Department of Information Technology
Delhi Technological University

Operating System Lab (IT-204)

Session 2020/21(IT-4th semester Section-2)

Submitted by:

Varun Kumar

Roll no: 2K19/IT/140

Group: G3

Submitted to:

Mr. Jasraj Meena

Assistant Professor

Dept. of IT, DTU

INDEX

Sno.	Name of the Program	Date of Experiment	Remarks
1.	Do the case study of basic Operating System commends related to File and Process.	08/01/2021	
2.	Write a Program to implement the First in First Out (FIFO) CPU Scheduling Algorithm.	15/01/2021	
3.	Write a Program to implement the Shortest Job First (SJF) CPU Scheduling Algorithm.	22/01/2021	
4.	Write a Program to Implement the Priority Scheduling Algorithm.	29/01/2021	
5.	Write a program to implement the Round Robin CPU Scheduling Algorithm.	5/02/2021	
6.	Write a Program to Implement the Longest Remaining Time First (LRTF) CPU Scheduling Algorithm.	26/02/2021	
7.	Write a Program to implement Bankers Algorithm.	5/03/2021	
8.	Write a program to implement Producer- Consumer Problem.	12/03/2021	
9.	Write a program to implement the First-In-First-Out (FIFO) Page Replacement Algorithm.	26/03/2021	

10.	Write a program to implement LRU Page Replacement algorithm.	9/04/2021	
11.	Write a program to implement Dining Philosophers problem.	16/04/2021	
12.	Write a program to implement the FCFS (First Come First Serve) Disk Scheduling Algorithms.	21/05/2021	
13.	Write a program to implement the SSTF (Shortest Seek Time First) Disk Scheduling Algorithim.	21/05/2021	

Problem-1: Case Study: Do the case study of basic Operating System commends related to File and Process.

Create a directory

```
PS C:\Users\NISHANT\Desktop> mkdir ./NewFolder


    Directory: C:\Users\NISHANT\Desktop


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----         1/8/2021  12:17 PM                NewFolder
```

Move a file

```
PS C:\Users\NISHANT\Desktop> mv ./Doc.docx ./NewFolder
PS C:\Users\NISHANT\Desktop>
```

Copy a file to another directory

```
PS C:\Users\NISHANT\Desktop> cp ./present.pptx ./NewFolder
PS C:\Users\NISHANT\Desktop>
```

Change Directory

```
PS C:\Users\NISHANT\Desktop> cd "./NewFolder"
PS C:\Users\NISHANT\Desktop\NewFolder>
```

Renaming a file

```
PS C:\Users\NISHANT\Desktop\NewFolder> mv ./present.pptx newppt.pptx
PS C:\Users\NISHANT\Desktop\NewFolder>
```

Listing Directory

```
PS C:\Users\NISHANT\Desktop\NewFolder> ls


    Directory: C:\Users\NISHANT\Desktop\NewFolder


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----         1/8/2021  12:21 PM              0 Doc.docx
-a----         1/8/2021  12:25 PM              0 newppt.pptx
```

Clear the Terminal screen of all previous commands

```
PS C:\Users\NISHANT\Desktop\NewFolder> clear
```

Remove an empty directory

```
PS C:\Users\NISHANT\Desktop> rmdir ./NewFolder

Confirm
The item at C:\Users\NISHANT\Desktop\NewFolder has children and the Recurse parameter was not specified. If you
continue, all children will be removed with the item. Are you sure you want to continue?
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"): y
```

