**Pandit Deendayal Energy University**

**School of Technology**

**Computer Science and Engineering**

Course: **OS Lab**

Course Code: **20CP207P**

Program: **B-Tech.**

A.Y.: **2023-2024**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Group Task 1**

| Sr.no | Name | Roll no |
| --- | --- | --- |
| 1 | Darshil Padmani | 22BCP321 |
| 2 | Dhariya Savaliya | 22BCP338 |
| 3 | Jenil Patel | 22BCP332 |
| 4 | Kashyap Vekariya | 22BCP346 |
| 5 | Deep Mathukiya | 22BCP348 |

**Set-I : CPU Scheduling**

1. Shortest Remaining Time Next (SRTN) algorithm
2. Shortest Job First (SJF) algorithm
3. Round Robin (RR) algorithm
4. Priority (Pre-emptive) scheduling algorithm
5. **Shortest Remaining Time Next (SRTN) algorithm**

#include <iostream>

using namespace std;

struct Process {

int pid;

int art;

int bt;

};

void findTAT(Process proc[], int n, int tat[])

{

int rt[n];

int complete = 0, t = 0, minm = 999;

int shortest = 0, finish\_time;

bool check = false;

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

rt[i] = proc[i].bt;

if(proc[i].bt == 0){

complete++;

}

}

while (complete != n) {

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

if ((proc[j].art <= t) &&

(rt[j] < minm) && rt[j] > 0) {

minm = rt[j];

shortest = j;

check = true;

}

}

if (check == false) {

cout<< "| \t";

t++;

continue;

}

rt[shortest]--;

minm = rt[shortest];

if(minm == 0)

minm = 999;

if (rt[shortest] == 0) {

complete++;

check = false;

finish\_time = t + 1;

tat[shortest] = finish\_time -proc[shortest].art;

if (tat[shortest] < 0)

tat[shortest] = 0;

}

cout <<"|"<< proc[shortest].pid << "\t";

t++;

}

cout<<"\n";

}

void findWT(Process proc[], int n,

int tat[], int wt[])

{

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

wt[i] = tat[i] - proc[i].bt;

}

void findavgTime(Process proc[], int n)

{

int wt[n], tat[n], total\_wt = 0,

total\_tat = 0;

findTAT(proc, n, tat);

findWT(proc, n, tat,wt);

cout << "P\t\t"

<< "BT \t\t"

<< "WT \t\t"

<< "TAT \t\t \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\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;

}

int main()

{

int n;

cout<<"Enter the number of process : ";

cin>>n;

cout<<"Insert the process informatio\n";

Process proc[n];

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

cout<<"Enter the process id: ";

cin>>proc[i].pid;

cout<<"Enter the arrival time: ";

cin>>proc[i].art;

cout<<"Enter the burst time: ";

cin>>proc[i].bt;

