// Task 2: Implementation of CPU scheduling algorithms to find turnaround time and waiting time. a) FCFS b) SJF c) Round Robin (pre-emptive) d) Priority.

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

struct Process {

    int process\_id;

    int burst\_time;

    int waiting\_time;

    int turnaround\_time;

    int priority;

    int remaining\_burst\_time;

};

// Function to calculate waiting and turnaround times

void calculateTimes(vector<Process>& processes) {

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

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

    }

}

void printProcesses(const vector<Process>& processes, const string& scheduling\_type) {

    double total\_waiting\_time = 0;

    double total\_turnaround\_time = 0;

    cout << scheduling\_type << " Scheduling:\n";

    cout << "Processes Burst time Waiting time Turn around time\n";

    for (const auto& proc : processes) {

        cout << " " << proc.process\_id << "\t\t" << proc.burst\_time << "\t\t"

             << proc.waiting\_time << "\t\t" << proc.turnaround\_time << endl;

        total\_waiting\_time += proc.waiting\_time;

        total\_turnaround\_time += proc.turnaround\_time;

    }

    double average\_waiting\_time = total\_waiting\_time / processes.size();

    double average\_turnaround\_time = total\_turnaround\_time / processes.size();

    cout << "Average Waiting Time: " << average\_waiting\_time << endl;

    cout << "Average Turnaround Time: " << average\_turnaround\_time << "\n\n";

}

// First-Come, First-Served (FCFS)

void FCFS(vector<Process>& p) {

    p[0].waiting\_time = 0;

    for (int i=1; i<p.size(); i++) {

        p[i].waiting\_time = p[i - 1].burst\_time + p[i - 1].waiting\_time;

    }

    calculateTimes(p);

    printProcesses(p, "FCFS");

}

// Shortest Job First (SJF)

bool sjf\_comparison(const Process& a, const Process& b) {

    return (a.burst\_time < b.burst\_time);

}

void SJF(vector<Process>& processes) {

    sort(processes.begin(), processes.end(), sjf\_comparison);

    processes[0].waiting\_time = 0;

    for (int i=1; i<processes.size(); i++) {

        processes[i].waiting\_time = processes[i - 1].burst\_time + processes[i - 1].waiting\_time;

    }

    calculateTimes(processes);

    printProcesses(processes, "SJF");

}

// Round Robin (Pre-emptive)

void RoundRobin(vector<Process>& processes, int time\_quantum) {

    int time = 0;

    for (auto& proc : processes) {

        proc.remaining\_burst\_time = proc.burst\_time;

    }

    while (true) {

        bool done = true;

        for (auto& proc : processes) {

            if (proc.remaining\_burst\_time > 0) {

                done = false;

                if (proc.remaining\_burst\_time > time\_quantum) {

                    time += time\_quantum;

                    proc.remaining\_burst\_time -= time\_quantum;

                } else {

                    time += proc.remaining\_burst\_time;

                    proc.waiting\_time = time - proc.burst\_time;

                    proc.remaining\_burst\_time = 0;

                }

            }

        }

        if (done) break;

    }

    calculateTimes(processes);

    printProcesses(processes, "Round Robin");

}

// Priority Scheduling

bool priority\_comparison(const Process& a, const Process& b) {

    return (a.priority > b.priority);

}

void PriorityScheduling(vector<Process>& processes) {

    sort(processes.begin(), processes.end(), priority\_comparison);

    processes[0].waiting\_time = 0;

    for (int i=1; i<processes.size(); i++) {

        processes[i].waiting\_time = processes[i - 1].burst\_time + processes[i - 1].waiting\_time;

    }

    calculateTimes(processes);

    printProcesses(processes, "Priority");

}

int main() {

    vector<Process> processes\_fcfs = {{0, 5}, {1, 3}, {2, 8}, {3, 6}};

    FCFS(processes\_fcfs);

    vector<Process> processes\_sjf = {{1, 6}, {2, 8}, {3, 7}, {4, 3}};

    SJF(processes\_sjf);

    vector<Process> processes\_rr = {{1, 10}, {2, 5}, {3, 8}};

    int time\_quantum = 2;

    RoundRobin(processes\_rr, time\_quantum);

    vector<Process> processes\_priority = {{1, 10, 3}, {2, 1, 1}, {3, 2, 4}, {4, 1, 5}, {5, 5, 2}};

    PriorityScheduling(processes\_priority);

    return 0;

}