OPERATING SYSTEMS LAB

Certificate of Achievement

FILE-1

THIS CERTIFICATE IS PROUDLY

PRESENTED FOR HONOURABLE ACHIEVEMENT TO

MOHD. AFTAB ALAM

AWARDED THIS DAY OF _____

Affix Your Passport Size Photo Here

Name of the Student MOHD. AFTAB ALAM

Roll no. 02013302 7 Class CSE-6A

Examination Centre HMRITM

Date of the Practical Eximination _____

Signature

Signature

DELTA® STATIONERY | INDEX

S.No	Experiment Description	Page No.	Experiment Date	Submission Date	Remarks
1.	WAP to implement CPU Scheduling for FCFS.	1.	24/01/2020		10 30/11/2020
2.	WAP to implement CPU Scheduling for SJF.	4.	30/01/2020		10 4/2/2020
3.	WAP to implement Priority CPU Scheduling.	9.	04/02/2020		10 11/2/2020
4.	WAP to implement Round Robin Scheduling.	13.	11/02/2020		
5.	WAP to implement LRU page replacement algorithm.	17.	18/02/2020		

Experiment : 04

Date 11/02/2020

Page No. 13

Round Robin CPU Scheduling

⊛ **Aim-** Write a Program to implement Round Robin CPU Scheduling.

\# Round Robin is a CPU scheduling algorithm where each process is assigned a fixed time slot in a cyclic way. It is preemptive in nature.

⊛ **Algorithm-** (1) create an array to keep track of remaining burst time of processes. this array is initially a copy of bt[].

(2) Create another array wt[]. initialize this array = 0.

(3) Initialize time : t = 0.

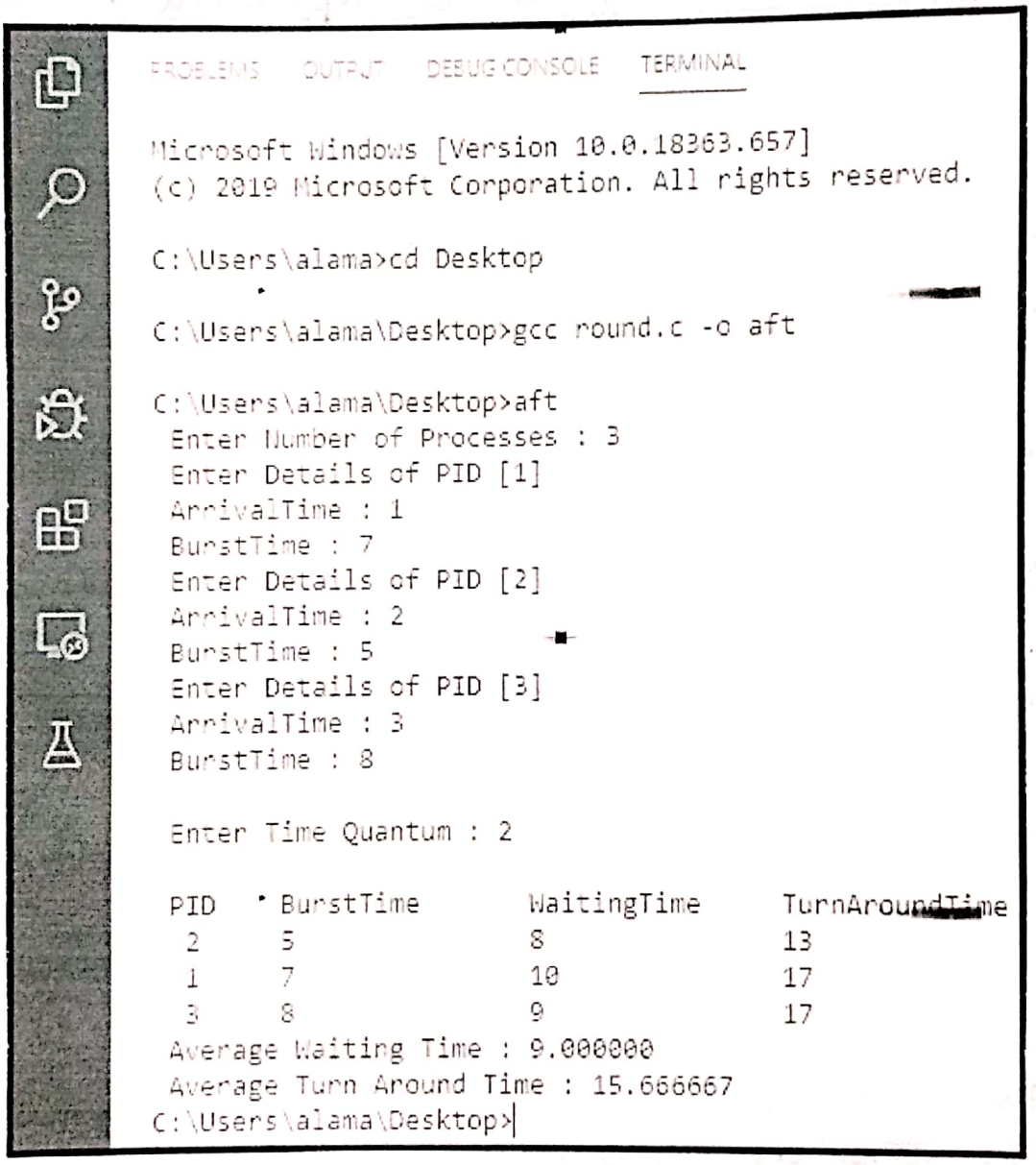
(4) keep traversing all processes while they are not done. do following for i'th process if it is not done yet.

