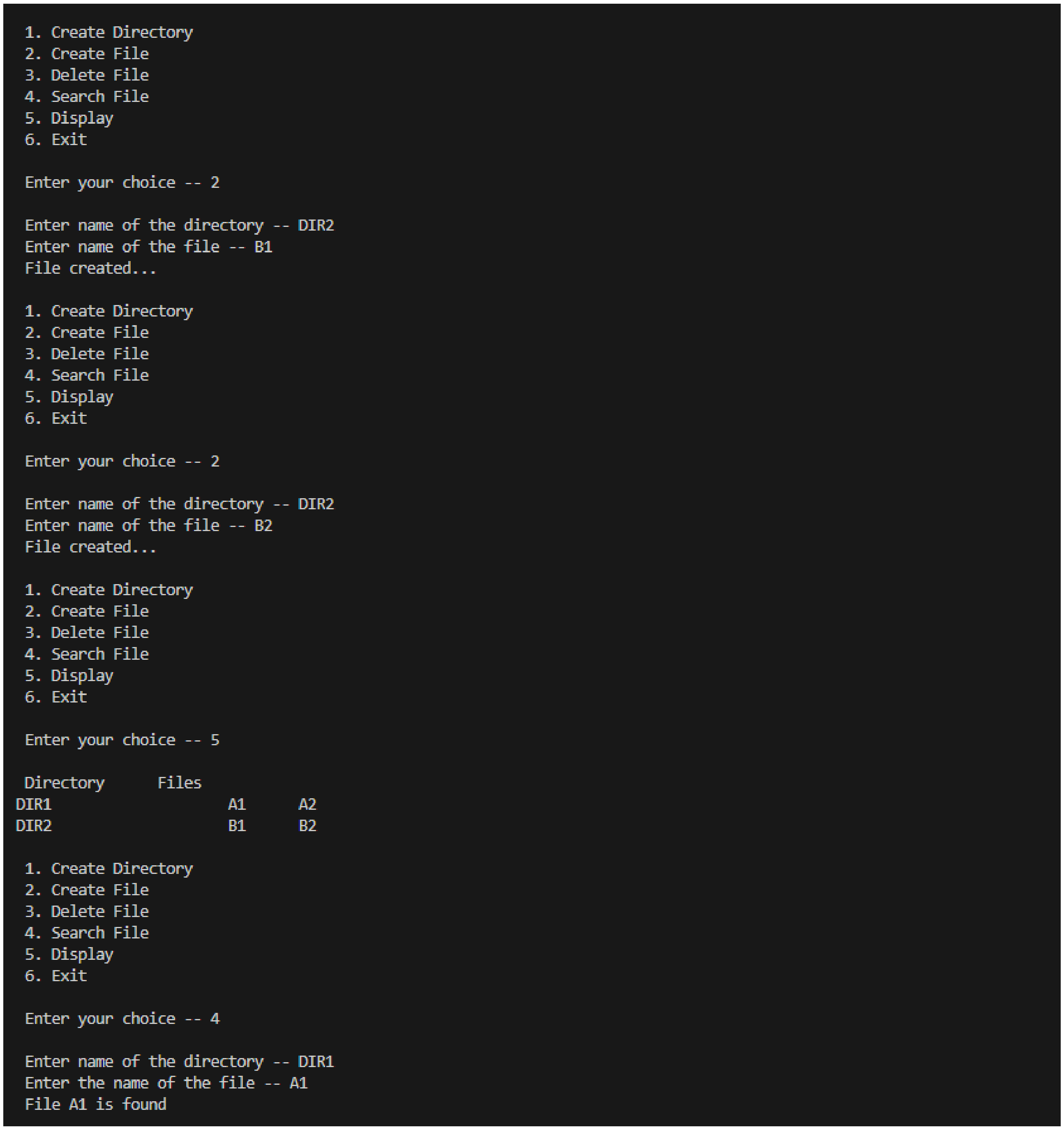
```
#include<stdio.h>
#include<string.h>

struct Directory {
    char dname[10];
    char fname[10][10];
    int fcnt;
};

int main() {
    struct Directory dir[10];
    int i, ch, dcnt, k;
    char f[30], d[30];

    dcnt = 0;

    while (1) {
        printf("\n\n 1. Create Directory\n 2. Create File\n 3. Delete File\n 4. Search File\n 5. Display\n
6. Exit\n\n Enter your choice -- ");
        scanf("%d", &ch);

        switch (ch) {
            case 1:
                printf("\n Enter name of directory -- ");
                scanf("%s", dir[dcnt].dname);
                dir[dcnt].fcnt = 0;
                dcnt++;
                printf(" Directory created...");
                break;

            case 2:
                printf("\n Enter 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("\n Enter 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("\n No Directories ");
                else {
                    printf("\n Directory\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:
                return 0;
        }
    }
}
```

