**FCFS CPU SCHEDULING**

#include <bits/stdc++.h>

using namespace std;

#define X (1000000007)

int main()

{ int n;//Processes input

cout<<"Input No. of processes : ";

cin>>n;

vector<pair<int,int> > v(n);

cout<<"\nEnter Arrival time and Burst time for all .\n";

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

cin>>v[i].first>>v[i].second;

}

sort(v.begin(),v.end());

queue<pair<int,int> > q;

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

q.push(v[i]);

}

int time=0;

int waiting\_time =0;

int turnover=0;

int i=0;

while(!q.empty()){

i++;

int a=waiting\_time,b=turnover;

int x=q.front().first,y=q.front().second;

q.pop();

if(time<=x){

waiting\_time+=0;

turnover+=y;

time=x+y;

}

else{

waiting\_time+=(time-x);

turnover+=y+(time-x);

time+=y;

}

cout<<endl<<"P"<<i<<" WaitingTime|Turnaround Time == "<<waiting\_time-a<<"|"<<turnover-b;

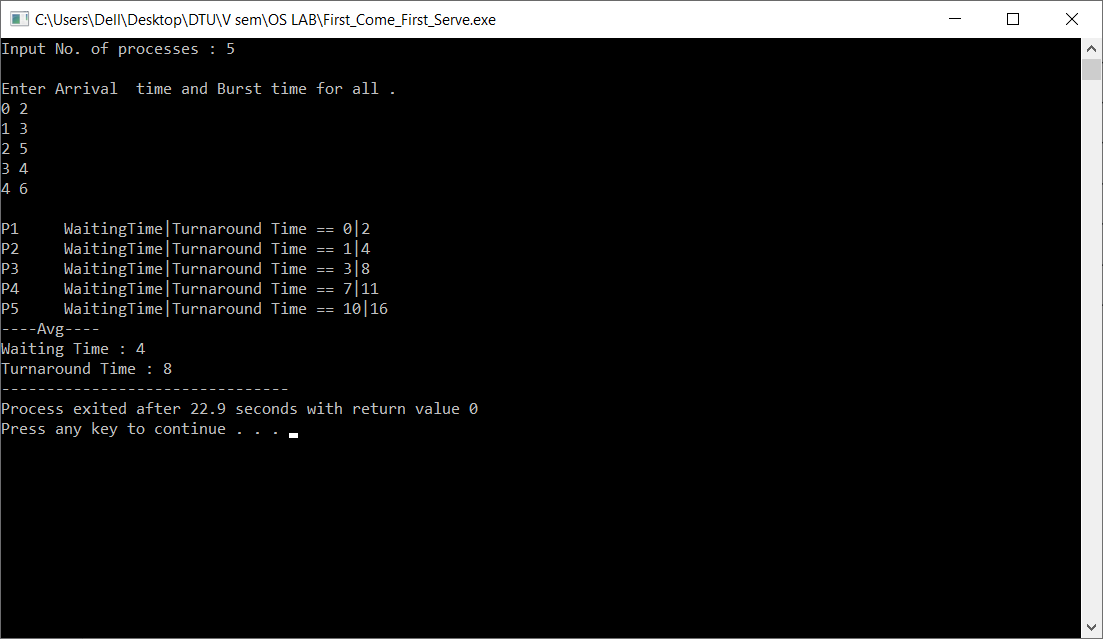
}

cout<<endl<<"----Avg----\nWaiting Time : "<<waiting\_time/n<<endl;

cout<<"Turnaround Time : "<<turnover/n;

return 0;

}

****

**Round-Robin CPU SCHEDULING**

#include <bits/stdc++.h>

using namespace std;

int main(){

int i,n,time,remain,temps=0,time\_quantum;

int wt=0,tat=0;

cout<<"Enter the total number of process="<<endl;

cin>>n;

remain=n;

vector<int>at(n);

vector<int>bt(n);

vector<int>rt(n);

cout<<"Enter the Arrival time, Burst time for All the processes"<<endl;

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

{

cin>>at[i];

cin>>bt[i];

rt[i]=bt[i];

}

cout<<"Enter the value of time QUANTUM:"<<endl;

cin>>time\_quantum;

cout<<"\n\nProcess\t:Turnaround Time:Waiting Time\n\n";

for(time=0,i=0;remain!=0;)

{

if(rt[i]<=time\_quantum && rt[i]>0)

{

time += rt[i];

//Addition using shorthand operators

rt[i]=0;

temps=1;

}

else if(rt[i]>0)

{

rt[i] -= time\_quantum;

//Subtraction using shorthand operators

time += time\_quantum;

//Addition using shorthand operators

}

if(rt[i]==0 && temps==1)

{

remain--;