Problem-2: Write a Program to implement the First in First Out (FIFO) CPU Scheduling Algorithm with arrival time.

```
#include<iostream>
using namespace std;

void findWaitingTime(int processes[], int n, int bt[],
                                                    int waiting[], int at[])
{
        int service_time[n];
        service_time[0] = 0;
        waiting[0] = 0;

        for (int i = 1; i < n ; i++)
        {
                service_time[i] = service_time[i-1] + bt[i-1];

                waiting[i] = service_time[i] - at[i];

                if (waiting[i] < 0)
                        waiting[i] = 0;
        }
```

```
}

void findTurnAroundTime(int processes[], int n, int bt[],
                                                                    int waiting[], int tat[])
{
        for (int i = 0; i < n ; i++)
                tat[i] = bt[i] + waiting[i];
}

void findavgTime(int processes[], int n, int bt[], int at[])
{
        int waiting[n], tat[n];

        findWaitingTime(processes, n, bt, waiting, at);

        findTurnAroundTime(processes, n, bt, waiting, tat);

        cout << "Processes " << " Burst Time " << " Arrival Time "
                << " Waiting Time " << " Turn-Around Time "
                << " Completion Time \n";
        int total_wt = 0, total_tat = 0;
        for (int i = 0 ; i < n ; i++)
        {
                total_wt = total_wt + waiting[i];
                total_tat = total_tat + tat[i];
                int compl_time = tat[i] + at[i];
                cout << " " << i+1 << "\t\t" << bt[i] << "\t\t"
                        << at[i] << "\t\t" << waiting[i] << "\t\t "
                        << tat[i] << "\t\t " << compl_time << endl;
        }

        cout << "Average waiting time = "
                << (float)total_wt / (float)n;
        cout << "\nAverage turn around time = "
```

```
                << (float)total_tat / (float)n;
}

int main()
{
        int processes[] = {1, 2, 3, 4, 5};
        int n = 5;

        int burst_time[] = {5, 9, 6, 8, 3};

        int arrival_time[] = {0, 3, 6, 9, 12};

        findavgTime(processes, n, burst_time, arrival_time);

        return 0;
}
```

```
PS C:\Users\NISHANT\Documents> .\t.exe
Processes  Burst Time  Arrival Time  Waiting Time  Turn-Around Time  Completion Time
 1              5              0              0              5              5
 2              9              3              2              11             14
 3              6              6              8              14             20
 4              8              9              11             19             28
 5              3              12             16             19             31
Average waiting time = 7.4
Average turn around time = 13.6
```

Problem-3: Write a Program to implement the Shortest Job First (SJF) CPU Scheduling Algorithm.

```
#include<iostream>
using namespace std;
int mat[10][6];

void swap(int *a, int *b)
{
	int temp = *a;
	*a = *b;
	*b = temp;
}

void arrangeArrival(int num, int mat[][6])
{
	for(int i=0; i<num; i++)
	{
		for(int j=0; j<num-i-1; j++)
		{
			if(mat[j][1] > mat[j+1][1])
			{
				for(int k=0; k<5; k++)
```

```
                                        {
                                                swap(mat[j][k], mat[j+1][k]);
                                        }
                                }
                        }
                }
}

void completionTime(int num, int mat[][6])
{
        int temp, val;
        mat[0][3] = mat[0][1] + mat[0][2];
        mat[0][5] = mat[0][3] - mat[0][1];
        mat[0][4] = mat[0][5] - mat[0][2];

        for(int i=1; i<num; i++)
        {
                temp = mat[i-1][3];
                int low = mat[i][2];
                for(int j=i; j<num; j++)
                {
                        if(temp >= mat[j][1] && low >= mat[j][2])
                        {
                                low = mat[j][2];
                                val = j;
                        }
                }
                mat[val][3] = temp + mat[val][2];
                mat[val][5] = mat[val][3] - mat[val][1];
                mat[val][4] = mat[val][5] - mat[val][2];
                for(int k=0; k<6; k++)
                {
                        swap(mat[val][k], mat[i][k]);
                }
```

```
        }
}

int main()
{
        int num, temp;

        cout<<"Enter number of Process: ";
        cin>>num;

        cout<<"...Enter the process ID...\n";
        for(int i=0; i<num; i++)
        {
                cout<<"...Process "<<i+1<<"...\n";
                cout<<"Enter Process Id: ";
                cin>>mat[i][0];
                cout<<"Enter Arrival Time: ";
                cin>>mat[i][1];
                cout<<"Enter Burst Time: ";
                cin>>mat[i][2];
        }

        cout<<"Before Arrange...\n";
        cout<<"Process ID\tArrival Time\tBurst Time\n";
        for(int i=0; i<num; i++)
        {
                cout<<mat[i][0]<<"\t\t"<<mat[i][1]<<"\t\t"<<mat[i][2]<<"\n";
        }

        arrangeArrival(num, mat);
        completionTime(num, mat);
        cout<<"Final Result...\n";
        cout<<"Process ID\tArrival Time\tBurst Time\tWaiting Time\tTurnaround Time\n";
        for(int i=0; i<num; i++)
```

```
        {

        cout<<mat[i][0]<<"\t\t"<<mat[i][1]<<"\t\t"<<mat[i][2]<<"\t\t"<<mat[i][4]<<"\t\t"<<mat[i][5]<<"\n";

        }
}
```

```
PS C:\Users\NISHANT\Documents> .\t.exe
Enter number of Process: 4
...Enter the process ID...
...Process 1...
Enter Process Id: 1
Enter Arrival Time: 2
Enter Burst Time: 3
...Process 2...
Enter Process Id: 2
Enter Arrival Time: 0
Enter Burst Time: 4
...Process 3...
Enter Process Id: 3
Enter Arrival Time: 4
Enter Burst Time: 2
...Process 4...
Enter Process Id: 4
Enter Arrival Time: 5
Enter Burst Time: 4
Before Arrange...
Process ID      Arrival Time    Burst Time
1               2               3
2               0               4
3               4               2
4               5               4
Final Result...
Process ID      Arrival Time    Burst Time      Waiting Time    Turnaround Time
2               0               4               0               4
3               4               2               0               2
1               2               3               4               7
4               5               4               4               8
```

