**Department of Information Technology**

**Delhi Technological University**

**Operating System Lab (IT-204)**

****

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

Submitted by: Submitted to:

Varun Kumar Mr. Jasraj Meena

Roll no: 2K19/IT/140 Assistant Professor

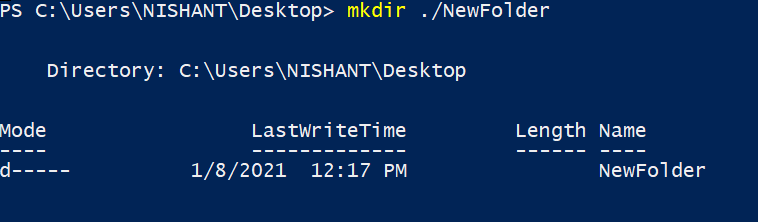
Group: G3 Dept. of IT, DTU

**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



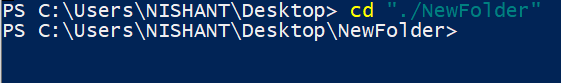
### Move a file



### Copy a file to another directory



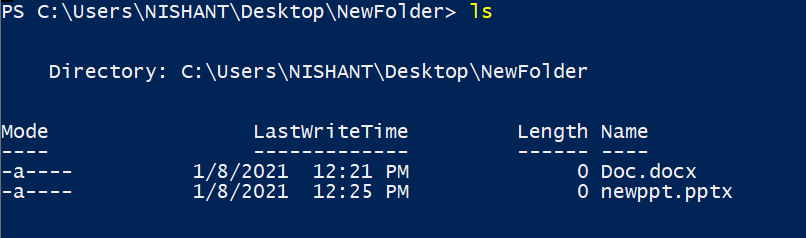
### Change Directory



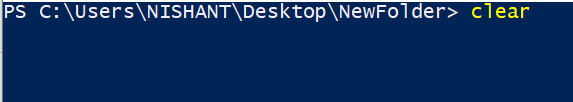
### Renaming a file



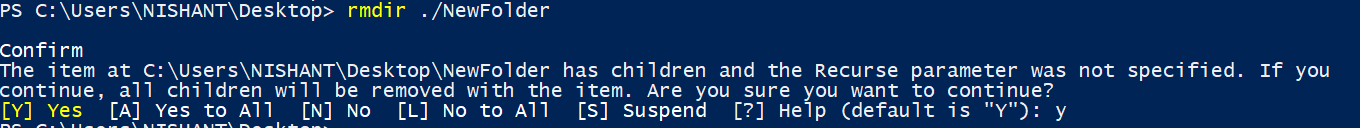
### Listing Directory



### Clear the Terminal screen of all previous commands



### Remove an empty directory



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;

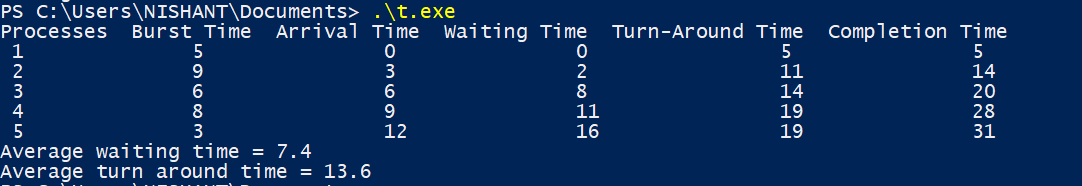
int burst\_time[] = {5, 9, 6, 8, 3};

int arrival\_time[] = {0, 3, 6, 9, 12};

findavgTime(processes, n, burst\_time, arrival\_time);

return 0;