→ if rem_bt[i] > quantum
(i) t = t + quantum; (ii) bt_rem[i] = quantum;

→ else // last cycle for this process.
(i) t = t + bt_rem[i];
(ii) wt[i] = t - bt[i];
(iii) bt_rem[i] = 0; // this process is over.

DELTA

11/02/2020



```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL

Microsoft Windows [Version 10.0.18363.657]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\alama>cd Desktop

C:\Users\alama\Desktop>gcc round.c -o aft

C:\Users\alama\Desktop>aft
 Enter Number of Processes : 3
 Enter Details of PID [1]
 ArrivalTime : 1
 BurstTime : 7
 Enter Details of PID [2]
 ArrivalTime : 2
 BurstTime : 5
 Enter Details of PID [3]
 ArrivalTime : 3
 BurstTime : 8

 Enter Time Quantum : 2

 PID    BurstTime       WaitingTime     TurnAroundTime
  2      5               8               13
  1      7               10              17
  3      8               9               17
 Average Waiting Time : 9.000000
 Average Turn Around Time : 15.666667
C:\Users\alama\Desktop>
```

PROGRAM SCREENSHOT

Experiment : 04

Date 11/02/2020

Page No. 14

Code-0

```
#include<stdio.h>
int main()
{
  int i, x, n, total=0, count=0, tquantum;
  int wt=0, tat=0, at[10], bt[10], temp[20];
  float awt, atat;
  printf("Enter Number of Processes: ");
  scanf("%d", &n);
  x=n;
  for(i=0; i<n; i++)
  {
    printf("Enter Details of PID[%d]\n", i+1);
    printf("Arrival Time: ");
    scanf("%d", &at[i]);
    printf("Burst Time: ");
    scanf("%d", &bt[i]);
    temp[i] = bt[i];
  }
  printf("\n Enter Time Quantum: ");
  scanf("%d", &tquantum);
  printf("\n PID \t Burst Time \t Waiting Time
                \t Turn Around Time");
  for(total=0, i=0; x!=0;)
  {
    if(temp[i] <= tquantum && temp[i]>0)
    {
```

DELTA

```
        total += temp[i];
        temp[i] = 0;
        count = 1;
    }

    else if (temp[i] > 0)
    {
        temp[i] -= tquantum;
        total += tquantum;
    }
    if (temp[i] == 0 && count == 1)
    {
        x--;
    printf("\n %d \t %d \t\t %d \t\t %d",
            i+1, bt[i], total - at[i] - bt[i],
             total - at[i]);
        wt = wt + total - at[i] - bt[i];
        tat = tat + total - at[i];
        count = 0;
    }
    if (i == n-1)
    i = 0;
    else if (at[i+1] <= total)
    i++;
    else
    i = 0;
}
```

```
awt = wt * 1.0 / n;
atat = tat * 1.0 / n;
printf("\n Average Waiting Time : %of", awt);
printf("\n Average Turn Around Time : %of",
                                    atat);
return 0;
}
```

⊛ Result → Round Robin CPU Scheduling is implemented successfully.

