1. **First come first serve (FCFS)**

**Code:**

#include<iostream>

using namespace std;

int main()

{

int n, bt[20], wt[20], tat[20], averageWaitingTime = 0, aveargeTurnAroundTime = 0;

cout<<"Enter total number of processes(maximum 20):";

cin>>n;

cout<<"\nEnter Process Burst Time\n";

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

{

cout<<"P["<<i+1<<"]:";

cin>>bt[i];

}

wt[0] = 0;

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

{

wt[i] = 0;

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

wt[i] += bt[j];

}

cout<<"\nProcess\t\tBurst Time\tWaiting Time\tTurnaround Time";

//calculating turnaround time

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

{

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

averageWaitingTime += wt[i];

aveargeTurnAroundTime += tat[i];

cout<<"\nP["<<i+1<<"]"<<"\t\t"<<bt[i]<<"\t\t"<<wt[i]<<"\t\t"<<tat[i];

}

averageWaitingTime /= n;

aveargeTurnAroundTime /= n;

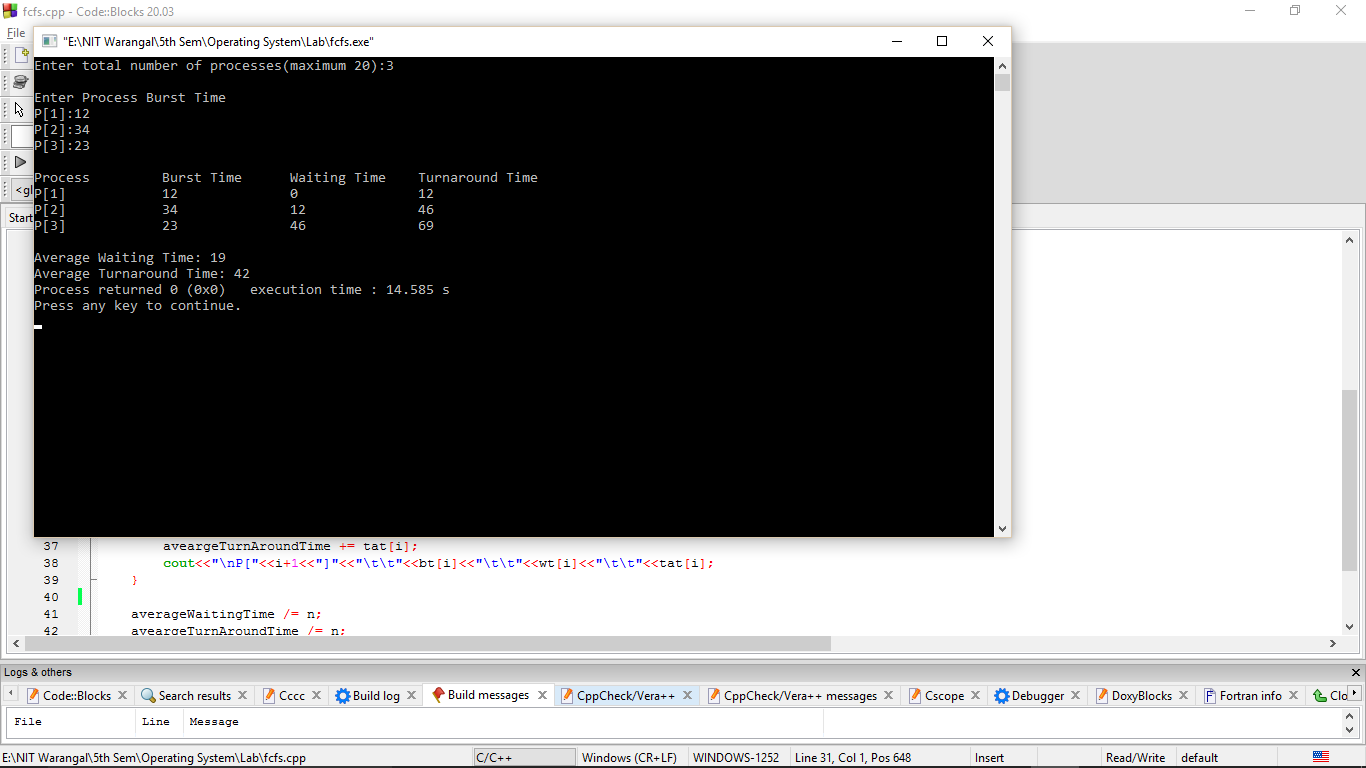
cout<<"\n\nAverage Waiting Time: "<<averageWaitingTime;

cout<<"\nAverage Turnaround Time: "<<aveargeTurnAroundTime;

return 0;

}

**Output:**



1. **Priority Scheduling**

**Code:**

#include<bits/stdc++.h>

using namespace std;

struct process{

int burstTime;

int priority;

int process\_number;

};

typedef struct process \*prc;

bool comp(prc A, prc B)

{

return A->priority < B->priority;

}

int main()

{

int n;

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

cin>>n;

vector<prc>vec;

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

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

{

int a, b;

