Working with Process Scheduling algorithms

1. Given the list of processes, their CPU burst times and arrival time write a program to implement Shortest Job First CPU scheduling algorithm. Compute the waiting time and turnaround time for each processor. Print the table of information which contains process name, execution time, waiting time and turnaround time. Finally print the average waiting time and average turnaround time.

**Process Burst Time Arrival Time**

P1 12 0

P2 25 2

P3 13 1

P4 7 0

P5 11 5

#include <iostream>

using namespace std;

int pro[10][6];

void swap(int\* a, int\* b)

{

int temp = \*a;

\*a = \*b;

\*b = temp;

}

void arrangeArrival(int num, int pro[][6])

{

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

for (int j = 0; j < num - i - 1; j++) {

if (pro[j][2] > pro[j + 1][2]) {

for (int k = 0; k < 5; k++) {

swap(pro[j][k], pro[j + 1][k]);

}

}

}

}

}

void completionTime(int num, int pro[][6])

{

int temp, val;

pro[0][3] = pro[0][2] + pro[0][1];

pro[0][5] = pro[0][3] - pro[0][2];

pro[0][4] = pro[0][5] - pro[0][1];

for (int i = 1; i < num; i++) {

temp = pro[i - 1][3];

int low = pro[i][1];

for (int j = i; j < num; j++) {

if (temp >= pro[j][2] && low >= pro[j][1]) {

low = pro[j][1];

val = j;

}

}

pro[val][3] = temp + pro[val][1];

pro[val][5] = pro[val][3] - pro[val][2];

pro[val][4] = pro[val][5] - pro[val][1];

for (int k = 0; k < 6; k++) {

swap(pro[val][k], pro[i][k]);

}

}

}

int main()

{

int num, temp;

cout << "Enter number of Process: ";

cin >> num;

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

cout << "\nEnter Process Id: ";

cin >> pro[i][0];

cout << "Enter Burst Time: ";

cin >> pro[i][1];

cout << "Enter Arrival Time: ";

cin >> pro[i][2];

}

cout << "\nProcess ID\tBurst Time\tArrival Time\n";

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

cout << pro[i][0] << "\t\t" << pro[i][1] << "\t\t"

<< pro[i][2] << "\n";

}

arrangeArrival(num, pro);

completionTime(num, pro);

cout << "\nProcess ID\tBurst Time\tArrival Time\tWaiting Time\tTurnaround Time\n";

int total\_wt = 0, total\_tt=0;

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

cout << pro[i][0] << "\t\t" << pro[i][1] << "\t\t" << pro[i][2]

<< "\t\t" << pro[i][4] << "\t\t" << pro[i][5] << "\n";

total\_wt = total\_wt + pro[i][4];

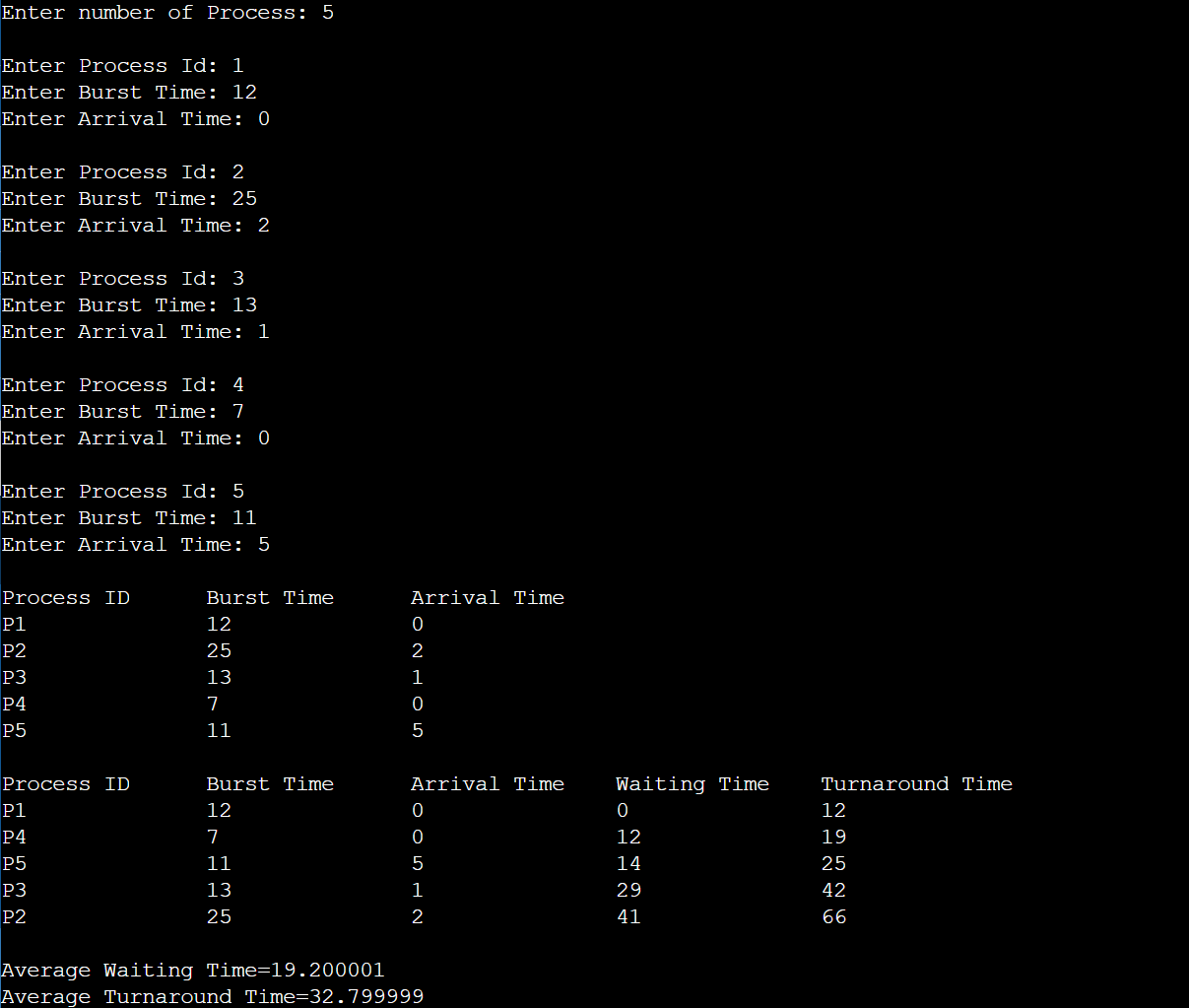
total\_tt = total\_tt + pro[i][5];

