**2100290120007\_Lab\_3**

RR Scheduling for same arrival time

#include <bits/stdc++.h>

using namespace std;

void WaitingTime(int processes[],int n,int bt[],int wt[],int Tq){

int rem\_bt[n];

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

rem\_bt[i] = bt[i];

int t = 0; // Current time

// Keep traversing processes in round robin manner

// until all of them are not done.

while (1)

{

bool done = true;

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

{

// If bt>0 then further processes

if (rem\_bt[i] > 0)

{

done = false; // There is a pending process

if (rem\_bt[i] > Tq)

{

// Increase the value of t i.e. shows

// how much time a process has been processed

t =t+ Tq;

// Decrease the burst\_time of current process

// by quantum

rem\_bt[i] =rem\_bt[i]- Tq;

}

// If burst time is smaller than or equal to

// quantum. Last cycle for this process

else

{

// Increase the value of t i.e. shows

// how much time a process has been processed

t = t + rem\_bt[i];

// Waiting time is current time minus time

// used by this process

wt[i] = t - bt[i];

// As the process gets fully executed

// make its remaining burst time = 0

rem\_bt[i] = 0;

}

}

}

// If all processes are done

if (done == true)

break;

}

}

void TurnAroundTime(int processes[], int n,

int bt[], int wt[], int tat[])

{

// calculating turnaround time by adding bt[i] + wt[i]

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

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

}

void avgTime(int processes[],int n,int bt[],int Tq){

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

// fint waiting time of all processes

WaitingTime(processes, n, bt, wt, Tq);

// find turn around time of all processes

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

cout << "PN\t"<< "BT\t"<< " WT\t" << " TAT\n\n";

// Calculate total waiting time and total turnaround time

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

{

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

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

cout << " " << i+1 << "\t" << bt[i] <<"\t "<< wt[i] <<"\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[6] = {1,2,3,4,5,6};

// function for Burst time

// for same arrivel time

int burst\_time[] = {7,4,15,11,20,9};

// Time quantum

int Tq = 5;

avgTime(processes, 6, burst\_time, Tq);

return 0;

}

**Output ->**

****

RR Scheduling for different arrival time

#include <bits/stdc++.h>

using namespace std;

void queueUpdation(int queue[],int timer,int arrival[],int n, int maxProccessIndex){

int zeroIndex;

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

if(queue[i] == 0){

zeroIndex = i;

break;

}

}

queue[zeroIndex] = maxProccessIndex + 1;

}

void queueMaintainence(int queue[], int n){

for(int i = 0; (i < n-1) && (queue[i+1] != 0) ; i++){

int temp = queue[i];

queue[i] = queue[i+1];

queue[i+1] = temp;

}

}

void checkNewArrival(int timer, int arrival[], int n, int maxProccessIndex,int queue[]){

if(timer <= arrival[n-1]){

bool newArrival = false;

for(int j = (maxProccessIndex+1); j < n; j++){

if(arrival[j] <= timer){

if(maxProccessIndex < j){

maxProccessIndex = j;

newArrival = true;

}

}

}

//adds the incoming process to the ready queue

//(if any arrives)

if(newArrival)

queueUpdation(queue,timer,arrival,n, maxProccessIndex);

}

}

//Driver Code

int main(){

int n,tq, timer = 0, maxProccessIndex = 0;

float avgWait = 0, avgTT = 0;

cout << "\nEnter the time quanta : ";

cin>>tq;

cout << "\nEnter the number of processes : ";

cin>>n;

int arrival[n], burst[n], wait[n], turn[n], queue[n], temp\_burst[n];

bool complete[n];

cout << "\nEnter the arrival time of the processes : ";

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

cin>>arrival[i];

cout << "\nEnter the burst time of the processes : ";

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

cin>>burst[i];

temp\_burst[i] = burst[i];

}

for(int i = 0; i < n; i++){ //Initializing the queue and complete array

complete[i] = false;

queue[i] = 0;