//Desplaying the result of wating, turn around time:

printf("Process{%d}\t:\t%d\t:\t%d\n",i+1,time-at[i],time-at[i]-bt[i]);

cout<<endl;

wt += time-at[i]-bt[i];

tat += time-at[i];

temps=0;

}

if(i == n-1)

i=0;

else if(at[i+1] <= time)

i++;

else

i=0;

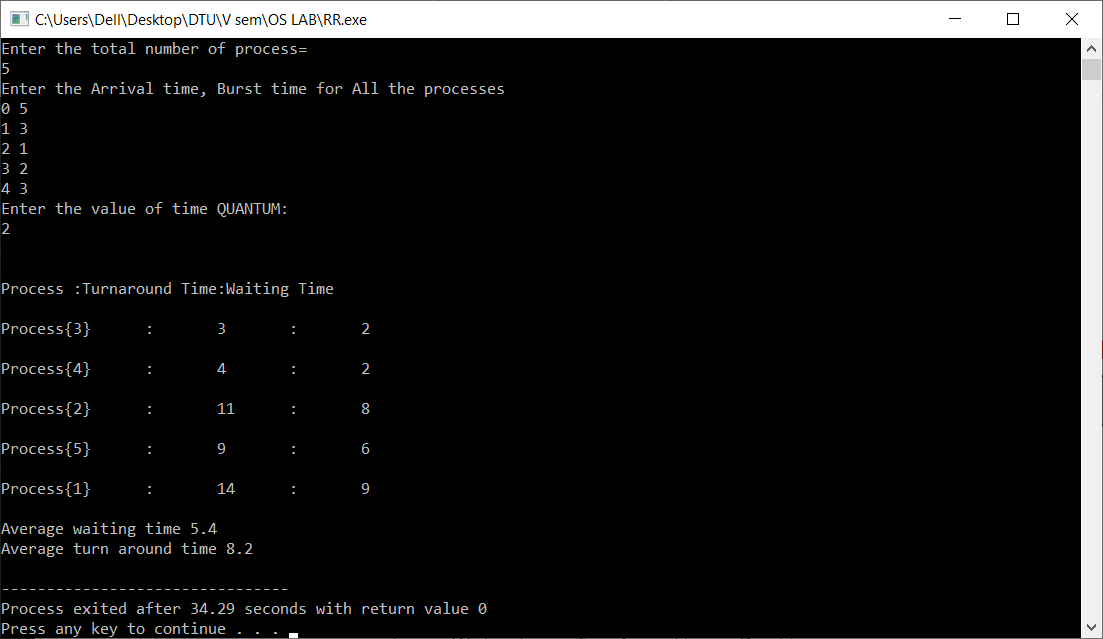
}

cout<<"Average waiting time "<<wt\*1.0/n<<endl;

cout<<"Average turn around time "<<tat\*1.0/n<<endl;;

return 0;

}

****

**SJF CPU SCHEDULING**

#include <bits/stdc++.h>

using namespace std;

#define X (1000000007)

class sjf{

public:

int at,bt,ct,cbt;

};

bool comp(sjf const &a,sjf const &b){

return a.at<b.at;

}

int main()

{

int n;//Processes input

cout<<"Input number of processes : ";

cin>>n;

vector<sjf> v(n);

cout<<endl<<"Enter AT & BT for All :\n";

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

cin>>v[i].at>>v[i].bt;

v[i].ct=0;

v[i].cbt=v[i].bt;

}

int time=0;

while(1){

int index=-1,st=INT\_MAX;

bool done=1;

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

if(v[i].bt!=0)done=0;

if(v[i].at<=time&&v[i].bt<st&&v[i].bt!=0){

st=v[i].bt;

index=i;

}

}

if(done){

break;

}

if(index==-1){

time++;

continue;

}

v[index].bt--;

v[index].ct=time+1;

time++;

}

cout<<"-------------------------------\n";

double avgwt=0,avgtt=0;

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

cout<<"Waiting Time for process : P"<<i+1<<" is "<<v[i].ct-v[i].at-v[i].cbt<<endl;

avgwt+=v[i].ct-v[i].at-v[i].cbt;