Experiment : 05

Date 18/02/2020

Page No. 17

LRU PAGE REPLACEMENT ALGO.

(*) Aim → Write a Program to implement LRU (Least Recent Used) Page Replacement Algorithm.

In OS that use paging for memory management, page replacement algorithm are needed to decide which page needed to be replaced when new page comes in.

In LRU (Least Recently Used) algorithm the page to be replaced is least recently used. The idea is based on locality of reference, the least recently used page is not likely.

(*) Algorithm → Let 'capacity' be the number of pages that memory can hold. Let 'set' be the current set of pages in memory.

(1) Start traversing the pages.

(i) if set holds less pages than capacity.

(a) Insert page into the set one by one until the size of set reaches capacity or all page requests are processed.

(b) simultaneously maintain the recent occured index of each page in a map called indexes.

(c) Increment page fault.

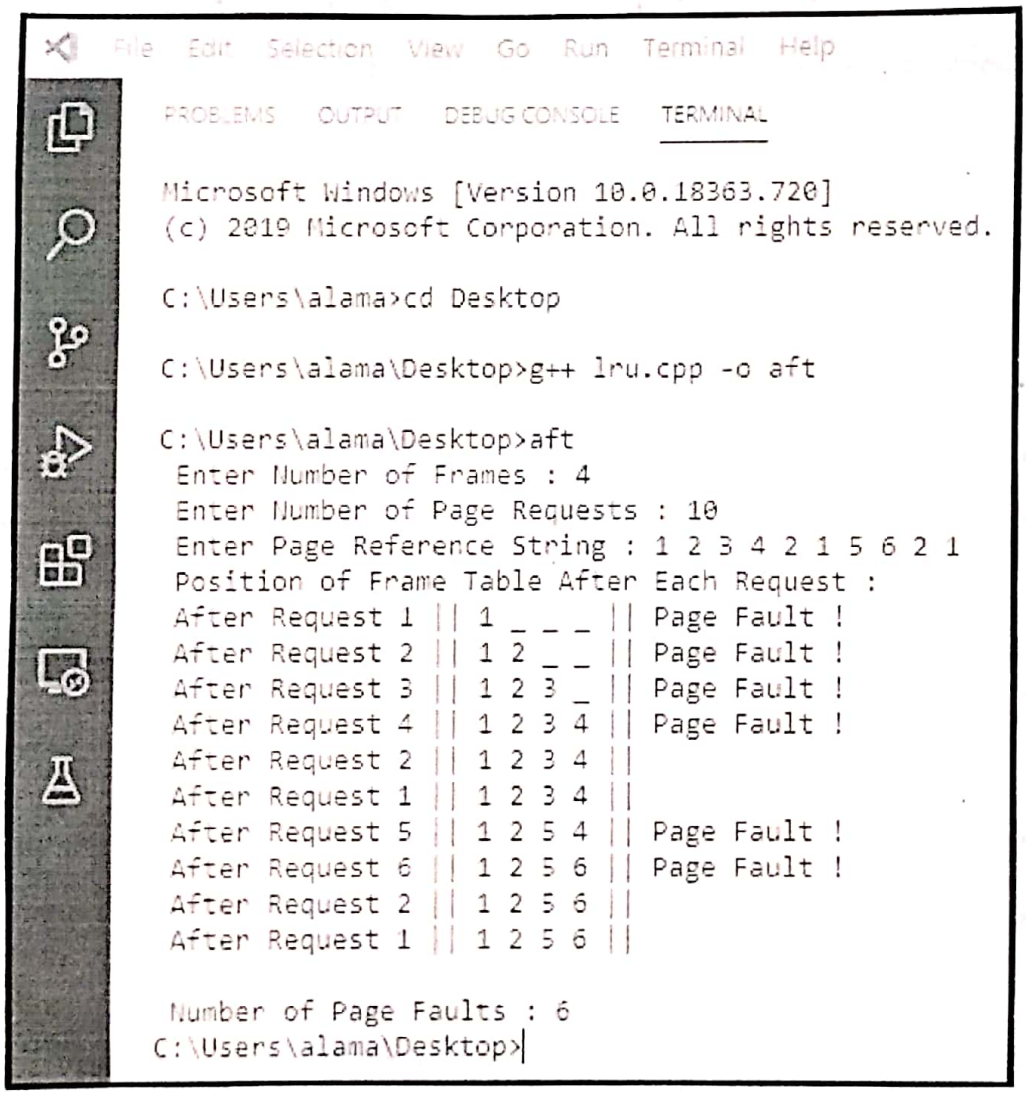
(ii) else

if current page is present in set, do nothing.