Problem-4 Write a Program to Implement the Priority Scheduling Algorithm.

```
#include<bits/stdc++.h>
using namespace std;

struct Process
{
	int pid;
	int bt;
	int priority;
};

bool comparison(Process a, Process b)
{
	return (a.priority > b.priority);
}

void findWaitingTime(Process proc[], int n,
                                        int wt[])
{
	wt[0] = 0;

	for (int i = 1; i < n ; i++ )
		wt[i] = proc[i-1].bt + wt[i-1] ;
}

void findTurnAroundTime( Process proc[], int n,
                                                int wt[], int tat[])
```

```
{
	for (int i = 0; i < n ; i++)
		tat[i] = proc[i].bt + wt[i];
}

void findavgTime(Process proc[], int n)
{
	int wt[n], tat[n], total_wt = 0, total_tat = 0;

	findWaitingTime(proc, n, wt);

	findTurnAroundTime(proc, n, wt, tat);

	cout << "\nProcesses "<< " Burst time "
		<< " Waiting time " << " Turn around time\n";

	for (int i=0; i<n; i++)
	{
		total_wt = total_wt + wt[i];
		total_tat = total_tat + tat[i];
		cout << " " << proc[i].pid << "\t\t"
			<< proc[i].bt << "\t " << wt[i]
			<< "\t\t " << tat[i] <<endl;
	}

	cout << "\nAverage waiting time = "
		<< (float)total_wt / (float)n;
	cout << "\nAverage turn around time = "
		<< (float)total_tat / (float)n;
}
```

```
void priorityScheduling(Process proc[], int n)
{
        sort(proc, proc + n, comparison);

        cout<< "Order in which processes gets executed \n";
        for (int i = 0 ; i < n; i++)
                cout << proc[i].pid <<" " ;

        findavgTime(proc, n);
}

int main()
{
        Process proc[] = {{1, 10, 2}, {2, 5, 0}, {3, 8, 1} ,{4 , 6, 3}};
        int n = 4;
        priorityScheduling(proc, n);
        return 0;
}
```

```
PS C:\Users\NISHANT\Documents> .\t.exe
Order in which processes gets executed
4 1 3 2
Processes  Burst time  Waiting time  Turn around time
 4               6          0               6
 1               10         6               16
 3               8          16              24
 2               5          24              29

Average waiting time = 11.5
Average turn around time = 18.75
```

