#include<bits/stdc++.h>

using namespace std;

float fcfs\_wt[10]={0}, fcfs\_tat[10]={0},fcfs\_awt=0,fcfs\_atat=0;

float rr\_wt[10]={0}, rr\_tat[10]={0}, rr\_awt=0, rr\_atat=0;

float sjf\_wt[10]={0}, sjf\_tat[10]={0}, sjf\_awt=0, sjf\_atat=0;

float srtf\_wt[10]={0}, srtf\_tat[10]={0}, srtf\_awt=0, srtf\_atat=0;

struct process{

int at;

int id;

};

struct Comparetime {

bool operator()(process const& p1, process const& p2)

{

if(p1.at != p2.at){

return p1.at > p2.at;}

else{

return p1.id>p2.id;

}

}

};

void fcfs\_compute(process p[], int bt[]){

int ct=0;

for(int i=0;i<10;i++){

if(p[i].at<=ct){

ct= ct + bt[p[i].id];

fcfs\_tat[p[i].id] = ct - p[i].at;

fcfs\_wt[p[i].id] = fcfs\_tat[p[i].id] - bt[p[i].id];

}

}

}

void sjf\_compute(process p[], int bt[]){

int left = 10, ct=0;

int vis[10]={0};

for(;left>0;){

int min=10000;

int flag;

for(int i=0;i<10;i++){

if(p[i].at <=ct && bt[p[i].id]<min && vis[p[i].id]==0){

min = bt[p[i].id];

flag=i;

}

}

ct+=min;

sjf\_tat[p[flag].id]= ct - p[flag].at;

sjf\_wt[p[flag].id]=sjf\_tat[p[flag].id] - bt[p[flag].id];

vis[p[flag].id] = 1;

left--;

}

}

void srtf\_compute(process p[], int bt[]){

int bt\_temp[10];

for(int i=0;i<10;i++){

bt\_temp[i]=bt[i];

}

int n=10;

int left =10;

int flag;

int MAX\_INT=1000000, ct=0;

for(;left>0;){

int temp = MAX\_INT;

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

if(bt\_temp[p[i].id]>0 && ct>=p[i].at && bt\_temp[p[i].id]<temp )

{

flag = i;

temp=bt\_temp[p[i].id];

}

}

ct++;

bt\_temp[p[flag].id]-=1;

if(bt\_temp[p[flag].id]==0){

left=left-1;

srtf\_tat[p[flag].id]= ct - p[flag].at;

srtf\_wt[p[flag].id] = srtf\_tat[p[flag].id] - bt[p[flag].id];

}

}

}

void rr\_compute(process p[], int bt[], int tq){

int n=10, quantum\_time;

int ariv[n], burst[n], burst\_temp[n], cpu\_ft[n]={0}, coml[n];

int vis[n]={0};

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

ariv[p[i].id]=p[i].at;

}

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

burst[i]= bt[i];

burst\_temp[i]=burst[i];

}

quantum\_time=tq;

queue <int>q;

q.push(0);

int l=1, left=n, ct=0;

while(left!=0){

int temp = q.front();

q.pop();

if(ariv[temp]<=ct){

if(vis[temp]==0){cpu\_ft[temp] = ct;vis[temp]=1;}

if(burst\_temp[temp]<=quantum\_time){

ct+= burst\_temp[temp];

coml[temp]=ct;

left -=1;

while(l<n){

if(ct>=ariv[l]){

q.push(l);

l++;

}else{break;}

}

}else{

ct+=quantum\_time;

burst\_temp[temp]-=quantum\_time;

while(l<n){

if(ct>=ariv[l]){

q.push(l);

l++;

}else{break;}

}

q.push(temp);

}

}else{

ct++;

}

}

for(int i=0;i<10;i++){

rr\_tat[i]=coml[i] - ariv[i];

rr\_wt[i]=coml[i] - ariv[i]- burst[i];

}

}

int main(){

process p[10];

int bt[]={10,12,8,11,3,15,16,12,7,5},tq;

int at[]={0,2,3,1,4,3,2,6,8,9};

priority\_queue<process, vector<process>, Comparetime> Q;

for(int i=0;i<10;i++){

p[i].at= at[i];

p[i].id=i;

}

// for(int i=0;i<10;i++){

// cin>>bt[i];

// }

// cin>>tq;

tq=2;

for (int i = 0; i <10; ++i) {

Q.push(p[i]);

}

for(int i=0;i<10;i++){

p[i]=Q.top();

Q.pop();

}

fcfs\_compute(p,bt);

sjf\_compute(p,bt);

srtf\_compute(p,bt);

rr\_compute(p,bt, tq);

cout<<"Process ID:"<<" FCFS\_WT"<<" FCFS\_TAT"<<" SJF\_WT"<<" SJF\_TAT"<<" RR\_WT"<<" RR\_TAT"<<" SRTF\_WT"<<" SRTF\_TAT\n";

for(int i=0;i<10;i++){

cout<<i+1<<"\t\t"<<fcfs\_wt[i]<<"\t"<<fcfs\_tat[i]<<"\t"<<sjf\_wt[i]<<"\t"<<sjf\_tat[i]<<"\t"<<rr\_wt[i]<<"\t"<<rr\_tat[i]<<"\t"<<srtf\_wt[i]<<"\t"<<srtf\_tat[i]<<"\n";