~~else~~ else

DELTA

18/02/2020



PROGRAM SCREENSHOT

Experiment : 05

Date 18/02/2020

Page No. 18

ⓐ Find the page in the set that was least recently used. We find it using index array. We basically need to replace the page with minimum index.

ⓑ Replace the found page with current page.

ⓒ Increment page faults.

ⓓ Update index of current page.

② Return page faults.

Code →

```
#include<iostream>
using namespace std;
int present(int tframe[], int nf, int page)
{
 for(int i=0; i<nf; i++)
   if(page == tframe[i])
       return 1;
   return 0;
}
void printtable(int tframe[], int nf)
{
 for(int i=0; i<nf; i++)
 {
   if(tframe[i]==-1)
     cout<<"_ ";
   else
     cout<<tframe[i]<<" ";
 }
 cout<<"\n";
}
```

DELTA

Experiment : 05

Date 18/02/2020

Page No. 19

```
int findpos(int tfrome[], int nf, int pages[], int curr,
                                                int nfs)
{
 for(int i=0; i<nf; i++)
    if(tfrome[i]==-1)
        return i;
    int pos[nf]={0};
    for(int i=0; i<nf; i++)
    {
       pos[i]=-1e9;
       for(int j=curr-1; j>=0; j--)
          if(pages[j]==tfrome[i])
          {
             pos[i]=j;
             break;
          }
    }
    int min1=1000000, retPos=-1;
    for(int i=0; i<nf; i++)
       if(min1>pos[i])
       {
          min1=pos[i];
          retPos=i;
       }
    return retPos;
}
int main()
{int n, nf, i, pos=0;
 cout<<"Enter Number Of Frames: ";
```

```
cin>>nf;
int tframe[nf];
for(i=0;i<nf;i++)
{ tframe[i]=-1; }
cout<<"Enter Number of Page Requests : ";
cin>>n;
int pages[n];
cout<<"Enter Page Reference String : ";
for(i=0;i<n;i++)
{ cin>>pages[i]; }
int count1=0;
cout<<"Position of Frame Table After Each Request : \n";
for(i=0;i<n;i++)
{ cout<<"After Request "<<pages[i]<<" ||";
 if(!present(tframe,nf,pages[i]))
{ int pos=findpos(tframe,nf,pages,i,n);
  tframe[pos]=pages[i];
  printtable(tframe,nf);
  cout<<"Page Fault ! \n";
  count1++;
  continue;
 }
 printtable(tframe,nf);
 cout<<"\n";
}
cout<<"\n Number of Page Faults : "<<count1;
}
```

Result → LRU Page Replacement Algorithm is implemented successfully.

OPERATING SYSTEMS LAB

Certificate of Achievement

FILE-2

THIS CERTIFICATE IS PROUDLY

PRESENTED FOR HONOURABLE ACHIEVEMENT TO

MOHD. AFTAB ALAM

AWARDED THIS DAY OF _____

Affix Your Passport Size Photo Here

Name of the Student MOHD. AFTAB ALAM

Roll no. 02013B027 Class CSE-6A

Examination Centre HMRITM

Date of the Practical Eximination _____

Signature

Signature

DELTA® STATIONERY | INDEX

S.No	Experiment Description	Page No.	Experiment Date	Submission Date	Remarks
6.	WAP to implement FIFO page replacement algorithm.	1.	25/02/2020		
7.	WAP to implement Optimal page replacement algorithm.	4.	25/02/2020		
8.	WAP to implement First Fit, Best Fit, Worst Fit Algorithm for Memory Management.	8.	17/03/2020		

Experiment : 06

Date 25/02/2020

Page No. 1

FIFO PAGE REPLACEMENT ALGO.

(*) Aim- Write a program to implement FIFO (First In First Out) Page Replacement Algorithm.

\# this is the simplest page replacement algorithm. In FIFO, the OS keeps track of all pages in the memory in a queue, oldest page is in the front of the queue. When a page needs to be replaced, page in the front of the queue is selected for removal.