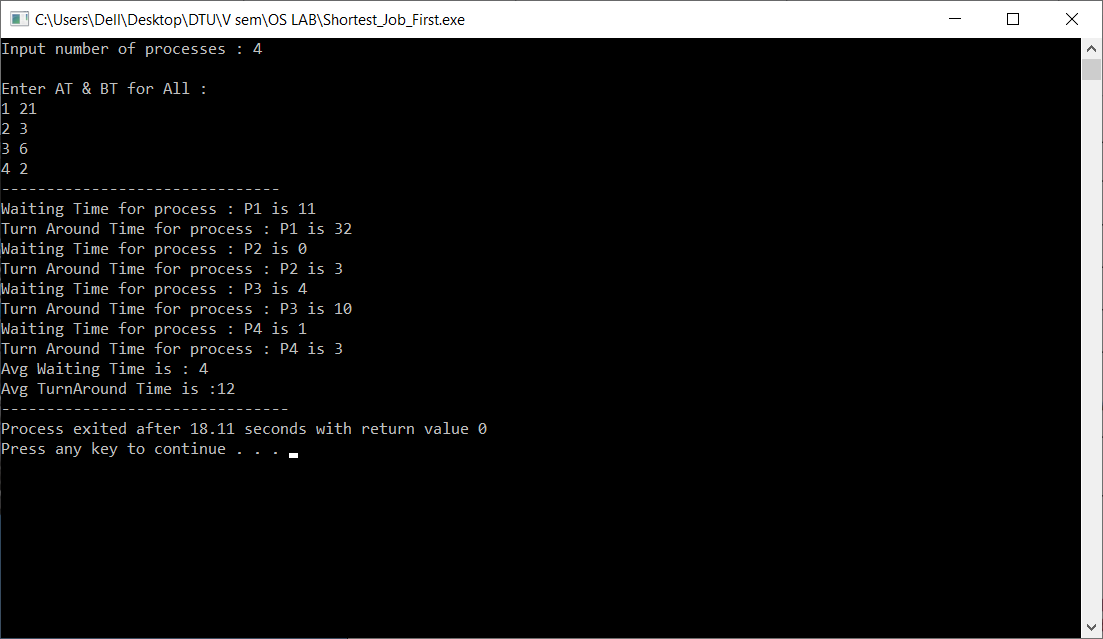
cout<<"Turn Around Time for process : P"<<i+1<<" is "<<v[i].ct-v[i].at<<endl;

avgtt+=v[i].ct-v[i].at;

}

cout<<"Avg Waiting Time is : "<<avgwt/n<<endl<<"Avg TurnAround Time is :"<<avgtt/n;

}

****

**PRIORITY CPU SCHEDULING**

#include<iostream>

using namespace std;

int main() {

int bt[20],p[20],wt[20],tat[20],pr[20],i,j,n,total=0,pos,temp,avg\_wt,avg\_tat;

cout<<"Enter Total Number of Process:";

cin>>n;

cout<<"\nEnter Burst Time and Priority\n";

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

cout<<"\nP["<<i+1<<"]\n";

cout<<"Burst Time:";

cin>>bt[i];

cout<<"Priority:";

cin>>pr[i];

p[i]=i+1;

//contains process number

}

//sorting burst time, priority and process number in ascending order using selection sort

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

pos=i;

for (j=i+1;j<n;j++) {

if(pr[j]<pr[pos])

pos=j;

}

temp=pr[i];

pr[i]=pr[pos];

pr[pos]=temp;

temp=bt[i];

bt[i]=bt[pos];

bt[pos]=temp;

temp=p[i];

p[i]=p[pos];

p[pos]=temp;

}

wt[0]=0;

//waiting time for first process is zero

//calculate waiting time

for (i=1;i<n;i++) {

wt[i]=0;

for (j=0;j<i;j++)

wt[i]+=bt[j];

total+=wt[i];

}

avg\_wt=total/n;

//average waiting time

total=0;

cout<<"\nProcesst Burst Time \tWaiting Time\tTurnaround Time";

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

tat[i]=bt[i]+wt[i];

//calculate turnaround time

total+=tat[i];

cout<<"\nP["<<p[i]<<"]\t\t "<<bt[i]<<"\t\t "<<wt[i]<<"\t\t\t"<<tat[i];

}

avg\_tat=total/n;