}



# 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";

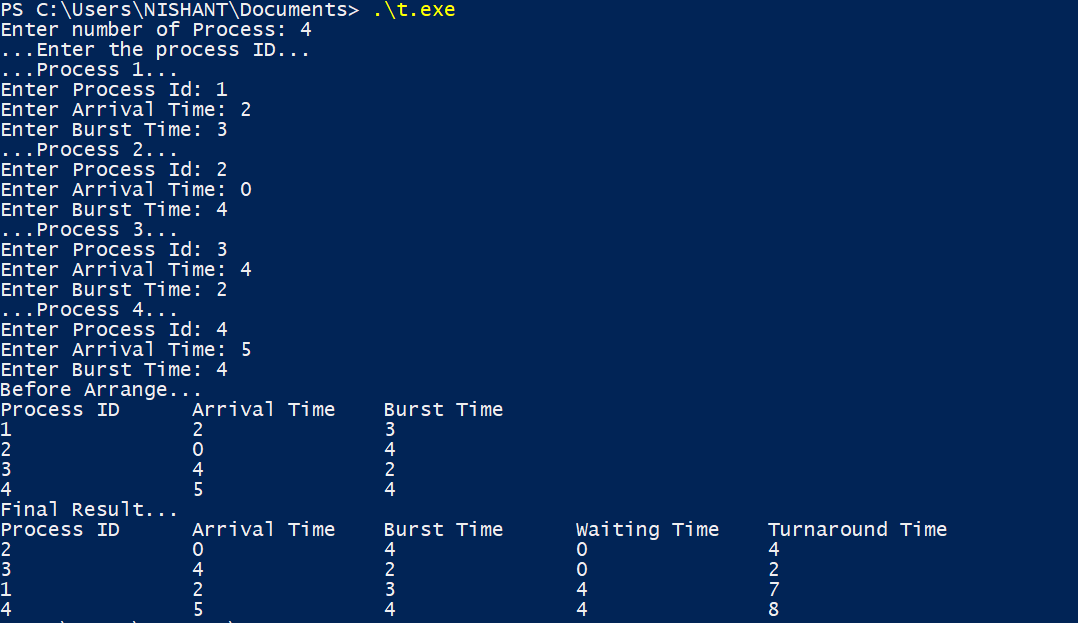
for(int i=0; i<num; i++)

{

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";

}

}



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()

{

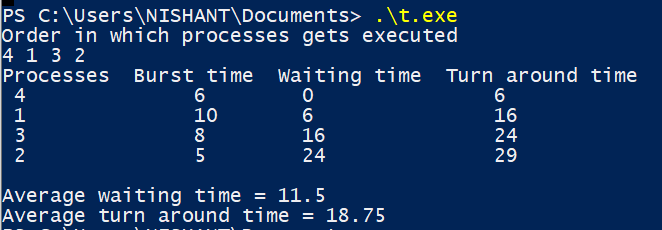
Process proc[] = {{1, 10, 2}, {2, 5, 0}, {3, 8, 1} ,{4 , 6, 3}};

int n = 4;

priorityScheduling(proc, n);

return 0;

}



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;

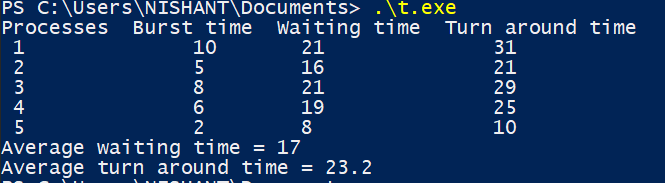
int burst\_time[] = {10, 5, 8, 6, 2};

int quantum = 2;

findavgTime(processes, n, burst\_time, quantum);

return 0;

}

****

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;

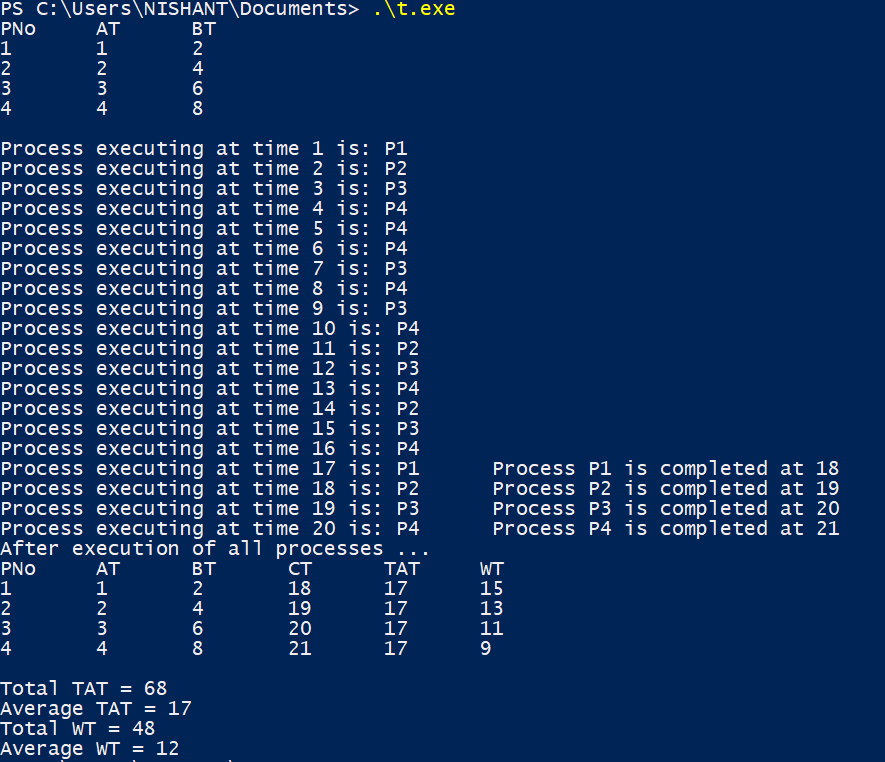
cout << "Average TAT = " << totalTAT / 4.0 << endl;

cout << "Total WT = " << totalWT << endl;

cout << "Average WT = " << totalWT / 4.0 << endl;

return 0;