(*) Algorithm- let 'capacity' be the no. of pages that memory can hold. let 'set' be the current set of pages in memory.

(1) Start traversing the pages.

(i) if set holds less pages than capacity.

(a) Insert page into the set one by one until the size of set reaches capacity or all page requests are processed.

(b) simultaneously maintain the pages in the queue to perform FIFO.

(c) Increment page fault.

(ii) else

if current page is present in set, do nothing.

else

(a) Remove the first page from the queue as it was the first to be entered in the memory.

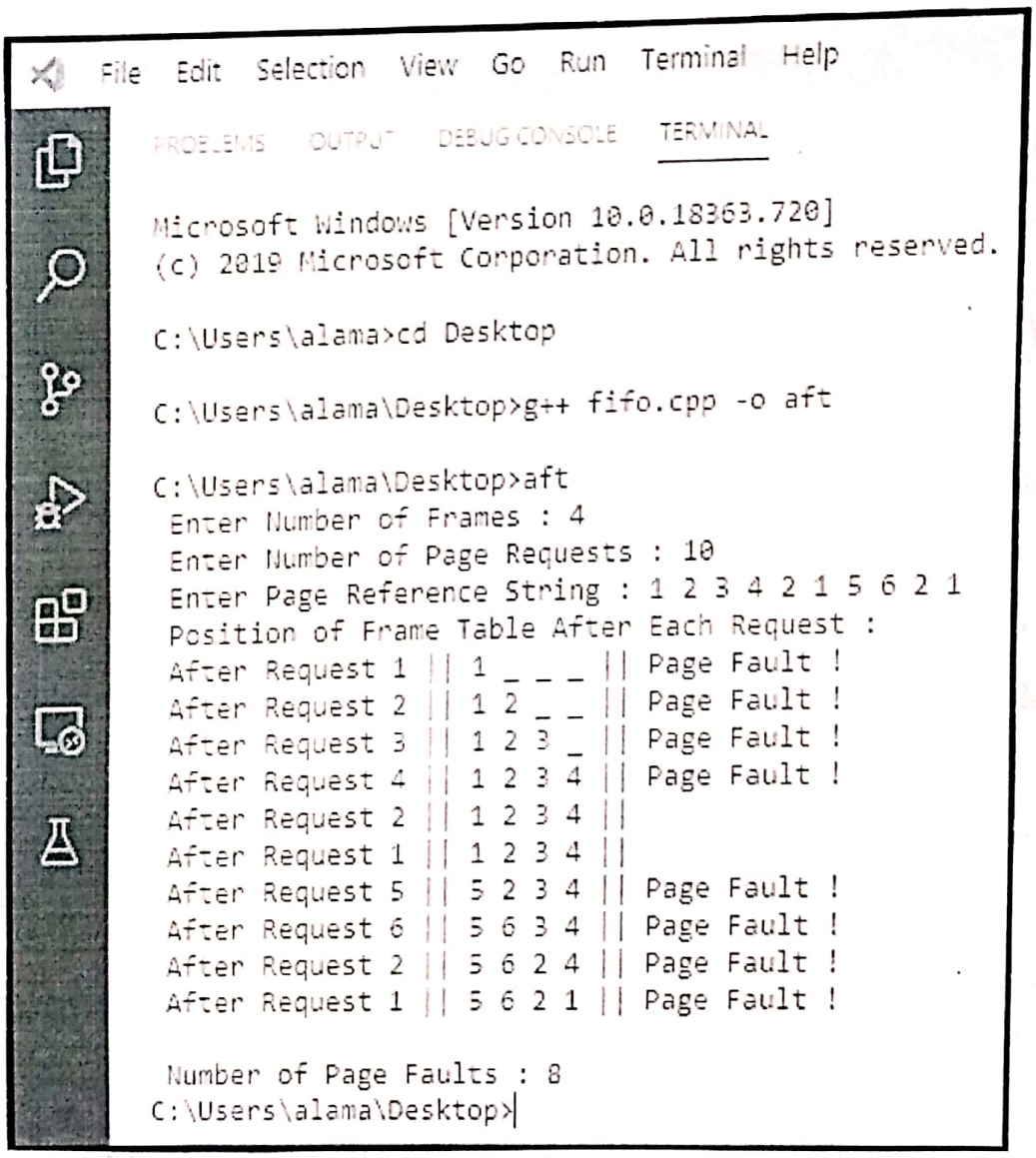
(b) Replace the first page in the queue with the current page in the string. (c) Store current page in the queue.