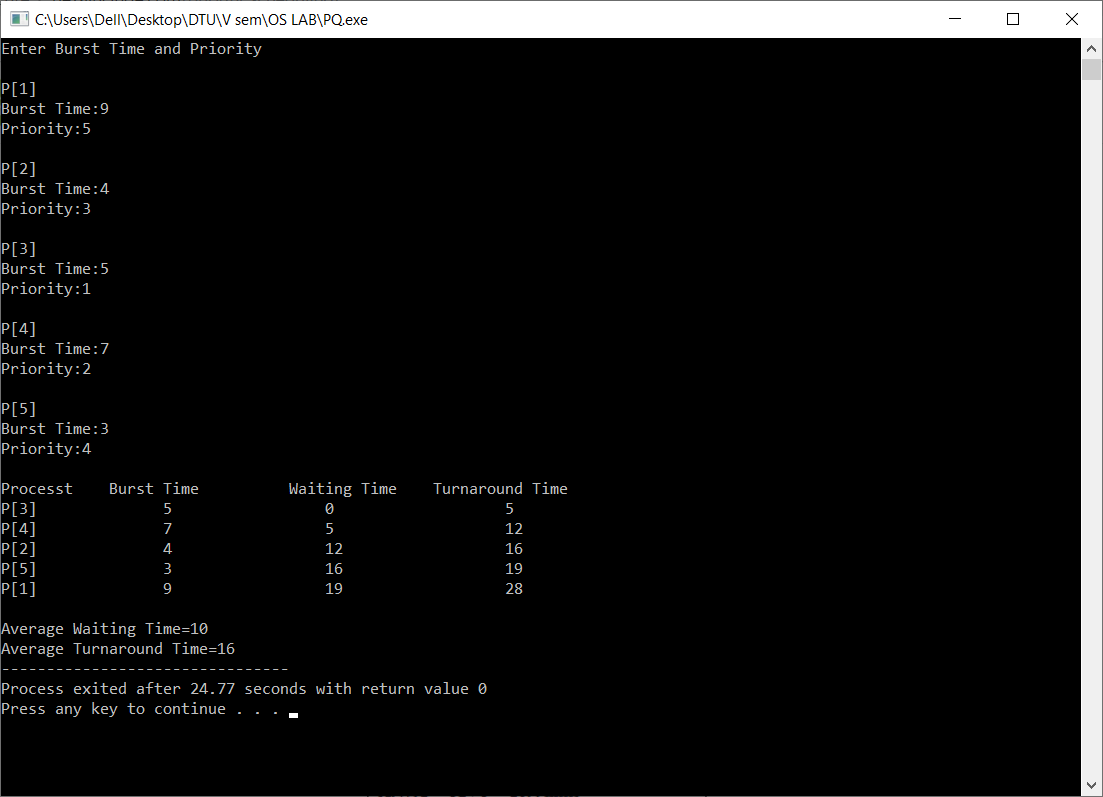
//average turnaround time

cout<<"\n\nAverage Waiting Time="<<avg\_wt;

cout<<"\nAverage Turnaround Time="<<avg\_tat;

return 0;

}

****

**Bankers Algorithm**

#include <stdio.h>

int current[5][5], maximum\_claim[5][5], available[5];

int allocation[5] = {0, 0, 0, 0, 0};

int maxres[5], running[5], safe = 0;

int counter = 0, i, j, exec, resources, processes, k = 1;

int main()

{

printf("\nEnter number of processes: ");

scanf("%d", &processes);

for (i = 0; i < processes; i++)

{

running[i] = 1;

counter++;

}

printf("\nEnter number of resources: ");

scanf("%d", &resources);

printf("\nEnter Claim Vector:");

for (i = 0; i < resources; i++)

{

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

}

printf("\nEnter Allocated Resource Table:\n");

for (i = 0; i < processes; i++)

{

for(j = 0; j < resources; j++)

{

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

}

}

printf("\nEnter Maximum Claim Table:\n");

for (i = 0; i < processes; i++)

{

for(j = 0; j < resources; j++)

{

scanf("%d", &maximum\_claim[i][j]);

}

}

printf("\nThe Claim Vector is: ");

for (i = 0; i < resources; i++)

{

printf("\t%d", maxres[i]);

}

printf("\nThe Allocated Resource Table:\n");

for (i = 0; i < processes; i++)

{

for (j = 0; j < resources; j++)

{

printf("\t%d", current[i][j]);

}

printf("\n");

}

printf("\nThe Maximum Claim Table:\n");

for (i = 0; i < processes; i++)

{

for (j = 0; j < resources; j++)

{

printf("\t%d", maximum\_claim[i][j]);

}

printf("\n");

}

for (i = 0; i < processes; i++)

{

for (j = 0; j < resources; j++)

{

allocation[j] += current[i][j];

}

}

printf("\nAllocated resources:");

for (i = 0; i < resources; i++)

{

printf("\t%d", allocation[i]);

}

for (i = 0; i < resources; i++)

available[i] = maxres[i] - allocation[i];

printf("\nAvailable resources:");

for (i = 0; i < resources; i++)

{

printf("\t%d", available[i]);

}

printf("\n");

while (counter != 0)

{

safe = 0;

for (i = 0; i < processes; i++)

{

if (running[i])

{

exec = 1;

for (j = 0; j < resources; j++)

{

if (maximum\_claim[i][j] - current[i][j] > available[j])

{

exec = 0;

break;

}

}

if (exec)

{

printf("\nProcess%d is executing\n", i + 1);

running[i] = 0;

counter--;

safe = 1;

for (j = 0; j < resources; j++)

{

available[j] += current[i][j];

}

break;

}

}

}

if (!safe)

{

printf("\nThe processes are in unsafe state.\n");

break;