}

float total\_wtime[4]={0}, total\_atime[4]={0};

for(int i=0;i<10;i++){

total\_wtime[0]+=fcfs\_wt[i];

total\_atime[0]+=fcfs\_tat[i];

total\_wtime[1]+=sjf\_wt[i];

total\_atime[1]+=sjf\_tat[i];

total\_wtime[2]+=srtf\_wt[i];

total\_atime[2]+=srtf\_tat[i];

total\_wtime[3]+=rr\_wt[i];

total\_atime[3]+=rr\_tat[i];

}

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

total\_atime[i]=total\_atime[i]/10;

total\_wtime[i]= total\_wtime[i]/10;

}

fcfs\_awt=total\_wtime[0];

fcfs\_atat=total\_atime[0];

sjf\_awt =total\_wtime[1];

sjf\_atat=total\_atime[1];

rr\_atat=total\_atime[3];

rr\_awt=total\_wtime[3];

srtf\_atat = total\_atime[2];

srtf\_awt = total\_wtime[2];

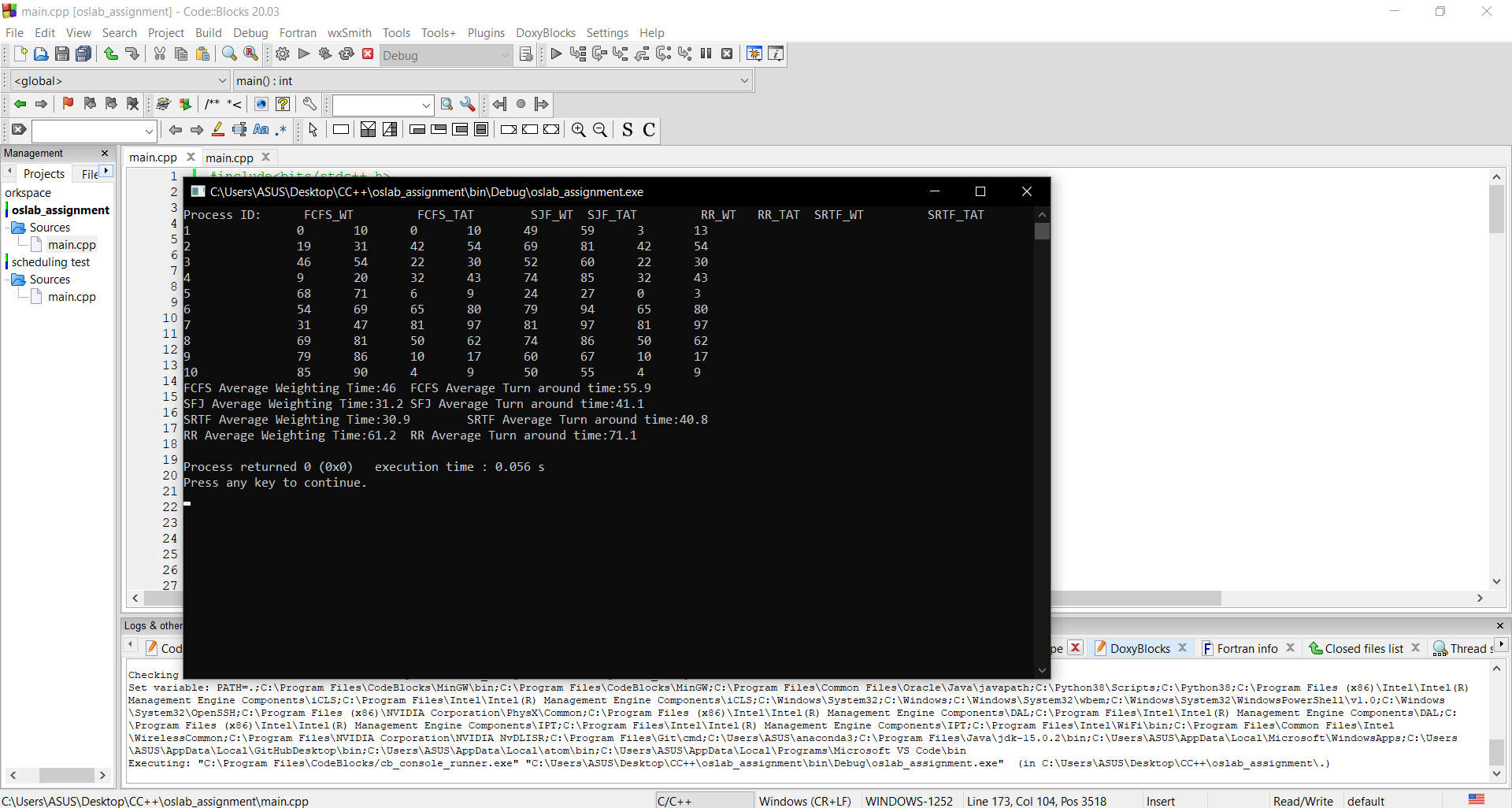
cout<<"FCFS Average Weighting Time:"<<fcfs\_awt<<"\t"<<"FCFS Average Turn around time:"<<fcfs\_atat<<"\n";

cout<<"SFJ Average Weighting Time:"<<sjf\_awt<<"\t"<<"SFJ Average Turn around time:"<<sjf\_atat<<"\n";

cout<<"SRTF Average Weighting Time:"<<srtf\_awt<<"\t"<<"SRTF Average Turn around time:"<<srtf\_atat<<"\n";

cout<<"RR Average Weighting Time:"<<rr\_awt<<"\t"<<"RR Average Turn around time:"<<rr\_atat<<"\n";

}



In my opinion SRTF is the best algorithm for this problem as it has the least average turnaround time and average waiting time.