(d) Increment page faults.

(2) Return page faults.

DELTA

25/02/2020



PROGRAM SCREENSHOT

Code-

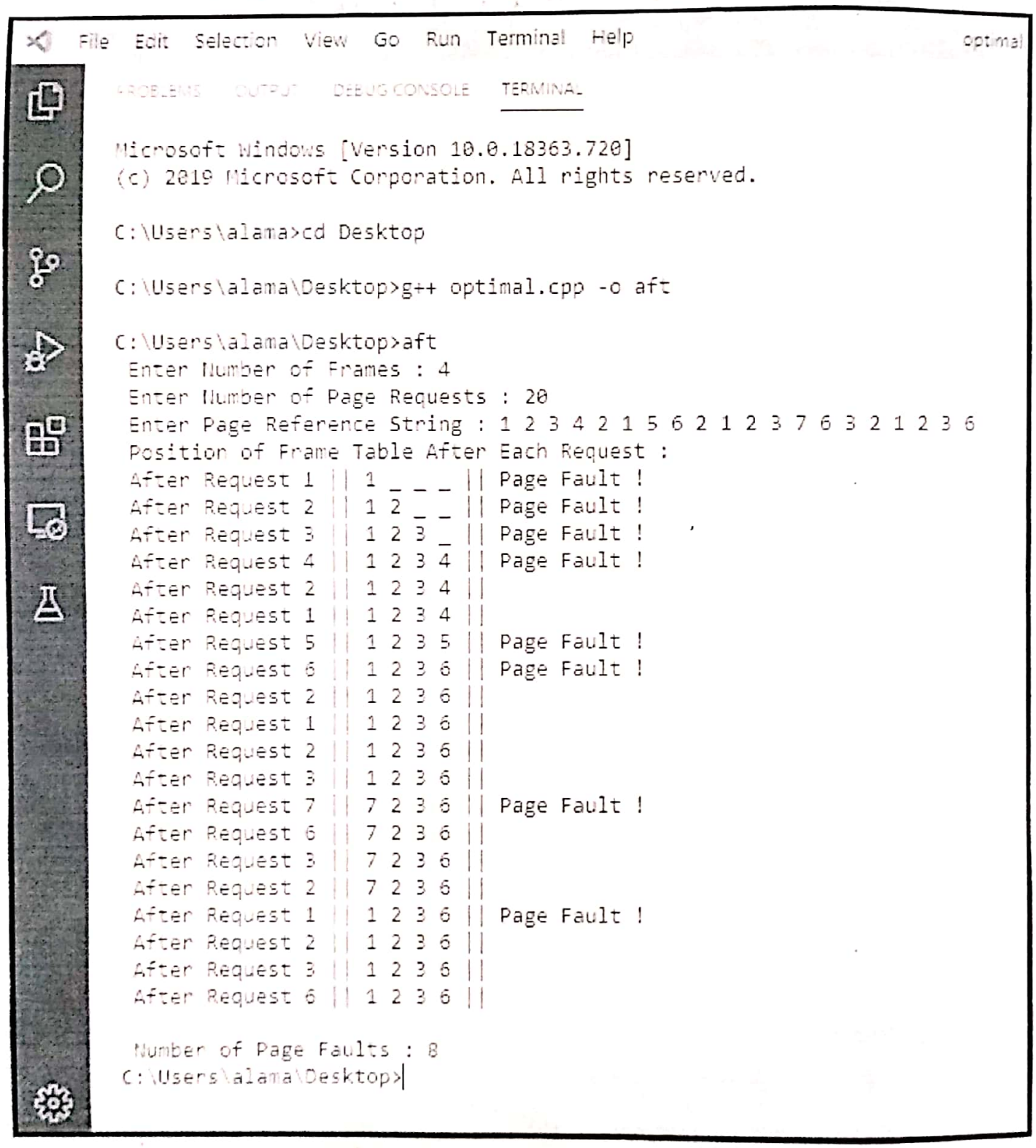
```
#include<iostream>
using namespace std;
int present(int tframe[], int nf, int page)
{
  for(int i=0; i<nf; i++)
      if(page == tframe[i])
          return 1;
  return 0;
}
void printtable(int tframe[], int nf)
{
  for(int i=0; i<nf; i++)
  {
    if(tframe[i]==-1)
      cout<<"_ ";
    else
      cout<<tframe[i]<<" ";
  }
  printf("|| ");
}
int main()
{int n,nf,i,pos=0;
  cout<<"Enter Number of Frames: ";
  cin>>nf;
  int tframe[nf];
  for(i=0;i<nf;i++)
{ tframe[i]=-1; }
```

```
cout<<"Enter Number of Page Requests :";
cin>>n;
int pages[n];
cout<<"Enter Page Reference String :";
for(i=0;i<n;i++)
{ cin>> pages[i]; }
int count1=0;
cout<<"Position of Frame Table After Each Request :\n";
for(i=0;i<n;i++)
{ cout<<"After Request "<< pages[i]<<" ||";
 if(!present(tframe,nf,pages[i]))
 { tframe[pos]=pages[i];
   pos=(pos+1)%nf;
   printtable(tframe,nf);
   cout<<"Page Fault !\n";
   count1++;
   continue;
 }
 printtable(tframe,nf);
 cout<<"\n";
}
cout<<"\n Number of Page Faults :"<<count1;
}
```