}

else

{

printf("\nThe process is in safe state");

printf("\nAvailable vector:");

for (i = 0; i < resources; i++)

{

printf("\t%d", available[i]);

}

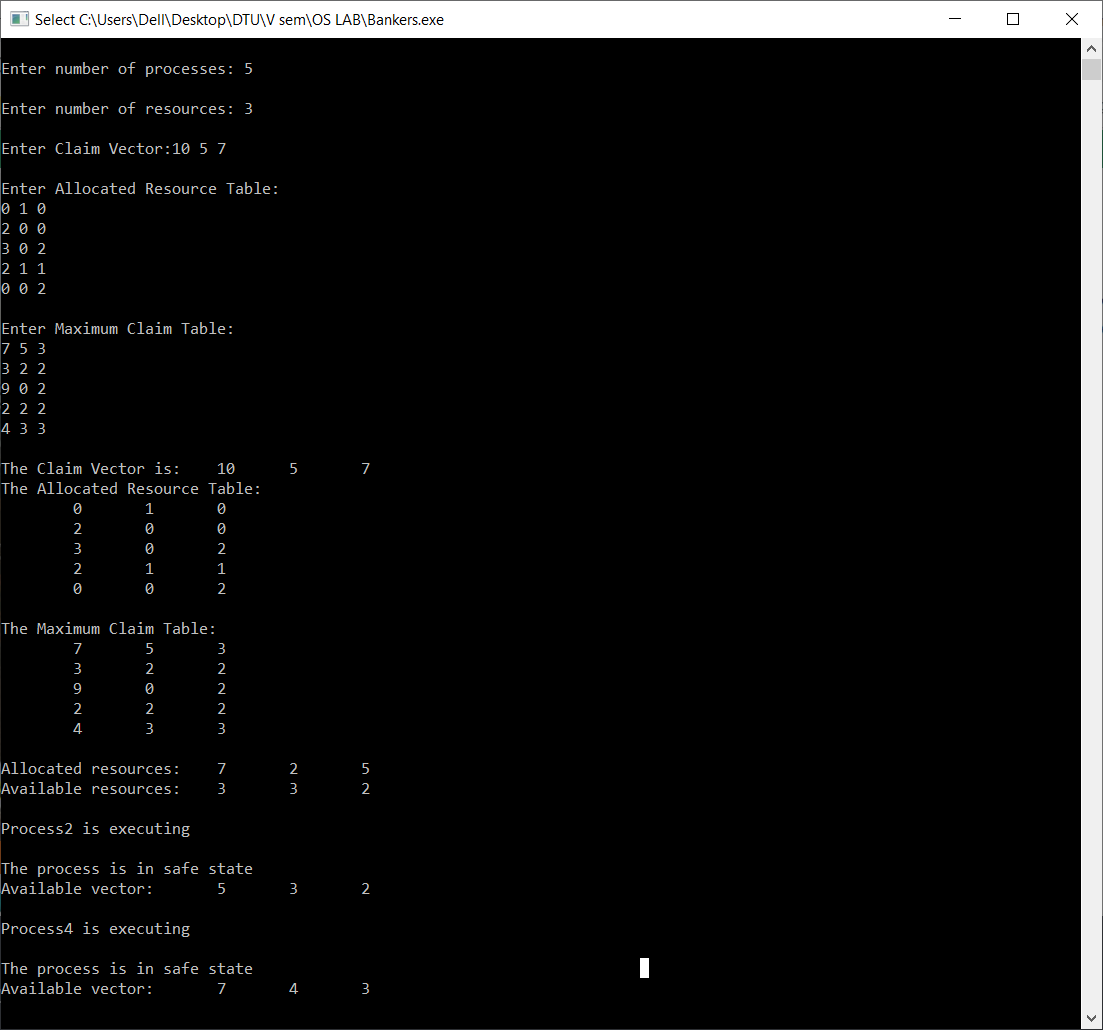
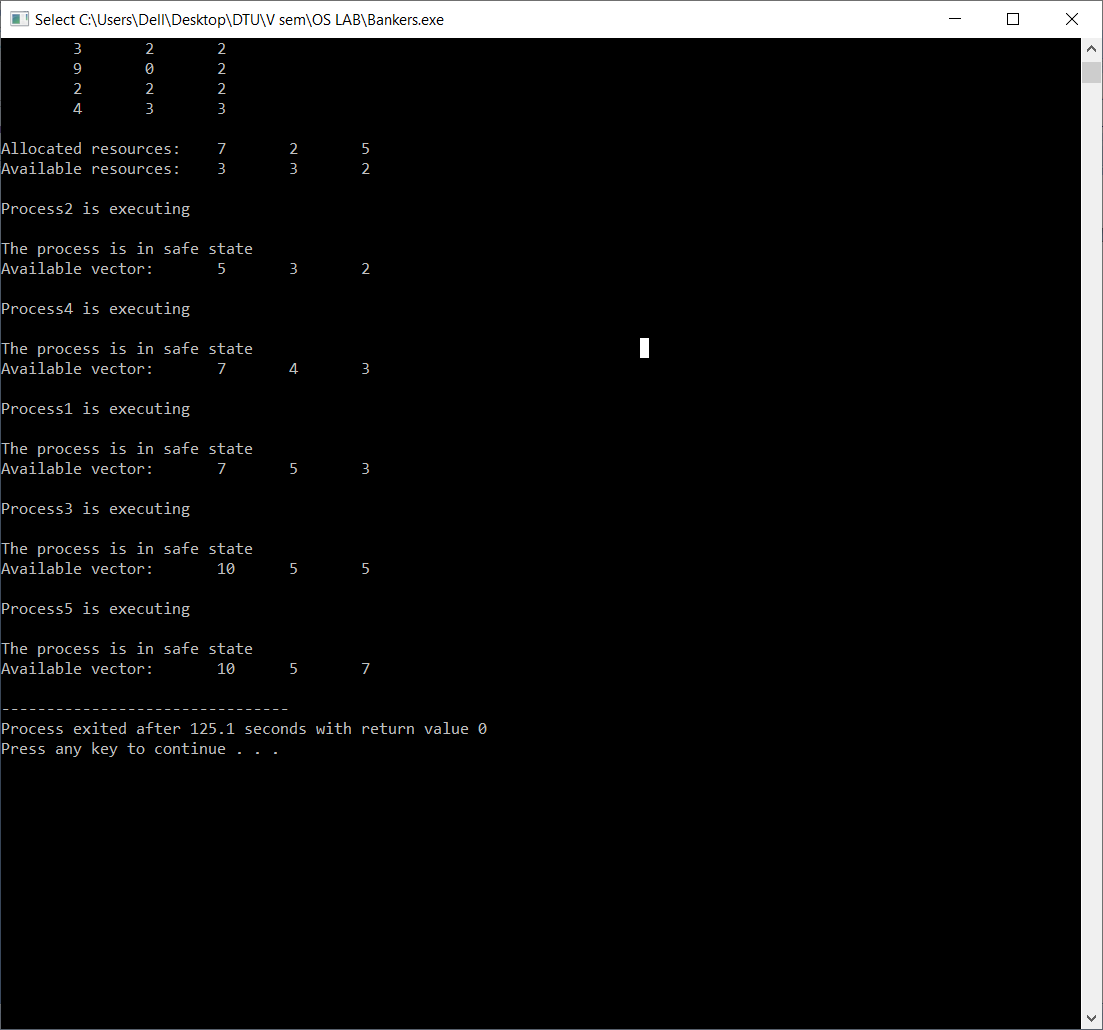
printf("\n");