Problem 5: Write a program to implement the Round Robin CPU Scheduling Algorithm.

```
#include<iostream>
using namespace std;

void findWaitingTime(int processes[], int n,
                        int bt[], int wt[], int quantum)
{
        int rem_bt[n];
        for (int i = 0 ; i < n ; i++)
                rem_bt[i] = bt[i];

        int t = 0;

        while (1)
        {
                bool done = true;

                for (int i = 0 ; i < n; i++)
                {
                        if (rem_bt[i] > 0)
                        {
                                done = false;

                                if (rem_bt[i] > quantum)
                                {
                                        t += quantum;

                                        rem_bt[i] -= quantum;
                                }

                                else
                                {
                                        t = t + rem_bt[i];

                                        wt[i] = t - bt[i];

                                        rem_bt[i] = 0;
                                }
                        }
                }

                if (done == true)
                break;
        }
}

void findTurnAroundTime(int processes[], int n,
                                                int bt[], int wt[], int tat[])
{
        for (int i = 0; i < n ; i++)
                tat[i] = bt[i] + wt[i];
}

void findavgTime(int processes[], int n, int bt[],int quantum)
```

```
{
	int wt[n], tat[n], total_wt = 0, total_tat = 0;

	findWaitingTime(processes, n, bt, wt, quantum);

	findTurnAroundTime(processes, n, bt, wt, tat);

	cout << "Processes "<< " Burst time "
		<< " Waiting time " << " Turn around time\n";

	for (int i=0; i<n; i++)
	{
		total_wt = total_wt + wt[i];
		total_tat = total_tat + tat[i];
		cout << " " << i+1 << "\t\t" << bt[i] <<"\t "
			<< wt[i] <<"\t\t " << tat[i] <<endl;
	}

	cout << "Average waiting time = "
		<< (float)total_wt / (float)n;
	cout << "\nAverage turn around time = "
		<< (float)total_tat / (float)n;
}

int main()
{
	int processes[] = { 1, 2, 3, 4, 5};
	int n = 5;

	int burst_time[] = {10, 5, 8, 6, 2};

	int quantum = 2;
	findavgTime(processes, n, burst_time, quantum);
	return 0;
}
```

```
PS C:\Users\NISHANT\Documents> .\t.exe
Processes  Burst time  Waiting time  Turn around time
 1               10          21               31
 2               5           16               21
 3               8           21               29
 4               6           19               25
 5               2           8                10
Average waiting time = 17
Average turn around time = 23.2
```

Problem 6: Write a Program to Implement the Longest Remaining Time First (LRTF) CPU Scheduling Algorithm

```
#include <bits/stdc++.h>

using namespace std;

struct process {
  int processno;
  int AT;
  int BT;

  int BTbackup;
  int WT;
  int TAT;
  int CT;
};

struct process p[4];

int totaltime = 0;
int prefinaltotal = 0;

bool compare(process p1, process p2)
{
  return p1.AT < p2.AT;
}

int findlargest(int at)
{
  int max = 0, i;
  for (i = 0; i < 4; i++) {
    if (p[i].AT <= at) {
```

```
    if (p[i].BT > p[max].BT)
      max = i;
   }
  }

  return max;
}

int findCT()
{

  int index;
  int flag = 0;
  int i = p[0].AT;
  while (1) {
   if (i <= 4) {
    index = findlargest(i);
   }

   else
    index = findlargest(4);
   cout << "Process executing at time " << totaltime
    << " is: P" << index + 1 << "\t";

   p[index].BT -= 1;
   totaltime += 1;
   i++;

   if (p[index].BT == 0) {
    p[index].CT = totaltime;
    cout << " Process P" << p[index].processno
     << " is completed at " << totaltime;
   }
   cout << endl;

   if (totaltime == prefinaltotal)
```

```
    break;
  }
}

int main()
{

  int i;

  for (i = 0; i < 4; i++) {
    p[i].processno = i + 1;
  }

  for (i = 0; i < 4; i++)
  {
    p[i].AT = i + 1;
  }

  for (i = 0; i < 4; i++) {

    p[i].BT = 2 * (i + 1);
    p[i].BTbackup = p[i].BT;
    prefinaltotal += p[i].BT;
  }

  cout << "PNo\tAT\tBT\n";

  for (i = 0; i < 4; i++) {
    cout << p[i].processno << "\t";
    cout << p[i].AT << "\t";
    cout << p[i].BT << "\t";
    cout << endl;
  }
  cout << endl;

  sort(p, p + 4, compare);
```

```
totaltime += p[0].AT;

prefinaltotal += p[0].AT;
findCT();
int totalWT = 0;
int totalTAT = 0;
for (i = 0; i < 4; i++) {
  p[i].TAT = p[i].CT - p[i].AT;
  p[i].WT = p[i].TAT - p[i].BTbackup;

  totalWT += p[i].WT;

  totalTAT += p[i].TAT;
}

cout << "After execution of all processes ... \n";

cout << "PNo\tAT\tBT\tCT\tTAT\tWT\n";

for (i = 0; i < 4; i++) {
  cout << p[i].processno << "\t";
  cout << p[i].AT << "\t";
  cout << p[i].BTbackup << "\t";
  cout << p[i].CT << "\t";
  cout << p[i].TAT << "\t";
  cout << p[i].WT << "\t";
  cout << endl;
}

cout << endl;
cout << "Total TAT = " << totalTAT << endl;
cout << "Average TAT = " << totalTAT / 4.0 << endl;
cout << "Total WT = " << totalWT << endl;
cout << "Average WT = " << totalWT / 4.0 << endl;
return 0;
```