(*) **Result** → FIFO Page Replacement Algorithm is implemented successfully.

DELTA®

25/02/2020



#PROGRAM SCREENSHOT

Experiment : 07

Date 25/02/2020

Page No. 4

OPTIMAL PAGE REPLACEMENT ALGO.

(*) **Aim** – Write a Program to implement Optimal Page Replacement Algorithm.

\# in this algorithm, OS replaces the page that will not be used for the longest period of time in future. Optimal page replacement is perfect but not possible in practice as the operating system cannot know future requests.

(*) **Algorithm** – the idea is simple, for every reference, we do following :

(1) if reference page is already present, increment hit count.

(2) if not present, find if a page that is never referenced in future ; If such a page exists, replace this page with new page ; If no such page exists, find a page that is referenced farthest in future. Replace this page with new page.

\# **Code** –

```
#include<iostream>
using namespace std;
int present(int sframe[], int nf, int page)
{ for(int i=0; i<nf; i++)
    if(page==sframe[i])
      return 1;
  return 0;
}
```

DELTA

Experiment : 07

Date 25/02/2020

Page No. 5

```
void printtable (int tfrome [], int nf)
{ for(int i=0; i<nf; i++)
  { if (tfrome[i]==-1)
        cout<<"_ ";
    else
        cout<<tfrome[i]<<" ";
  }
  cout<<"||";
}
int findpos( int tfrome[], int nf, int pages[], int curr,
                                    int np)
{ int i, j;
  for( i=0; i<nf ; i++)
  { if ( tfrome[i]==-1)
        return i;
  }
  int pos[nf]={0};
  for( i=0; i<nf ; i++)
  { pos[i]=1e8;
    for( j= curr+1; j<np; j++)
    { if ( pages[j]==tfrome[i])
      { pos[i]=j;
        break;
      }
    }
  }
  int not1=-1;
  int returnpos=-1;
```

```
for(i=0;i<nf;i++)
{ if(pos[i] > max1)
   { max1 = pos[i];
     returnpos = i;
   }
}
return returnpos;
}
int main()
{ int n, nf, i, pos=0;
  cout<<"Enter Number of Frames:";
  cin>>nf;
  int tframe[nf];
  for(i=0; i<nf; i++)
  { tframe[i]=-1; }
  cout<<"Enter Number of Page Requests:";
  cin>>n;
  int pages[n];
  cout<<"Enter Page Reference String:";
  for(i=0; i<n; i++)
  { cin>>pages[i]; }
  int count1=0;
  cout<<"Position of Frame Table After Each Request:\n";
  for(i=0; i<n; i++)
  { cout<<"After Request"<<pages[i]<<" ||";
    if(!present(tframe, nf, pages[i]))
    {
```

```
        int pos = findpos(tframe, nf, pages, i, n);
        tframe[pos] = pages[i];
        printtable(tframe, nf);
        cout<<"Page Fault!\n";
        count1++;
        continue;
    }
    printtable(tframe, nf);
    cout<<"\n";
}
cout<<"\nNumber of Page Faults :"<<count1;
}
```

(*) Result → Optimal Page Replacement Algorithm is implemented successfully.