}

while(timer < arrival[0]) //Incrementing Timer until the first process arrives

timer++;

queue[0] = 1;

while(true){

bool flag = true;

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

if(temp\_burst[i] != 0){

flag = false;

break;

}

}

if(flag)

break;

for(int i = 0; (i < n) && (queue[i] != 0); i++){

int ctr = 0;

while((ctr < tq) && (temp\_burst[queue[0]-1] > 0)){

temp\_burst[queue[0]-1] -= 1;

timer += 1;

ctr++;

//Checking and Updating the ready queue until all the processes arrive

checkNewArrival(timer, arrival, n, maxProccessIndex, queue);

}

//If a process is completed then store its exit time

//and mark it as completed

if((temp\_burst[queue[0]-1] == 0) && (complete[queue[0]-1] == false)){

//turn array currently stores the completion time

turn[queue[0]-1] = timer;

complete[queue[0]-1] = true;

}

//checks whether or not CPU is idle

bool idle = true;

if(queue[n-1] == 0){

for(int i = 0; i < n && queue[i] != 0; i++){

if(complete[queue[i]-1] == false){

idle = false;

}

}

}

else

idle = false;

if(idle){

timer++;

checkNewArrival(timer, arrival, n, maxProccessIndex, queue);

}

//Maintaining the entries of processes

//after each premption in the ready Queue

queueMaintainence(queue,n);

}

}

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

turn[i] = turn[i] - arrival[i];

wait[i] = turn[i] - burst[i];

}

cout << "\nProgram No.\tArrival Time\tBurst Time\tWait Time\tTurnAround Time"

<< endl;

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

cout<<i+1<<"\t\t"<<arrival[i]<<"\t\t"

<<burst[i]<<"\t\t"<<wait[i]<<"\t\t"<<turn[i]<<endl;

}

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

avgWait += wait[i];

avgTT += turn[i];

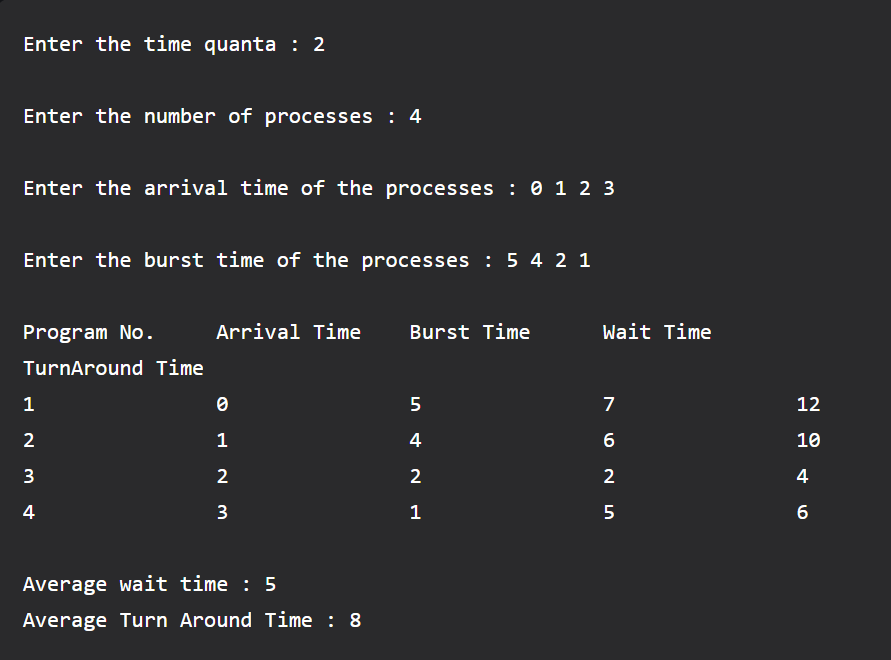
}

cout<<"\nAverage wait time : "<<(avgWait/n)

<<"\nAverage Turn Around Time : "<<(avgTT/n);

return 0;

}

****