**Task 9**

**Description**

This program simulates and compares the turnaround times of 3 different scheduling algorithms FCFS,SJF and RR, a manually set of processes is fed into each function for equal comparison.

**Code**

#include <stdio.h>  
#include <limits.h>  
  
void FCFS(int id[], int arrival[], int burst[], int waiting[], int turnaround[], int n) {  
    waiting[0] = 0;  
  
    for (int i = 1; i < n; i++) {  
        waiting[i] = waiting[i - 1] + burst[i - 1] - (arrival[i] - arrival[i - 1]);  
        if (waiting[i] < 0) waiting[i] = 0;  
    }  
  
    for (int i = 0; i < n; i++) {  
        turnaround[i] = waiting[i] + burst[i];  
    }  
  
     
    printf("ID\tArrival\tBurst\tWaiting\tTurnaround\n");  
    for (int i = 0; i < n; i++) {  
        printf("%d\t%d\t%d\t%d\t%d\n", id[i], arrival[i], burst[i], waiting[i], turnaround[i]);  
    }  
}  
  
void SJF(int id[], int arrival[], int burst[], int waiting[], int turnaround[], int n) {  
    int completed[n]; // Track completed processes (0 = not completed, 1 = completed)  
    int currentTime = 0, completedCount = 0;  
  
     
    for (int i = 0; i < n; i++) {  
        completed[i] = 0;  
    }  
  
    while (completedCount < n) {  
        int shortest = -1, minBurst = INT\_MAX;  
  
        // Find the shortest job that has arrived and is not completed  
        for (int i = 0; i < n; i++) {  
            if (arrival[i] <= currentTime && completed[i] == 0 && burst[i] < minBurst) {  
                minBurst = burst[i];  
                shortest = i;  
            }  
        }  
  
         
        if (shortest == -1) {  
            currentTime++;  
            continue;  
        }  
  
         
        currentTime += burst[shortest]; // Move time forward  
        turnaround[shortest] = currentTime - arrival[shortest]; // TAT = Completion - Arrival  
        waiting[shortest] = turnaround[shortest] - burst[shortest]; // WT = TAT - Burst  
        completed[shortest] = 1; // Mark as completed  
        completedCount++;  
    }  
  
     
    printf("ID\tArrival\tBurst\tWaiting\tTurnaround\n");  
    for (int i = 0; i < n; i++) {  
        printf("%d\t%d\t%d\t%d\t%d\n", id[i], arrival[i], burst[i], waiting[i], turnaround[i]);  
    }  
}  
  
void RR(int id[], int arrival[], int burst[], int waiting[], int turnaround[], int n, int quantum) {  
    int remaining[n]; // Remaining burst times  
    int time = 0;     // Current time  
    int done = 0;     // Number of completed processes  
  
     
    for (int i = 0; i < n; i++) {  
        remaining[i] = burst[i];  
    }  
  
     
    while (done < n) {  
        int executed = 0;  
  
        for (int i = 0; i < n; i++) {  
            if (remaining[i] > 0 && arrival[i] <= time) { // Process has arrived and is not yet completed  
                if (remaining[i] > quantum) {  
                    time += quantum;  
                    remaining[i] -= quantum;  
                } else {  
                    time += remaining[i];  
                    waiting[i] = time - arrival[i] - burst[i]; // WT = Completion - Arrival - Burst  
                    turnaround[i] = time - arrival[i];         // TAT = Completion - Arrival  
                    remaining[i] = 0;  
                    done++;  
                }  
                executed = 1;  
            }  
        }  
         
        if (!executed) time++;  
    }  
  
    // Print Results  
    printf("ID\tArrival\tBurst\tWaiting\tTurnaround\n");  
    for (int i = 0; i < n; i++) {  
        printf("%d\t%d\t%d\t%d\t%d\n", id[i], arrival[i], burst[i], waiting[i], turnaround[i]);  
    }  
}  
  
int main() {  
    int id[] = {1, 2, 3, 4, 5};  
    int arrival[] = {0, 1, 2, 3, 4};  
    int burst[] = {5, 3, 8, 6, 2};  
    int quantum = 4;  
    int n = sizeof(id) / sizeof(id[0]); // Number of processes  
     
  
    int waiting[n], turnaround[n]; // Arrays to store waiting and turnaround times  
    printf("FCFS SCHEDULING\n");  
    FCFS(id, arrival, burst, waiting, turnaround, n);  
    printf("SJF SCHEDULING\n");  
    SJF(id, arrival, burst, waiting, turnaround, n);  
    printf("Round robin scheduling\n");  
    RR(id, arrival, burst, waiting, turnaround, n,quantum);  
    return 0;  
}

**Gitlink**

[**https://github.com/FirasAhmed2/Operating-systems-coursework.git**](https://github.com/FirasAhmed2/Operating-systems-coursework.git)