Experiment : 08

Date 17/03/2020

Page No. 8

BEST FIT, FIRST FIT, WORST FIT ALGORITHM for MEMORY MANAGEMENT

(*) Aim → Write a program to implement FirstFit, BestFit and WorstFit Algorithm for memory management

(#) FIRST FIT — the partition is allocated which is first sufficient from the top of main memory.

(*) Algorithm → (1) Input memory blocks with size and processes with size.

(2) Initialize all memory blocks as free.

(3) Start by picking each process and check if it can be assigned to current block.

(4) If size of process <= size of block if yes then assign and check for next process.

(5) If not then keep checking the further blocks.

(#) BEST FIT — it allocates the process to a partition which is the smallest sufficient partition among the free available partitions.

(*) Algorithm → (1) & (2) same as above. (3) Start by picking each process and find the minimum block size that can be assigned to current process that is, find min(blockSize[1], blockSize[2], ... blockSize[n]) > processSize[current], if found then assign it to the current process.

(4) If not then leave that process and keep checking the further processes.

DELTA

Experiment : 08

Date 17/03/2020

Page No. 9

(iii) WORST FIT – it allocates a process to the partition which is largest sufficient among the freely available partitions available in the main memory. if a large process comes at a later stage, then memory will not have to space to accommodate it.

(iv) Algorithm – (1) & (2) same as previous. (3) start by picking each process and find the maximum block size that can be assigned to current process that is, find max (blockSize[1], blocksize[2], ... blocksize[n]) > processSize[current], if found then assign it to the current process.

(4) If not then leave that process and keep checking the further processes.

Code –

```
#include<iostream>
#include<stdio.h>
using namespace std;
int main()
{
 int blosize[5], prosize[5], temp[5], i=0, j=0, k=0, n, m,
 count=0, ch;
 cout<<"Enter Number of Processes :";
 cin>>n;
 cout<<"Enter Number of Blocks :";
 cin>>m;
 cout<<"Enter the SIZE of Processes :";
```

DELTA

```
for(i=0;i<n;i++)
cin>>prosize[i];
cout<<"Enter the SIZE of Blocks:";
for(j=0;j<m;j++)
cin>>blosize[j];
for(k=0;k<n;k++)
temp[k]=-1;
cout<<"Options --- 1.FirstFit \t 2.BestFit \t 3.WorstFit";
cout<<"Enter Your Choice:";
cin>>ch;
switch(ch)
{
case 1: for(i=0;i<n;i++)
        {
        for(j=0;j<m;j++)
        {
        if(prosize[i]<=blosize[j]&&blosize[j]!=0)
        {
          temp[i]=j;
          blosize[j]-=prosize[i];
          break;
        }
      }
    }
  }
  break;
case 2: for(i=0;i<n;i++)
        {
          int diff=9999;
```

```
    for(j=0; j<m; j++)
    {
      if(prosize[i]<=blosize[j]&blosize[j]!=0&&
      (blosize[j]-prosize[i])<diff)
      {
        diff=blosize[j]-prosize[i];
        count=j;
      }
    }
    if(diff!=0)
    {
      blosize[count--]=prosize[i];
      temp[i]=count;
    }
  }
  break;
case3: for(i=0;i<n;i++)
       {
         int diff=0;
         for(j=0;j<m;j++)
         {
           if(prosize[i]<=blosize[j]&blosize[j]!=0&&
         (blosize[j]-prosize[i]>diff)
       { diff=blosize[j]-prosize[i];
         count=j;
       }
     }
```

```
if(diff!=0)
{
  blosize[count]-=prosize[i];
  temp[i]=count;
}
}
break;
default : cout<<"Wrong Choice!"; break;
}
cout<<"\n Process NO.\tProcess Size\tBlock NO.\n";
for(i=0;i<n;i++)
{
 cout<<" "<<i+1<<"\t\t"<<prosize[i]<<"\t\t";
 if(temp[i]!=-1)
 cout<<temp[i]+1;
 else
 cout<<"Not Allowed";
 cout<<endl;
}
 return 0;
}
```

(*) Result – e First Fit, Best Fit, Worst Fit Algorithm for memory Management is implemented successfully.

DELTA