}

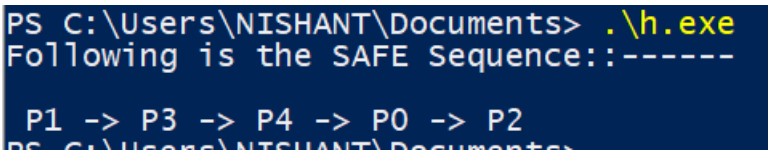
```
PS C:\Users\NISHANT\Documents> .\t.exe
PNo     AT      BT
1       1       2
2       2       4
3       3       6
4       4       8

Process executing at time 1 is: P1
Process executing at time 2 is: P2
Process executing at time 3 is: P3
Process executing at time 4 is: P4
Process executing at time 5 is: P4
Process executing at time 6 is: P4
Process executing at time 7 is: P3
Process executing at time 8 is: P4
Process executing at time 9 is: P3
Process executing at time 10 is: P4
Process executing at time 11 is: P2
Process executing at time 12 is: P3
Process executing at time 13 is: P4
Process executing at time 14 is: P2
Process executing at time 15 is: P3
Process executing at time 16 is: P4
Process executing at time 17 is: P1      Process P1 is completed at 18
Process executing at time 18 is: P2      Process P2 is completed at 19
Process executing at time 19 is: P3      Process P3 is completed at 20
Process executing at time 20 is: P4      Process P4 is completed at 21
After execution of all processes ...
PNo     AT      BT      CT      TAT     WT
1       1       2       18      17      15
2       2       4       19      17      13
3       3       6       20      17      11
4       4       8       21      17      9

Total TAT = 68
Average TAT = 17
Total WT = 48
Average WT = 12
```

Program 7: Write a Program to implement Bankers Algorithm.

```
#include <iostream>
using namespace std;

int main()
{

        int n, m, i, j, k;
        n = 5;
        m = 3;
        int allocation[5][3] = { { 0, 1, 0 },
                                 { 2, 0, 0 },
                                 { 3, 0, 2 },
                                 { 2, 1, 1 },
                                 { 0, 0, 2 } };

        int maximum[5][3] = { { 7, 5, 3 },
                              { 3, 2, 2 },
                              { 9, 0, 2 },
                              { 2, 2, 2 },
                              { 4, 3, 3 } };

        int available[3] = { 3, 3, 2 };

        int f[n], answer[n], idx = 0;
```

```
for (k = 0; k < n; k++) {
        f[k] = 0;
}
int req[n][m];
for (i = 0; i < n; i++) {
        for (j = 0; j < m; j++)
                req[i][j] = maximum[i][j] - allocation[i][j];
}
int y = 0;
for (k = 0; k < 5; k++) {
        for (i = 0; i < n; i++) {
                if (f[i] == 0) {

                        int flag = 0;
                        for (j = 0; j < m; j++) {
                                if (req[i][j] > available[j]){
                                        flag = 1;
                                        break;
                                }
                        }

                        if (flag == 0) {
                                answer[idx++] = i;
                                for (y = 0; y < m; y++)
                                        available[y] += allocation[i][y];
                                f[i] = 1;
                        }
                }
        }
}
```

```
        cout << "\nFollowing is the SAFE Sequence::------" << endl<<endl;
        for (i = 0; i < n - 1; i++)
                cout << " P" << answer[i] << " ->";
        cout << " P" << answer[n - 1] <<endl;

        return (0);
}
```



Problem 8: Write a program to implement Producer- Consumer Problem.

```
#include<bits/stdc++.h>
using namespace std;

int mutex=1,full=0,empty=3,x=0;

int main()
{

int n;

void producer();

void consumer();

int wait(int);

int signal(int);

cout<<"\n1.Producer\n2.Consumer\n3.Exit";
while(1)
{
        cout<<"\nEnter your choice:";
cin>>n;

switch(n)
{
```

```
case 1: if((mutex==1)&&(empty!=0))
producer();
else
cout<<"Buffer is full!!";
break;

case 2: if((mutex==1)&&(full!=0))
consumer();
else
cout<<"Buffer is empty!!";
break;

case 3:
exit(0);
break;

}}

return 0;
}

int wait(int s)
{
        return (--s);
}

int signal(int s)
{
        return(++s);
}
```

```
void producer()
{
        mutex=wait(mutex);
full=signal(full);
empty=wait(empty);
x++;
cout<<"\nProducer produces the item  "<<x;
mutex=signal(mutex);
}
void consumer()
{
        mutex=wait(mutex);
full=wait(full);
empty=signal(empty);
cout<<"\nConsumer consumes item  "<<x;
x--;

mutex=signal(mutex);
}
```

