**Student Name:** Amit Kumar

**Student ID:** 11808600 (A06)

**Email Address:** [akcpr3268@gmail.com](mailto:akcpr3268@gmail.com)

**GitHub Link:** https://github.com/amitcpr/Os-project.doc

Problem:

**Ques6**. Design a scheduling program that is capable of scheduling many processes that comes in at some time interval and are allocated the CPU not more that 10 time units. CPU must schedule processes having short execution time first. CPU is idle for 3 time units and does not entertain any process prior this time. Scheduler must maintain a queue that keeps the order of execution of all the processes. Compute average waiting and turnaround time.

**Ans:** We will now solve the problem by round robin Scheduling:

**What is Round-Robin Scheduling?**

➢ The name of this algorithm comes from the round-robin principle, where each person gets an equal share of something in turns. It is the oldest, simplest scheduling algorithm, which is mostly used for multitasking. In Round-robin scheduling, each ready task runs turn by turn only in a cyclic queue for a limited time slice. This algorithm also offers starvation free execution of processes.

**Algorithm:**

➢ Start the process ➢ We will now solve the problem by pre-emptive round robin Scheduling:

**What is Round-Robin Scheduling?**

➢ The name of this algorithm comes from the round-robin principle, where each person gets an equal share of something in turns. It is the oldest, simplest scheduling algorithm, which is mostly used for multitasking. In Round-robin scheduling, each ready task runs turn by turn only in a cyclic queue for a limited time slice. This algorithm also offers starvation free execution of processes.

**Algorithm:**

➢ Start the process ➢ CPU is assigned to the process on the basis of FCFS for a fixed amount of time. ➢ This fixed amount of time is called as **time quantum** or **time slice**. ➢ After the time quantum expires, the running process is pre-empted and sent to the ready

queue. ➢ Then, the processor is assigned to the next arrived process. ➢ It is always pre-emptive in nature. ➢ **average waiting** and **Turn around time**.

(a) Waiting time for process(n) = waiting time of process(n-1)+ burst time of process(n-1 )+ the time difference in getting the CPU from process(n-1)

(b) Turn around time for process(n) = waiting time of process(n) + burst time of process(n)+ the time difference in getting CPU from process(n).

➢ **Calculate:**

(a) Average waiting time = Total waiting Time / Number of process

Average Turnaround time = Total Turnaround Time / Number of process ➢ Stop the process.

PU is assigned to the process based on FCFS for a fixed amount of time. ➢ This fixed amount of time is called as **time quantum** or **time slice**. ➢ After the time quantum expires, the running process is pre-empted and sent to the ready

queue. ➢ Then, the processor is assigned to the next arrived process. ➢ It is always pre-emptive in nature. ➢ **average waiting** and **Turn around time**.

(a) Waiting time for process(n) = waiting time of process(n-1) + burst time of process(n-1 )+ the time difference in getting the CPU from process(n-1)

(b) Turn around time for process(n) = waiting time of process(n) + burst time of process(n)+ the time difference in getting CPU from process(n).

➢ **Calculate:**

(a) Average waiting time = Total waiting Time / Number of process

Average Turnaround time = Total Turnaround Time / Number of process ➢ Stop the process.

Code:

//Question no 06 Amit Kumar (A06 ) : #include <iostream>

using namespace std;

/\*

\* Round Robin Scheduling Algorithm

\*/

void roundRobin(const int n, const int q, const int BT[], int WT[], int TAT[]) {

int tempBT[n];

int totalBT = 0;

int counterBT = 0;

// Copying Burst Time to Temp Burst Time Array

// Calculating total Burst Time

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

tempBT[i] = BT[i];

totalBT += BT[i];

WT[i] = 0;

TAT[i] = 0;

}

// Calculation Loop

int i = -1;

while (true) {

// Loop ending and counter variable conditions

if (counterBT == totalBT) // End of Gantt Chart

break;

else if (i == (n - 1)) // If it's last process then again begin from first process

i = 0;

else

i += 1;

// Gantt Chart Calculation

if (tempBT[i] == 0) {

continue; // i'th burst time zero means this process has finished execution completely

} else if (tempBT[i] <= q) {

counterBT += tempBT[i];

tempBT[i] = 0;

} else if (tempBT[i] > q) {

tempBT[i] -= q;

counterBT += q;

}

// Calculating immediate Turn Around Time and Waiting Time

TAT[i] = counterBT - 0;

WT[i] = TAT[i] - BT[i];

}

//

}

/\*

\* Main Function

\*/

int main() {

int n;

cout << "Number of process:";

cin >> n;

int q;

cout << "Time quantum:";

//

cin >> q;

int BT[n];

int WT[n];

int TAT[n];

cout << endl;

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

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

cin >> BT[i];

}

//

cout << endl;

roundRobin(n, q, BT, WT, TAT); // Round Robin Scheduling Algorithm

//

int totalWT = 0;

int totalTAT = 0;

cout << "PNO\t" << "BT\t" << "AT\t" << "WT\t" << "TAT" << endl;

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

cout << (i + 1) << "\t";

cout << BT[i] << "\t";

cout << 0 << "\t";

cout << WT[i] << "\t";

cout << TAT[i] << endl;

//

totalWT += WT[i];

totalTAT += TAT[i];

}

cout << endl;

cout << "Average Waiting Time: " << ((float) totalWT / (float) n) << endl;

cout << "Average Turn Around Time: " << ((float) totalTAT / (float) n) << endl;

//

return 0;

}

// End of code

Output:

