Student Name: Anmol Pradhan

Student ID: 11813478

Email Address : [anamolpradhan@gmail.com](mailto:anamolpradhan@gmail.com)

GitHub Link: https://github.com/anmolpradhan/OSCA-CSE316

Description

In the problem we have to design a scheduler using round robin scheduler. Than arrival time and burst time requirement of the process is taken as input where the scheduler

1. schedules the processes by interrupting the processor after every 6 units of time and does consider the completion of the process in this iteration which is first iteration.

1. Than again at the second iteration the scheduler than checks for the number of process waiting for the processor and allots the processor to the process but interrupting the processor every 10 unit of time instead of 6 as in first iteration and but similar to first iteration considers the completion of the processes in this iteration.
2. Than at last the scheduler checks the number of processes waiting in the queue for the processor after the second iteration and gives the processor to the process which needs more time to complete than the other processes to go in the terminated state.

Algorithm

Step -1: Declare array arrivalTime[ ],burstTime[ ], remainingTime[ ],

waitingTime[ ], turnAroundTime[ ] and int variable n.

Step -2: Take input number of process in size.

Step -3: Repeat for int I = 0,1,2.... Size-1

Take arrival time and burst time of the process as input.

Step -4: remainingProcess = size , timeQuantum\_i1=6, timeQuantum\_i2=10;

currentTime=0 and Repeat step -5 to step - 9 while remainingProcess != 0

Step -5: if(remainingBTime[Process\_no]<timeQuantum\_i1 && remainingBTime

[Process\_no]>=0&&currentTime<9)

currentTime+=remainingBTime[Process\_no];

remainingBTime[Process\_no]=0

indicator = 1

timeQuantum\_itration++

remainingProcess--

Step -6: else if(remainingBTime[Process\_no]>0&&currentTime<9)

if(timeQuantum\_itration==1)

remainingBTime[Process\_no]-=timeQuantum\_i1

timeQuantum\_iteration++

currentTime+=timeQuantum

else if(timeQuantum\_itration==2)

remainingBTime[Process\_no]-=timeQuantum\_i2

currentTime+=timeQuantum\_i2

Step -7: else if(remainingBTime[Process\_no]<9 && remainingBTime

[Process\_no]>=3&&currentTime<9)

currentTime+=remainingBTime[Process\_no];

remainingBTime[Process\_no]=0;

remainingProcess-- , indicator = 1

Step -8: else if(remainingBTime[Process\_no]>3&&currentTime<9)

remainingBTime[Process\_no]-=timeQuantum\_i2

currentTime+=timeQuantum\_i2

Step -9: if(remainingBTime[Process\_no]==0 && indicator==1)

remainingProcess--

remainingTime[Process\_no]=currentTime

turnAroundTime[Process\_no]=remainingTime[Process\_no]-

arrivalTime[Process\_no]

waitingTime[Process\_no]=turnAroundTime[Process\_no]-

burstTime[Process\_no]

indicator = 0

Step -10: End of step 4 loop, set:- Process\_no=0

Repeat Step -11 to Step - 12 while remaining process != 0

Step -11: Pick a job whose burst time is lesser among all of them.

Step -12: Execute that job and set

remainingTime[Process\_no]=currentTime

turnAroundTime[Process\_no]=remainingTime[Process\_no]-arrivalTime[Process\_no]

waitingTime[Process\_no]=trunaroundTime[Process\_no]-burstTime

Step -13: End of step step 10 loop.

Step -14: Repeat for int I = 0,1,2.... Size-1

A) Calculate average waiting time and average burst time.

B) Print all the information.