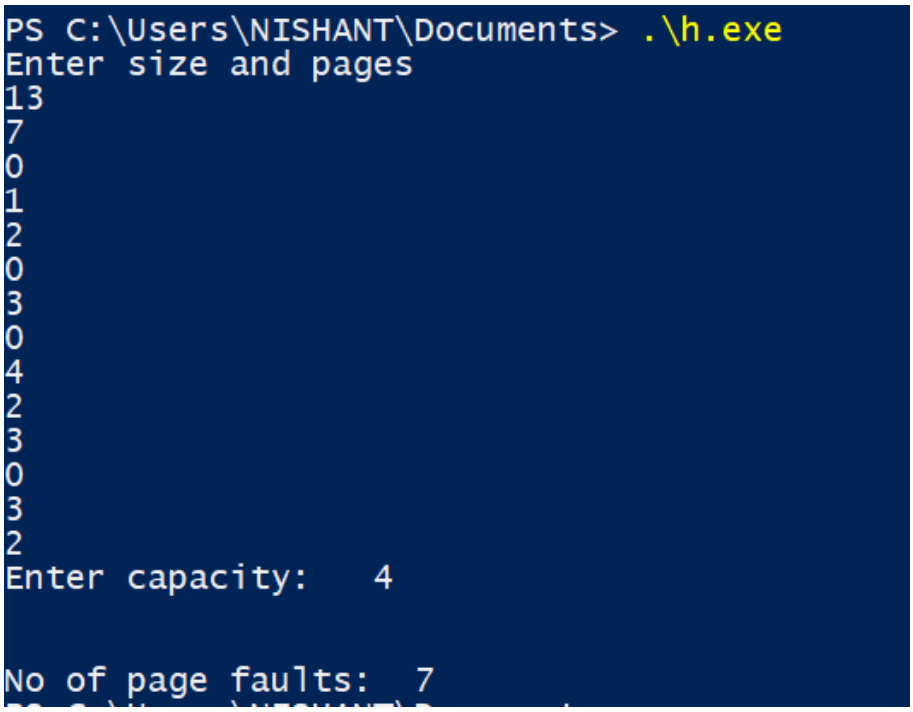
```
PS C:\Users\NISHANT\Documents> .\h.exe

1.Producer
2.Consumer
3.Exit
Enter your choice:1

Producer produces the item  1
Enter your choice:1

Producer produces the item  2
Enter your choice:2

Consumer consumes item  2
Enter your choice:2

Consumer consumes item  1
Enter your choice:2
Buffer is empty!!
Enter your choice:1

Producer produces the item  1
Enter your choice:1

Producer produces the item  2
Enter your choice:1

Producer produces the item  3
Enter your choice:1
Buffer is full!!
Enter your choice:2

Consumer consumes item  3
Enter your choice:3
```

Problem 9: Write a program to implement the First-In-First-Out (FIFO) Page Replacement Algorithm

```
#include<bits/stdc++.h>
using namespace std;

int pageFaults(int pages[], int n, int capacity)
{
	unordered_set<int> s;

	queue<int> indexes;

	int page_faults = 0;
	for (int i=0; i<n; i++)
	{
		if (s.size() < capacity)
		{
			if (s.find(pages[i])==s.end())
			{
				s.insert(pages[i]);

				page_faults++;

				indexes.push(pages[i]);
			}
		}
```

```
            else
            {
                if (s.find(pages[i]) == s.end())
                {
                    int val = indexes.front();

                    indexes.pop();

                    s.erase(val);

                    s.insert(pages[i]);

                    indexes.push(pages[i]);

                    page_faults++;
                }
            }
        }

        return page_faults;
}

int main()
{
        int capacity,n;

    cout<<"Enter size and pages\n";
    cin>>n;
```