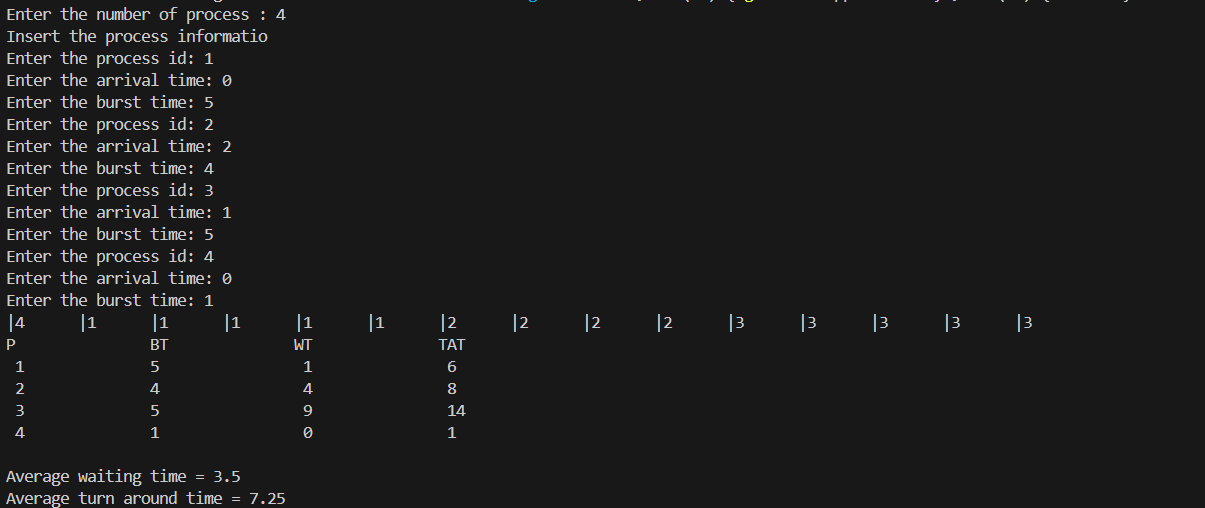
}

findavgTime(proc, n);

return 0;

}

**OUTPUT:**



1. **SJF**

#include<iostream>

using namespace std;

struct Process {

int pid;

int art;

int bt;

};

void findTAT(Process proc[], int n, int tat[])

{

int rt[n];

int complete = 0, t = 0, minm = 999;

int shortest = 0, finish\_time;

bool check = false;

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

rt[i] = proc[i].bt;

if(proc[i].bt == 0){

complete++;

}

}

while (complete != n) {

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

if ((proc[j].art <= t) &&

(rt[j] < minm) && rt[j] > 0) {

minm = rt[j];

shortest = j;

check = true;

}

}

if (check == false) {

cout<< "| \t";

t++;

continue;

}

rt[shortest] == 0;

minm = 999;

t += proc[shortest].bt;

tat[shortest] = t - proc[shortest].art;

complete++;

cout <<"|"<< proc[shortest].pid << "\t";

}

cout << endl;

}

void findWT(Process proc[], int n,

int tat[], int wt[])

{

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

wt[i] = tat[i] - proc[i].bt;

}

void findavgTime(Process proc[], int n)

{

int wt[n], tat[n], total\_wt = 0, total\_tat = 0;

findTAT(proc, n, tat);

findWT(proc, n, tat,wt);

cout << "P\t\t"

<< "BT \t\t"

<< "WT \t\t"

<< "TAT \t\t \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\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;

}

int main()

{

int n;

cout<<"Enter the number of process : ";

cin>>n;

cout<<"Insert the process information\n";

Process proc[n];

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

cout<<"Enter the process id: ";

cin>>proc[i].pid;

cout<<"Enter the arrival time: ";

cin>>proc[i].art;

cout<<"Enter the burst time: ";

cin>>proc[i].bt;

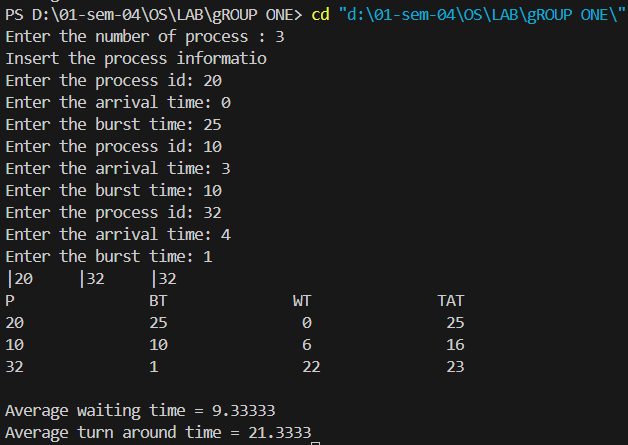
}

findavgTime(proc, n);

return 0;

}

**OUTPUT:**



1. **Round Robin (RR) algorithm**

#include <iostream>

#include <vector>

using namespace std;

struct Process {

int pid;

int art;

int bt;

int remaining\_time;

int wt;

int tat;