}

printf("\nAverage Waiting Time=%f",(float)total\_wt/num);

printf("\nAverage Turnaround Time=%f\n",(float)total\_tt/num);

}



1. Write a program to implement Shortest Job First CPU scheduling algorithm where Processes will arrive at the same time. Display the following results in a table format.
2. Waiting time and turnaround time of every process.
3. Average waiting time and turnaround time.

**Process Burst Time**

P1 12

P2 22

P3 14

P4 7

P5 10

#include <iostream>

using namespace std;

int pro[10][6];

void swap(int\* a, int\* b)

{

int temp = \*a;

\*a = \*b;

\*b = temp;

}

void arrangeBurst(int num, int pro[][6])

{

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

for (int j = 0; j < num - i - 1; j++) {

if (pro[j][1] > pro[j + 1][1]) {

for (int k = 0; k < 5; k++) {

swap(pro[j][k], pro[j + 1][k]);

}

}

}

}

}

void completionTime(int num, int pro[][6])

{

pro[0][4] = 0;

pro[0][5]=pro[0][1]+pro[0][4];

for (int i = 1; i < num; i++) {

pro[i][4]=0;

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

pro[i][4]+=pro[j][1];

pro[i][5]=pro[i][1]+pro[i][4];

}

}

int main()

{

int num, temp;

cout << "Enter number of Process: ";

cin >> num;

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

cout << "\nEnter Process Id: ";

cin >> pro[i][0];

cout << "Enter Burst Time: ";

cin >> pro[i][1];

}

cout << "\nProcess ID\tBurst Time\n";

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

cout <<"P"<<pro[i][0] << "\t\t" << pro[i][1] << "\n";

}

arrangeBurst(num, pro);

completionTime(num, pro);

cout << "\nProcess ID\tBurst Time\tWaiting Time\tTurnaround Time\n";

int total\_wt = 0, total\_tt=0;

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

cout <<"P"<< pro[i][0] << "\t\t" << pro[i][1]

<< "\t\t" << pro[i][4] << "\t\t" << pro[i][5] << "\n";

total\_wt = total\_wt + pro[i][4];

total\_tt = total\_tt + pro[i][5];

}

printf("\nAverage Waiting Time=%f",(float)total\_wt/num);

printf("\nAverage Turnaround Time=%f\n",(float)total\_tt/num);

}

Text

Description automatically generated

1. Given the list of processes, their CPU burst times and arrival time, display/print the Gantt chart for FCFS. Compute the waiting time and turnaround time for each processor. Print the table of information which contains process name, execution time, waiting time and turnaround time. Finally print the average waiting time and average turnaround time.

**Process Burst Time Arrival Time**

P1 12 0

P2 25 2

P3 13 1

P4 7 0

P5 11 5

#include<iostream>

using namespace std;

void findWaitingTime(int processes[], int n, int bt[], int wt[], int at[])

{

int temp = 0;

wt[0] = 0;

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

{

temp = temp + bt[i-1];

wt[i] = temp - at[i];

if (wt[i] < 0)

wt[i] = 0;

}

}

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 at[])

{

int wt[n], tat[n];

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

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

cout << "Processes " << " Burst Time " << " Arrival Time "

<< " Waiting Time " << " Turn-Around Time\n";

int total\_wt = 0, total\_tat = 0;

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\t" << at[i]

<< "\t\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 n;

cout << "Enter number of Process: ";

cin >> n;

int pro[n], burst\_time[n], arrival\_time[n];

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

cout << "\nEnter Process Id: ";

cin >> pro[i];

cout << "Enter Burst Time: ";

cin >> burst\_time[i];

cout << "Enter Arrival Time: ";

cin >> arrival\_time[i];

}

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

return 0;

}

Text

Description automatically generated with medium confidence