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

cout<<"Burst Time: ";

cin>>a;

cout<<"Priority: ";

cin>>b;

prc p = new process;

p->burstTime = a;

p->priority = b;

p->process\_number = i + 1;

vec.push\_back(p);

//contains process number

}

// less number, higher priority

sort(vec.begin(), vec.end(), comp);

int wt[n], tat[n];

wt[0]=0;

int total = 0, pos, temp, avg\_wt, avg\_tat, i, j; //waiting time for first process is zero

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

{

wt[i]=0;

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

wt[i] += vec[j]->burstTime;

total += wt[i];

}

avg\_wt = total/n;

total = 0;

cout<<"\nProcess\t Burst Time \tWaiting Time\tTurnaround Time";

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

{

tat[i] = vec[i]->burstTime + wt[i];

total += tat[i];

cout<<"\nP["<<vec[i]->process\_number<<"]\t\t "<<vec[i]->burstTime<<"\t\t "<<wt[i]<<"\t\t\t"<<tat[i];

}

avg\_tat = total/n; //average turnaround time

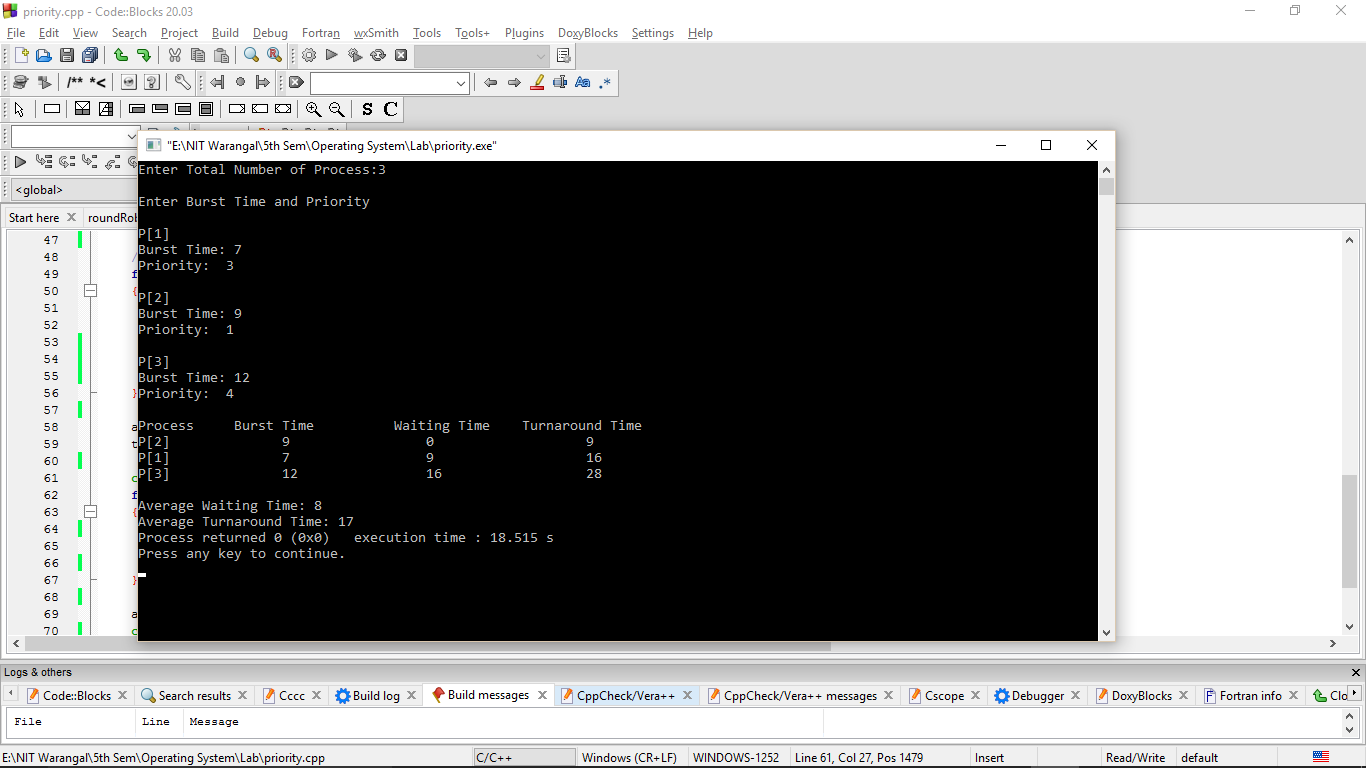
cout<<"\n\nAverage Waiting Time: "<<avg\_wt;

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

return 0;

}

**Output:**



1. **Shortest Job First**

**Code:**

#include<iostream>

using namespace std;

int main()

{

int n, temp, tt = 0, m, d = 0, i, j;

float atat = 0, awt = 0, stat = 0, swt = 0;

cout<<"enter no of process:"<<endl;

cin>>n;

int a[n], b[n], e[n], tat[n], wt[n];

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

{

cout<<"enter arrival time: ";

cin>>a[i];

}

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

{

cout<<"enter burst time: ";

cin>>b[i];