Time Complexity

Overall Time complexity of the program is O(n2)

Code Snippet

*#include* <iostream>

using namespace std;

class Scheduler{

private:

int processNo, currentTime, remainingProcess, indicator = 0, wait = 0, turnAroundTime = 0, arrivalTime[10], burstTime[10], remainingTime[10], x = 1;

public:

void process(int n){

remainingProcess = n;

*for* (processNo = 0; processNo < n; processNo++)

{

cout<<"\nEnter Arrival time and Burst time for Process P"<<processNo + 1<<"\n";

cin>>arrivalTime[processNo];

cin>>burstTime[processNo];

remainingTime[processNo] = burstTime[processNo];

}

cout<<"Time quantum for first round is 6.\n";

int timeQuantum = 6;

currentTime = 0;

*for* (processNo = 0; remainingProcess != 0;)

{

*if* (remainingTime[processNo] <= timeQuantum && remainingTime[processNo] > 0)

{

currentTime += remainingTime[processNo];

remainingTime[processNo] = 0;

indicator = 1;

}

*else* *if* (remainingTime[processNo] > 0)

{

remainingTime[processNo] -= timeQuantum;

currentTime += timeQuantum;

}

*else* *if* (remainingTime[processNo] == 0 && indicator == 1)

{

cout<<"Process:"<<processNo;

remainingProcess--;

cout<<"P "<<processNo + 1;

cout<<"\t\t\t"<<currentTime - arrivalTime[processNo];

cout<<"\t\t\t"<< currentTime - burstTime[processNo] - arrivalTime[processNo]<<"\n";

wait += currentTime - arrivalTime[processNo] - burstTime[processNo];

turnAroundTime += currentTime - arrivalTime[processNo];

indicator = 0;

}

*if* (processNo == n - 1)

{

x++;

*if* (x == 2)

{

processNo = 0;

timeQuantum = 10;

cout<<"Time quantum for second round is 10. \n";

}

*else*

{

*break*;

}

}

*else* *if* (currentTime >= arrivalTime[processNo + 1])

{

processNo++;

}

*else*

{

processNo = 0;

}

}

thirdIteration(n, remainingTime, currentTime, arrivalTime);

}

void thirdIteration(int n, int remainingTime[10], int currenttime, int arrivalTime[10])

{

float averageWait, averageTurnAround;

int bTime[20], processNo[20], waitingTime[20], turnAroundTime[20];

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

{

bTime[i] = remainingTime[i];

waitingTime[i] = currenttime - arrivalTime[i] - bTime[i];

processNo[i] = i + 1;

}

int loc = 0, total;

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

{

loc = i;

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

{

*if* (bTime[j] < bTime[loc])

{

loc = j;

}

}

int temp = bTime[i];

bTime[i] = bTime[loc];

bTime[loc] = temp;

temp = processNo[i];

processNo[i] = processNo[loc];

processNo[loc] = temp;

}

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

{

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

{

waitingTime[i] += bTime[j];

}

total += waitingTime[i];

}

averageWait = (float)total / n;

total = 0;

cout<<"\nProcess\t\tBurst time\t\twaiting time\t\tTurn Around Time";

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

{

turnAroundTime[i] = bTime[i] + waitingTime[i];

total = total + turnAroundTime[i];

cout << "\nP0" << processNo[i] << "\t\t\t" << bTime[i] << "\t\t\t" << waitingTime[i] << "\t\t\t" << turnAroundTime[i];

}

averageTurnAround = (float)total / n;

cout<<"\nAverage Waiting Time:"<<averageWait<<"\nAverage Turn Around Time:"<<averageTurnAround<<"\n";

}

};

int main()

{

cout << "Enter the no. of processes:";

int n;

cin >> n;

Scheduler s;

s.process(n);

*return* 0;

}

Boundary Conditions

Total no. of processes must be less than equal to 10 because of while declaring an array only 10 size is allocated.

The processes Burst Time and Arrival Time must be reliable without any random values. If the random input is passed than the program does not return the desired output

Additional Algorithm

Object Oriented concept are used while designing the scheduler where the process has its own function and the display has its function with the input to take the no. of process in the main function.

Test Case

Test Case And Output(Waiting Time and TurnAround Time)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| PID | Arrival Time | Burstime | Waiting Time | Turn Around Time |
| P1 | 0 | 20 | 58 | 60 |
| P2 | 5 | 36 | 39 | 46 |
| P3 | 13 | 19 | 49 | 67 |
| P4 | 26 | 42 | 31 | 61 |

Github Link: https://github.com/anmolpradhan/OSCA-CSE316