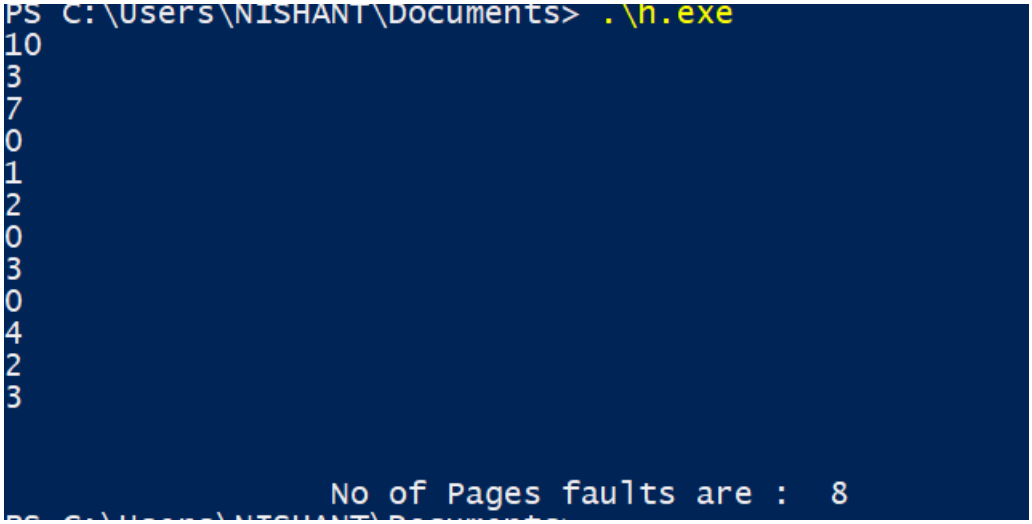
```
    int pages[n];

    for(int i=0;i<n;i++) cin>>pages[i];

    cout<<"Enter capacity:   "; cin>>capacity;


        cout<<"\n\nNo of page faults:  "<<pageFaults(pages, n, capacity);

        return 0;
}
```



Problem 10: Write a program to implement LRU Page Replacement algorithm

```
#include<bits/stdc++.h>
using namespace std;

int main()
{


   int n,capacity;


   cin>>n>>capacity;


   int pages[n];


   for(int i=0;i<n;i++)
   cin>>pages[i];


    int fault = 0;
    set<pair<int,int>>s;
    map<int,int>mp;


    for(int i=0;i<n;i++)
    {
        if(mp.find(pages[i]) == mp.end())
        {
            fault++;
            if(s.size()<capacity)
```

```
            {
                mp[pages[i]] = i;
                s.insert({i,pages[i]});
            }
            else
            {
                mp.erase((*s.begin()).second);
                s.erase(s.begin());
                mp[pages[i]] = i;
                s.insert({i,pages[i]});
            }
        }
        else
        {
            s.erase({mp[pages[i]] , pages[i]});
            mp[pages[i]] = i;
            s.insert({i,pages[i]});
        }
    }

    cout<<"\n\n\t\t No of Pages faults are :  "<<fault<<endl;




    return 0;
}
```



Problem 11: Write a program to implement Dining Philosophers problem.

```
#include<bits/stdc++.h>
using namespace std;

void wait(bool& x)
{
x = false;
}

void signal(bool& x)
{
x = true;
}

int32_t main()
{
```

```
int count=0;

cout<<"Enter Number Of Philospher ";
int n;cin>>n;

bool chopsticks[n];
memset(chopsticks , true,sizeof chopsticks);
bool philospher[n];
memset(philospher , false, sizeof philospher);
while(count<n)
{
for(int i=0;i<n;i++)
{
if(philospher[i]==false && chopsticks[i]&&chopsticks[(i+1)%n])
{
cout<<"philospher "<<i+1<<" is eating\n\n";
philospher[i] = true;
wait(chopsticks[i]);
wait(chopsticks[(i+1)%n]);
count++;
}
else if(philospher[i]==false)
{
cout<<"philospher "<<i+1<<" is thinking\n\n";
}
}
for(int i=0;i<n;i++)
signal(chopsticks[i]);
}
return 0;
```

}

```
PS C:\Users\NISHANT\Documents> .\H.EXE
Enter Number Of Philospher 6
philospher 1 is eating

philospher 2 is thinking

philospher 3 is eating

philospher 4 is thinking

philospher 5 is eating

philospher 6 is thinking

philospher 2 is eating

philospher 4 is eating

philospher 6 is eating
```