}

}

return 0;

}

** **

**FIFO Memory Management**

#include<bits/stdc++.h>

#include<set>

using namespace std;

int pageFaults(vector<int> pages, int capacity) {

set<int> s;

queue<int> indexes;

int n =pages.size();

double 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/n)\*100;

}

int main(){

int n,c;

cout<<"Input Capacity : "; cin>>c;

cout<<"\nEnter number of pages : "; cin>>n;

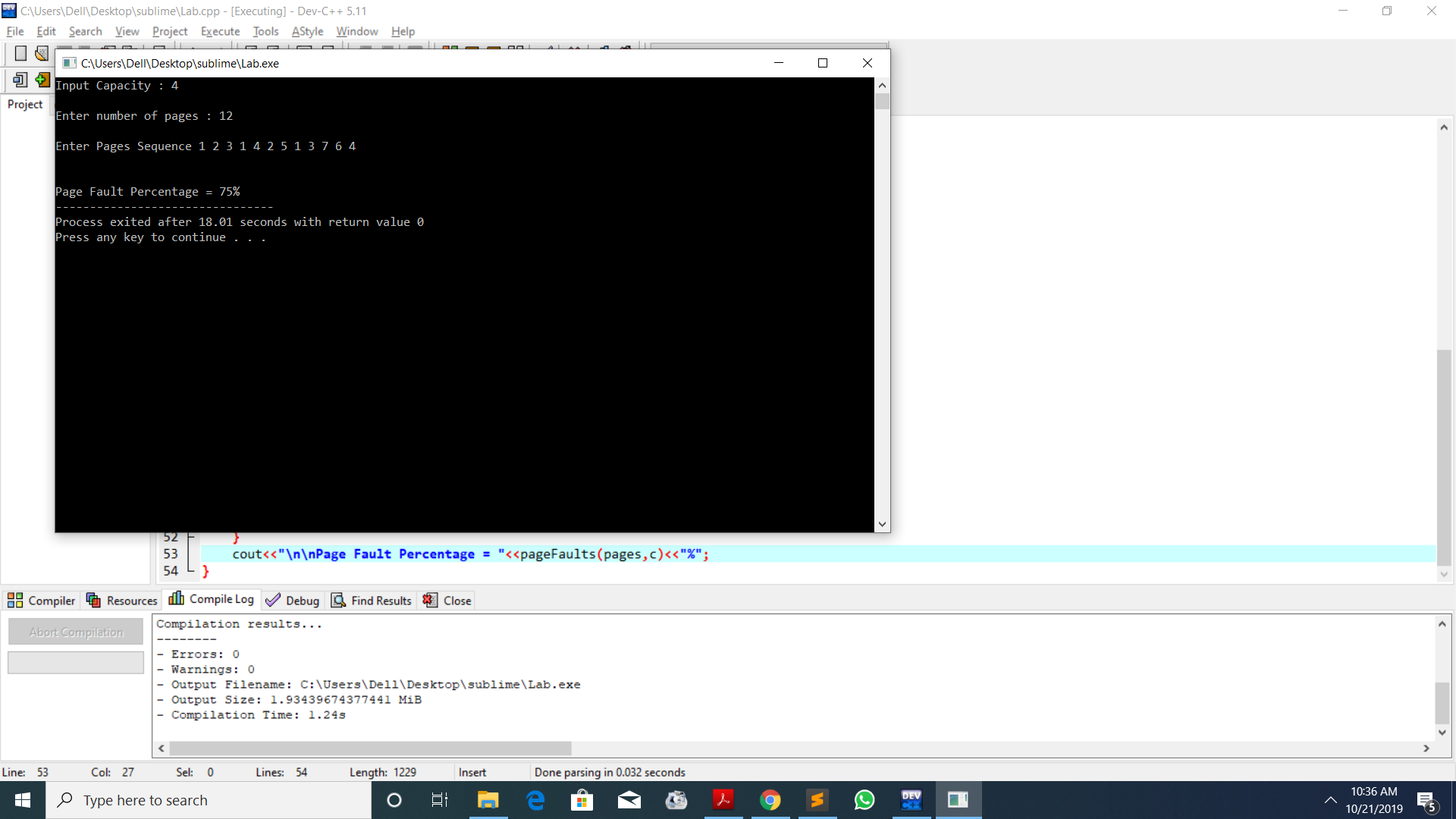
cout<<"\nEnter Pages Sequence "; vector<int> pages(n);

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

cin>>pages[i];

cout<<"\n\nPage Fault Percentage = "<<pageFaults(pages,c)<<"%";

}

****

**LRU Memory Management**

#include<bits/stdc++.h>

using namespace std;

int pageFaults(vector<int> &pages, int capacity){

set<int> s; map<int, int> indexes;

int n=pages.size();

double 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[pages[i]] = i;

}

else

{

if (s.find(pages[i]) == s.end())

{

int lru = INT\_MAX, val; set<int>::iterator it;

for (it=s.begin(); it!=s.end(); it++)

{

if (indexes[\*it] < lru)

{

lru = indexes[\*it];

val = \*it;

}

}

s.erase(val);

s.insert(pages[i]);

page\_faults++;

}

indexes[pages[i]] = i;

}

}

return (page\_faults/n)\*100;

}

int main(){

int n,c;

cout<<"Input Capacity : "; cin>>c;

cout<<"\nEnter number of pages : "; cin>>n;