};

void round\_robin\_scheduling(vector<Process>& processes, int time\_quantum) {

int n = processes.size();

int current\_time = 0;

vector<int> remaining\_bt(n, 0);

vector<bool> is\_completed(n, false);

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

remaining\_bt[i] = processes[i].bt;

}

int completed\_processes = 0;

int index = 0;

bool is\_first\_process = true;

cout << "Gantt Chart:\n";

while (completed\_processes != n) {

bool is\_cpu\_idle = true;

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

if (processes[i].art <= current\_time && !is\_completed[i]) {

is\_cpu\_idle = false;

if (is\_first\_process) {

cout << "| " << processes[i].pid << " ";

is\_first\_process = false;

} else {

cout << "| " << processes[i].pid << " ";

}

if (remaining\_bt[i] > time\_quantum) {

current\_time += time\_quantum;

remaining\_bt[i] -= time\_quantum;

} else {

current\_time += remaining\_bt[i];

processes[i].wt = current\_time - processes[i].bt - processes[i].art;

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

remaining\_bt[i] = 0;

is\_completed[i] = true;

completed\_processes++;

}

}

}

if (is\_cpu\_idle) {

current\_time++;

cout << "| -- ";

}

}

cout << "|\n";

}

void print\_results(vector<Process>& processes) {

cout << "P\t\tBT\tWT\tTAT\n";

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

for (int i = 0; i < processes.size(); i++) {

cout << processes[i].pid << "\t\t" << processes[i].bt << "\t" << processes[i].wt << "\t" << processes[i].tat << endl;

total\_wt += processes[i].wt;

total\_tat += processes[i].tat;

}

cout << "\nAverage Waiting Time: " << total\_wt / processes.size() << endl;

cout << "Average Turnaround Time: " << total\_tat / processes.size() << endl;

}

int main() {

int n;

cout << "Enter the number of processes: ";

cin >> n;

vector<Process> processes(n);

cout << "Insert the process information\n";

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

cout << "Enter the process id: ";

cin >> processes[i].pid;

cout << "Enter the arrival time: ";

cin >> processes[i].art;

cout << "Enter the burst time: ";

cin >> processes[i].bt;

processes[i].remaining\_time = processes[i].bt;

processes[i].wt = 0;

processes[i].tat = 0;

}

int time\_quantum;

cout << "Enter the time quantum: ";