Problem 12: Write a program to implement the FCFS (First Come First Serve) Disk Scheduling Algorithms.

```
#include <bits/stdc++.h>
using namespace std;

int size = 8;

void FCFS(int arr[], int head)
{
        int seek_count = 0;
        int distance, cur_track;
```

```
	for (int i = 0; i < size; i++) {
		cur_track = arr[i];

		distance = abs(cur_track - head);

		seek_count += distance;

		head = cur_track;
	}

	cout << "Total number of seek operations = "
		<< seek_count << endl;

	cout << "Seek Sequence is" << endl;

	for (int i = 0; i < size; i++) {
		cout << arr[i] << endl;
	}
}

int main()
{

	int arr[size] = { 176, 79, 34, 60, 92, 11, 41, 114 };
	int head = 50;

	FCFS(arr, head);

	return 0;
```

}

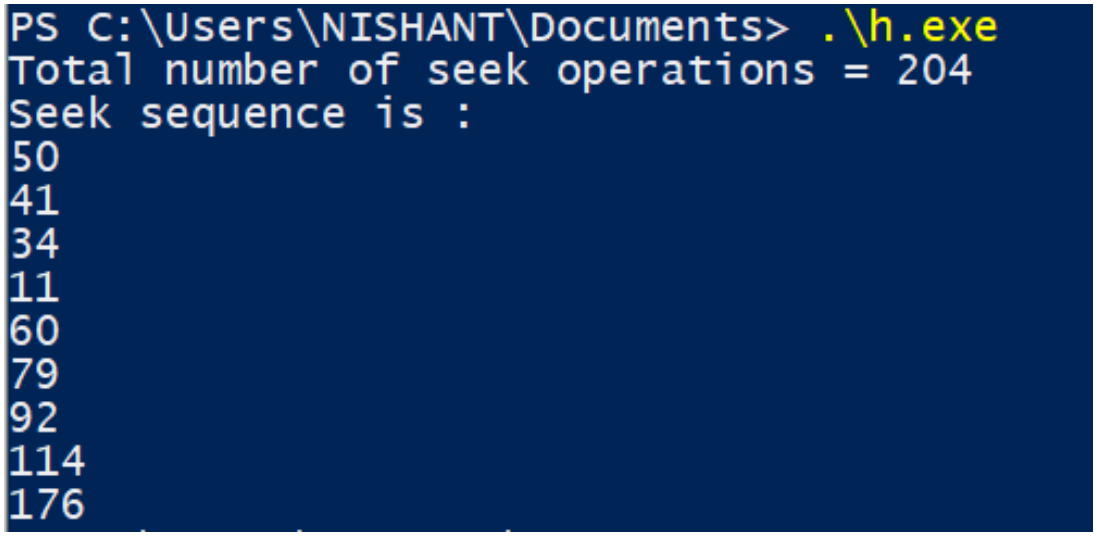
```
PS C:\Users\NISHANT\Documents> .\h.exe
Total number of seek operations = 510
Seek Sequence is
176
79
34
60
92
11
41
114
```

Problem 13: Write a program to implement the SSTF (Shortest Seek Time First) Disk Scheduling Algorithms.

```
#include <bits/stdc++.h>
using namespace std;

void calculatedifference(int request[], int head,
                                                int diff[][2], int n)
{
        for(int i = 0; i < n; i++)
        {
                diff[i][0] = abs(head - request[i]);
        }
}

int findMIN(int diff[][2], int n)
{
        int index = -1;
        int minimum = 1e9;

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

```
        {
                if (!diff[i][1] && minimum > diff[i][0])
                {
                        minimum = diff[i][0];
                        index = i;
                }
        }
        return index;
}

void shortestSeekTimeFirst(int request[],
                                                int head, int n)
{
        if (n == 0)
        {
                return;
        }

        int diff[n][2] = { { 0, 0 } };

        int seekcount = 0;

        int seeksequence[n + 1] = {0};

        for(int i = 0; i < n; i++)
        {
                seeksequence[i] = head;
                calculatedifference(request, head, diff, n);
                int index = findMIN(diff, n);
                diff[index][1] = 1;
```

```
            seekcount += diff[index][0];

            head = request[index];
        }
        seeksequence[n] = head;

        cout << "Total number of seek operations = "
            << seekcount << endl;
        cout << "Seek sequence is : " << "\n";

        for(int i = 0; i <= n; i++)
        {
            cout << seeksequence[i] << "\n";
        }
}

int main()
{
        int n = 8;
        int proc[n] = { 176, 79, 34, 60, 92, 11, 41, 114 };

        shortestSeekTimeFirst(proc, 50, n);

        return 0;
}
```


PS C:\Users\NISHANT\Documents> .\h.exe
Total number of seek operations = 204
Seek sequence is :
50
41
34
11
60
79
92
114
176