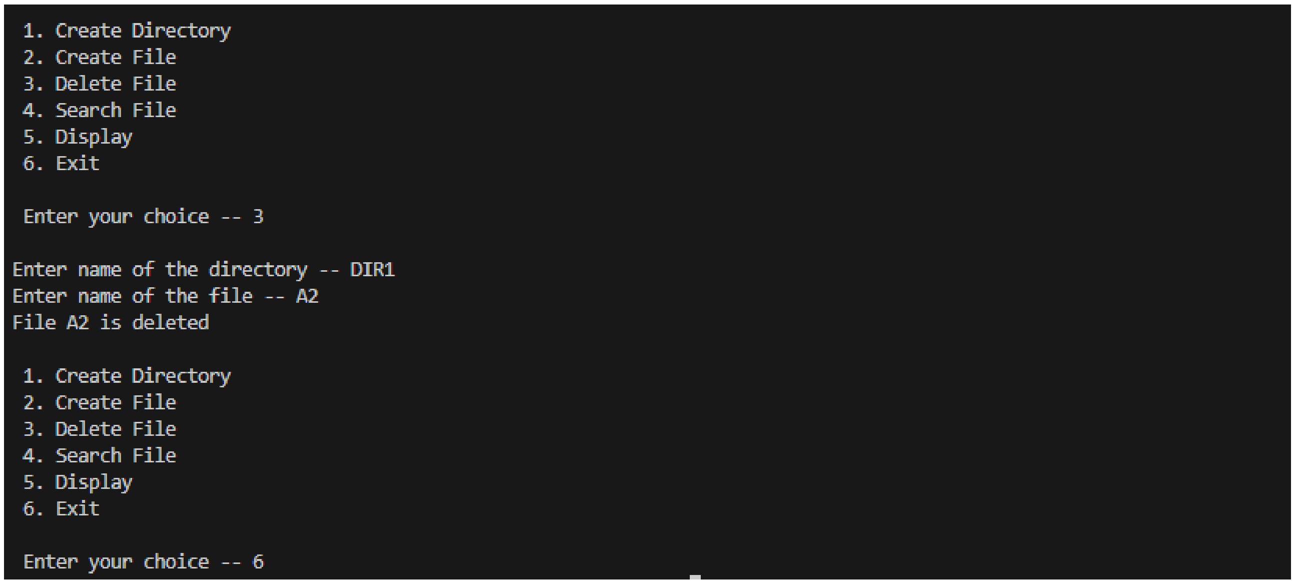
OUTPUT:



```
PS C:\Users\Adityan\OneDrive\Desktop\CODES\Operating Systems> cd "c:\Users\Adityan\OneDrive\Desktop\CODES\Operating Systems\"
?) { gcc TwoLvlDirectoryOrganisation.c -o TwoLvlDirectoryOrganisation } ; if ($?) { .\TwoLvlDirectoryOrganisation }


 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
 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 -- B2
 File created...

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

 Enter your choice -- 5

 Directory      Files
DIR1                   A1      A2
DIR2                   B1      B2

 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 -- DIR1
 Enter the name of the file -- A1
 File A1 is 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
```

HIERARCHICAL DIRECTORY ORGANIZATION

Code:

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

struct tree_element
{
    char name[20];
    int x, y, ftype, lx, rx, nc, level;
    struct tree_element *link[5];
};

typedef struct tree_element node;

void 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 forfile :");
        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;
    }
}

void 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]);
        }
    }
}

int  main()
{
    int gd = DETECT, gm;
    node *root;
    root = NULL;

    create(&root, 0, "root", 0, 639, 320);

    initgraph(&gd, &gm, "c:\\tc\\BGI");
    display(root);

    closegraph();

    return 0;
}
```

OUTPUT:

```
Enter Name of dir/file (under root): ROOT
Enter 1 for Dir / 2 For File : 1
No of subdirectories / files (for ROOT) :2
Enter Name of dir/file (under ROOT):USER 1
Enter 1 for Dir /2 for file:1
No of subdirectories /files (for USER 1):1
Enter Name of dir/file (under USER 1):SUBDIR
Enter 1 for Dir /2 for file:1
No of subdirectories /files (for SUBDIR):2
Enter Name of dir/file (under USER 1):
JAVA Enter 1 for Dir /2 for file:1
No of subdirectories /files (for JAVA): 0
Enter Name of dir/file (under SUBDIR):VB
Enter 1 for Dir /2 for file:1
No of subdirectories /files (for VB): 0

Enter Name of dir/file (under ROOT):USER2
Enter 1 for Dir /2 for file:1
No of subdirectories /files (for USER2):2
Enter Name of dir/file (under ROOT):A
Enter 1 for Dir /2 for file:2
Enter Name of dir/file (under USER2):SUBDIR 2
Enter 1 for Dir /2 for file:1
No of subdirectories /files (for SUBDIR 2):2
```

```
Enter Name of dir/file (under SUBDIR2):PPL
Enter 1 for Dir /2 for file:1
No of subdirectories /files (for PPL):2
Enter Name of dir/file (under PPL):B
Enter 1 for Dir /2 for file:2
Enter Name of dir/file (under PPL):C
Enter 1 for Dir /2 for file:2
Enter Name of dir/file (under SUBDIR):AI
Enter 1 for Dir /2 for file:1
No of subdirectories /files (for AI): 2
Enter Name of dir/file (under AI):D
Enter 1 for Dir /2 for file:2
Enter Name of dir/file (under AI):E


Enter 1 for Dir /2 for file:2
```