}

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

{

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

if(b[i] > b[j])

{

temp = a[i];

a[i] = a[j];

a[j] = temp;

temp = b[i];

b[i] = b[j];

b[j] = temp;

}

}

}

m = a[0];

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

{

if(m > a[i])

{

cout<<"hi";

m = a[i];

d = i;

}

}

tt = m;

e[d] = tt + b[d];

tt = e[d];

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

{

if(a[i] != m)

{

e[i] = b[i] + tt;

tt = e[i];

}

}

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

{

tat[i] = e[i] - a[i];

stat = stat + tat[i];

wt[i] = tat[i] - b[i];

swt = swt + wt[i];

}

atat = stat/n;

awt = swt/n;

cout<<"Process Arrival-time(s) Burst-time(s) Waiting-time(s) Turnaround-time(s)\n";

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

{

cout<<"P"<<i+1<<" "<<a[i]<<" "<<b[i]<<" "<<wt[i]<<" "<<tat[i]<<endl;

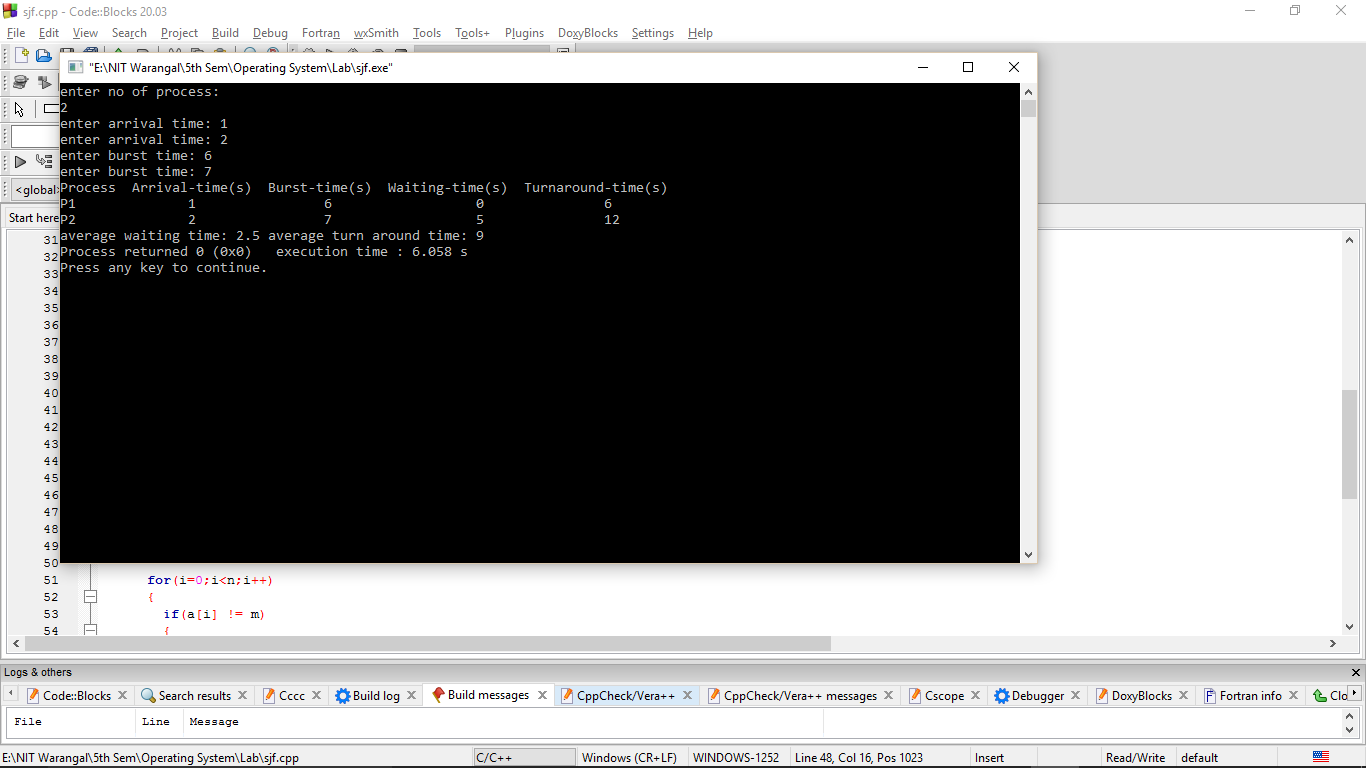
}

cout<<"average waiting time: "<<awt<<" average turn around time: "<<atat;

return 0;

}

**Output:**



1. **Round Robin Algorithm**

**Code:**

#include<iostream>

using namespace std;

// Function to find the waiting time for all

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; // Current time

while (1) {

bool done = true;

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

{

if (rem\_bt[i] > 0)

{

done = false; // There is a pending process

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;

}

}

// Function to calculate turn around time

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];

}

// Function to calculate average time

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[] = { 4, 5, 6};

int n = sizeof processes / sizeof processes[0];

int burst\_time[] = {10, 15, 20};

int quantum = 3;

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

return 0;

}

**Output:**