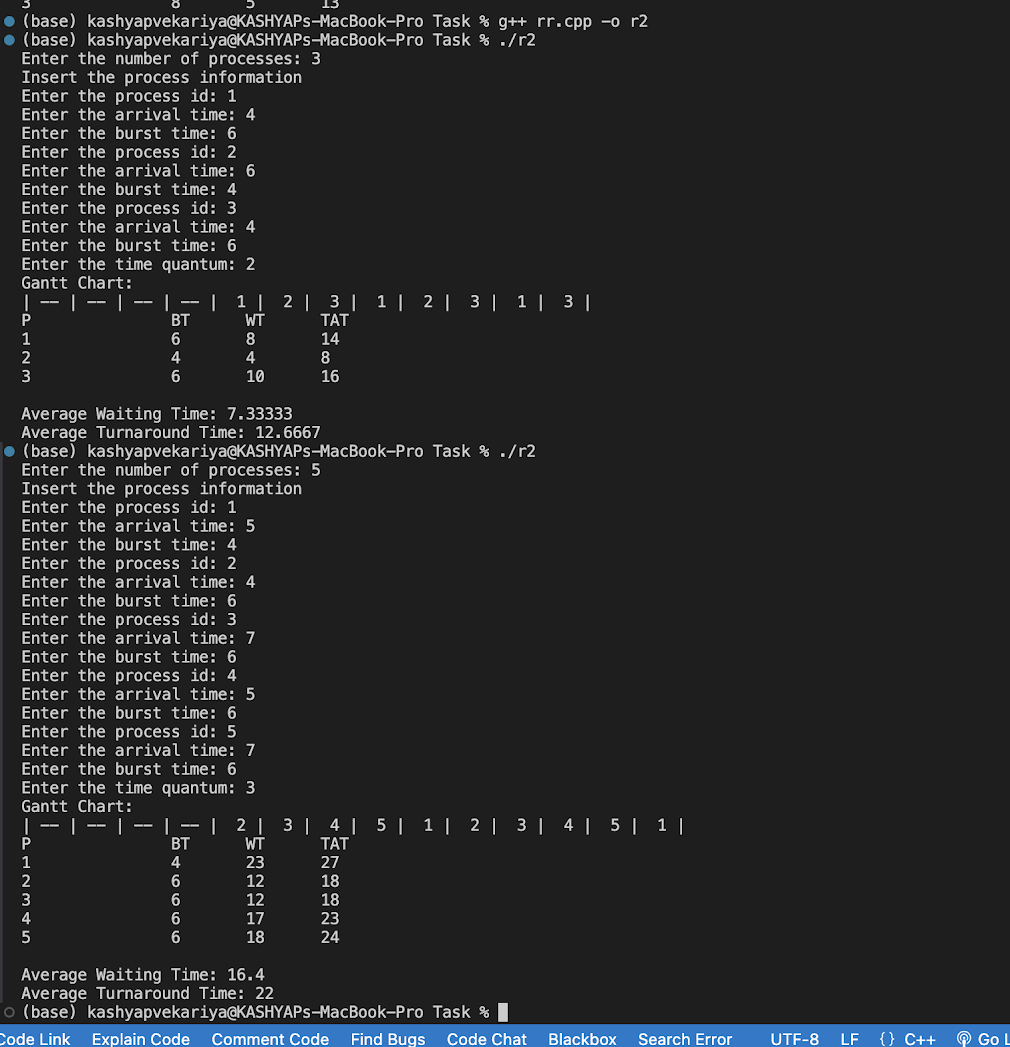
cin >> time\_quantum;

round\_robin\_scheduling(processes, time\_quantum);

print\_results(processes);

return 0;

}



**4. Priority (Pre-emptive) scheduling algorithm**

#include <iostream>

using namespace std;

struct Process {

int pid;

int art;

int bt;

int priority;

};

void findTAT(Process proc[], int n, int tat[])

{

int rt[n];

int complete = 0, t = 0, shortest = 0;

bool check = false;

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

rt[i] = proc[i].bt;

}

while (complete != n) {

int minm = 999, min\_priority = 999; // Initialize with a high value

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

if ((proc[j].art <= t) && (rt[j] > 0) && (proc[j].priority < min\_priority)) {

minm = rt[j];

min\_priority = proc[j].priority;

shortest = j;

check = true;

}

}

if (check == false) {

cout << "| \t";

t++;

continue;

}

rt[shortest]--;

minm = rt[shortest];

if (minm == 0) {

minm = 999;

complete++;

check = false;

int finish\_time = t + 1;

tat[shortest] = finish\_time - proc[shortest].art;

if (tat[shortest] < 0)

tat[shortest] = 0;

}

cout << "|" << proc[shortest].pid << "\t";

t++;

}

cout << "\n";

}

void findWT(Process proc[], int n, int tat[], int wt[])

{

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

wt[i] = tat[i] - proc[i].bt;

}

void findavgTime(Process proc[], int n)

{

int wt[n], tat[n], total\_wt = 0, total\_tat = 0;

findTAT(proc, n, tat);

findWT(proc, n, tat, wt);

cout << "P\t\t"

<< "BT \t\t"

<< "WT \t\t"

<< "TAT \t\t \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\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;

}

int main()

{

int n;

cout << "Enter the number of processes: ";

cin >> n;

cout << "Insert the process information\n";

Process proc[n];

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

cout << "Enter the process id: ";

cin >> proc[i].pid;

cout << "Enter the arrival time: ";

cin >> proc[i].art;

cout << "Enter the burst time: ";

cin >> proc[i].bt;

cout << "Enter the priority: ";

cin >> proc[i].priority;

}

findavgTime(proc, n);

return 0;

}