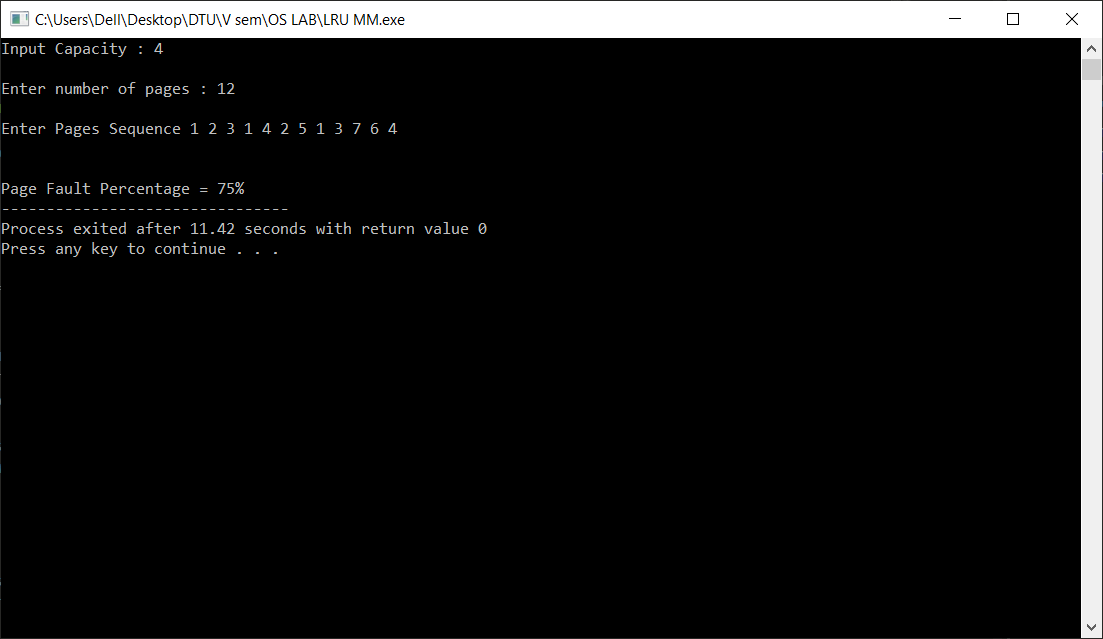
cout<<"\nEnter Pages Sequence "; vector<int> pages(n);

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

cin>>pages[i];

cout<<"\n\nPage Fault Percentage = "<<pageFaults(pages,c)<<"%";

}

****

**SSTF DISK SCHEDULING**

#include<bits/stdc++.h>

using namespace std;

int find\_short(int ref[],int n,int num)

{

int min=99999,ind,temp;

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

{

if(ref[i]!=-1)

{

temp=abs(num-ref[i]);

if(min>temp)

{

min=temp;

ind=i;

}

}

}

return ind;

}

void total\_move(int ref[],int pos,int n)

{

int num=pos,move=0,ind;

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

{

ind=find\_short(ref,n,num);

move+=abs(num-ref[ind]);

num=ref[ind];

ref[ind]=-1;

}

cout<<"Total head movements : "<<move;

}

int main()

{

int ref[1000];

int pos,n;

cout<<"Enter the current position of head ";

cin>>pos;

cout<<"Enter the Size of queue : ";

cin>>n;

cout<<"Enter the request for tracks : ";

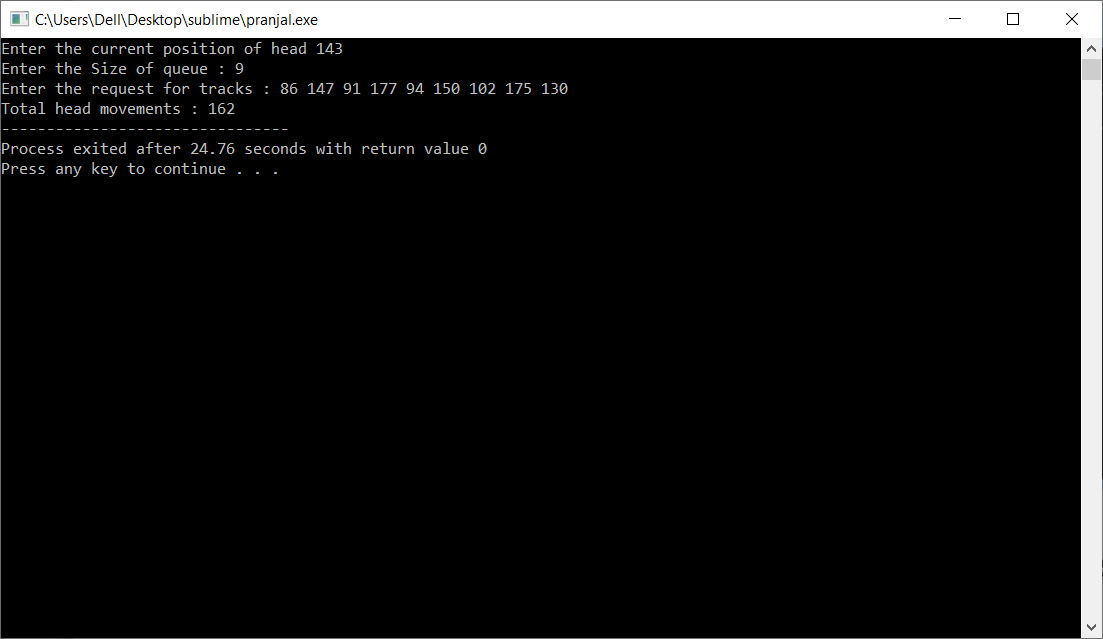
for(int i=0;i<n;i++)

cin>>ref[i];

total\_move(ref,pos,n);

return 0;

}

****