**Operating System Lab**

**22F-3168**

**Mariam Fatima**

**Lab 9**

**Code:**

#include <iostream>

#include <algorithm>

using namespace std;

struct Process {

int process\_id;

int burst\_time;

int arrival\_time;

int waiting\_time;

int turnaround\_time;

int completion\_time;

int response\_time;

int remaining\_time; // For Round Robin

bool is\_first\_response; // To track first response time in RR

};

void inputProcesses(Process processes[], int n) {

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

cout << "Enter Process ID: ";

cin >> processes[i].process\_id;

cout << "Enter Burst Time: ";

cin >> processes[i].burst\_time;

cout << "Enter Arrival Time: ";

cin >> processes[i].arrival\_time;

processes[i].remaining\_time = processes[i].burst\_time; // Initialize remaining time for Round Robin

processes[i].is\_first\_response = true; // To capture first CPU response time in RR

}

}

void fcfsScheduling(Process processes[], int n) {

// Sort processes by arrival time

sort(processes, processes + n, [](Process a, Process b) {

return a.arrival\_time < b.arrival\_time;

});

int current\_time = 0;

int total\_waiting\_time = 0, total\_turnaround\_time = 0, total\_response\_time = 0;

cout << "\nFCFS Scheduling:\n";

cout << "Process ID | Completion Time | Turnaround Time | Waiting Time | Response Time\n";

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

if (current\_time < processes[i].arrival\_time)

current\_time = processes[i].arrival\_time;

processes[i].completion\_time = current\_time + processes[i].burst\_time;

processes[i].turnaround\_time = processes[i].completion\_time - processes[i].arrival\_time;

processes[i].waiting\_time = processes[i].turnaround\_time - processes[i].burst\_time;

processes[i].response\_time = current\_time - processes[i].arrival\_time;

current\_time += processes[i].burst\_time;

total\_waiting\_time += processes[i].waiting\_time;

total\_turnaround\_time += processes[i].turnaround\_time;

total\_response\_time += processes[i].response\_time;

cout << processes[i].process\_id << " "

<< processes[i].completion\_time << " "

<< processes[i].turnaround\_time << " "

<< processes[i].waiting\_time << " "

<< processes[i].response\_time << endl;

}

cout << "Average Waiting Time: " << (float)total\_waiting\_time / n << endl;

cout << "Average Turnaround Time: " << (float)total\_turnaround\_time / n << endl;

cout << "Average Response Time: " << (float)total\_response\_time / n << endl;

}

void roundRobinScheduling(Process processes[], int n, int quantum) {

int current\_time = 0;

int total\_waiting\_time = 0, total\_turnaround\_time = 0, total\_response\_time = 0;

int remaining\_processes = n;

cout << "\nRound Robin Scheduling (Quantum = " << quantum << "):\n";

cout << "Process ID | Completion Time | Turnaround Time | Waiting Time | Response Time\n";

while (remaining\_processes > 0) {

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

if (processes[i].remaining\_time > 0) {

if (processes[i].is\_first\_response) {

processes[i].response\_time = current\_time - processes[i].arrival\_time;

processes[i].is\_first\_response = false;}

int time\_slice = min(processes[i].remaining\_time, quantum);

processes[i].remaining\_time -= time\_slice;

current\_time += time\_slice;

if (processes[i].remaining\_time == 0) {

processes[i].completion\_time = current\_time;

processes[i].turnaround\_time = processes[i].completion\_time - processes[i].arrival\_time;

processes[i].waiting\_time = processes[i].turnaround\_time - processes[i].burst\_time;

total\_waiting\_time += processes[i].waiting\_time;

total\_turnaround\_time += processes[i].turnaround\_time;

total\_response\_time += processes[i].response\_time;

remaining\_processes--;

cout << processes[i].process\_id << " "

<< processes[i].completion\_time << " "

<< processes[i].turnaround\_time << " "

<< processes[i].waiting\_time << " "

<< processes[i].response\_time << endl;

}

}

}

}

cout << "Average Waiting Time: " << (float)total\_waiting\_time / n << endl;

cout << "Average Turnaround Time: " << (float)total\_turnaround\_time / n << endl;

cout << "Average Response Time: " << (float)total\_response\_time / n << endl;

}

int main() {

int n, quantum;

//taking input for FCFS

cout << "Enter number of processes: ";

cin >> n;

Process processes1[n];

inputProcesses(processes1, n);

// FCFS Scheduling

fcfsScheduling(processes1, n);

// Reset remaining times and response flags for Round Robin

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

processes1[i].remaining\_time = processes1[i].burst\_time;

processes1[i].is\_first\_response = true; // Reset for Round Robin

}

// Round Robin Scheduling

//taking input for round robin

cout << "Enter number of processes: ";

cin >> n;

Process processes[n];

inputProcesses(processes, n);

cout << "\nEnter Time Quantum for Round Robin: ";

cin >> quantum;

roundRobinScheduling(processes, n, quantum);

return 0;

}