}



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;

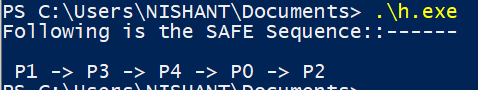
for (i = 0; i < n - 1; i++)

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);

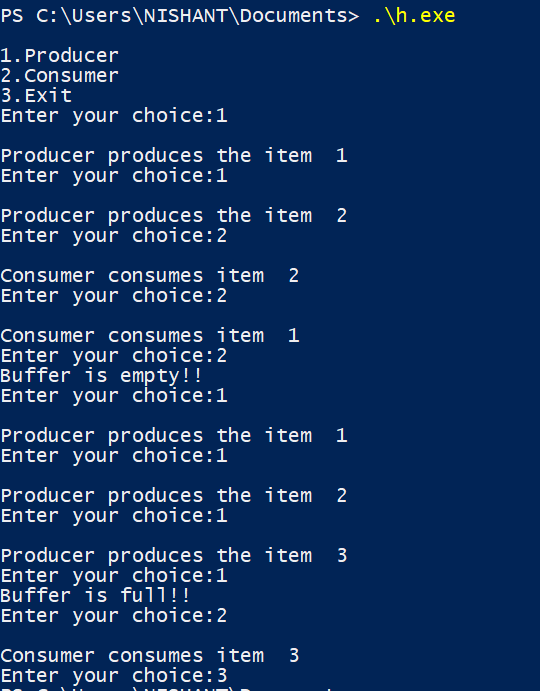
empty=signal(empty);

cout<<"\nConsumer consumes item "<<x;

x--;

mutex=signal(mutex);

}

****

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];

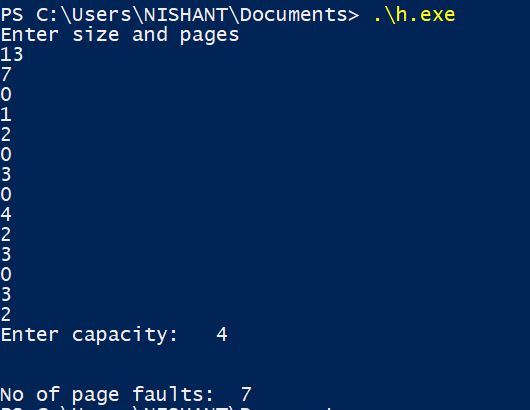
for(int i=0;i<n;i++) cin>>pages[i];

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]});

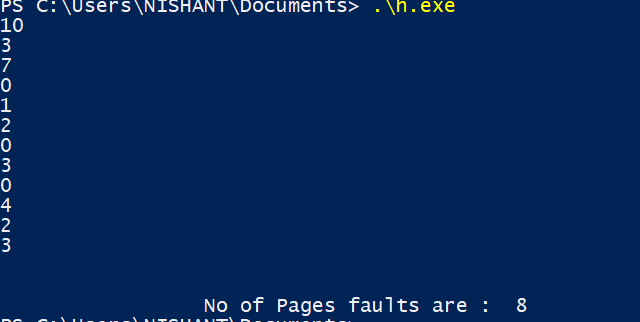
}

}

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";

}

}

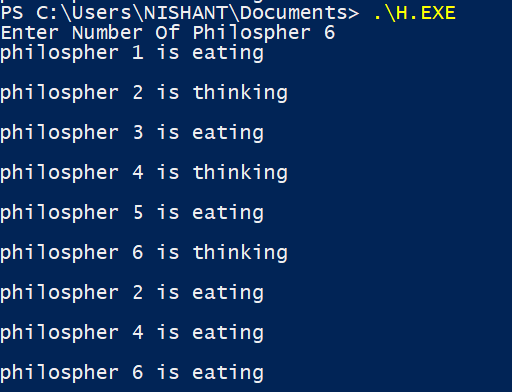
for(int i=0;i<n;i++)

signal(chopsticks[i]);

}

return 0;

}



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()

{

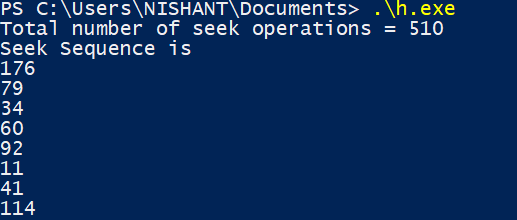
int arr[size] = { 176, 79, 34, 60, 92, 11, 41, 114 };

int head = 50;

FCFS(arr, head);

return 0;

}



# 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;

}

