. Write a program to compute the finish time, turnaround time and waiting time for the

Priority (Preemptive and Non Preemptive)

#include <stdio.h>

#include <stdbool.h>

struct Process {

int pid; // Process ID

int arrivalTime;

int burstTime;

int remainingTime; // For preemptive scheduling

int priority;

int finishTime;

int turnAroundTime;

int waitingTime;

bool isCompleted;

};

// Function for Non-Preemptive Priority Scheduling

void priorityNonPreemptive(struct Process processes[], int n) {

int currentTime = 0, completed = 0;

while (completed < n) {

int minIndex = -1;

int highestPriority = 1e9;

// Select the process with the highest priority that has arrived

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

if (!processes[i].isCompleted && processes[i].arrivalTime <= currentTime &&

processes[i].priority < highestPriority) {

highestPriority = processes[i].priority;

minIndex = i;

}

}

if (minIndex == -1) {

currentTime++;

} else {

// Calculate the finish time, turnaround time, and waiting time

processes[minIndex].finishTime = currentTime + processes[minIndex].burstTime;

processes[minIndex].turnAroundTime = processes[minIndex].finishTime - processes[minIndex].arrivalTime;

processes[minIndex].waitingTime = processes[minIndex].turnAroundTime - processes[minIndex].burstTime;

processes[minIndex].isCompleted = true;

currentTime = processes[minIndex].finishTime;

completed++;

}

}

}

// Function for Preemptive Priority Scheduling

void priorityPreemptive(struct Process processes[], int n) {

int currentTime = 0, completed = 0;

while (completed < n) {

int minIndex = -1;

int highestPriority = 1e9;

// Select the process with the highest priority that has arrived

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

if (processes[i].arrivalTime <= currentTime && !processes[i].isCompleted &&

processes[i].priority < highestPriority) {

highestPriority = processes[i].priority;

minIndex = i;

}

}

if (minIndex != -1) {

// Process one unit of the burst time

processes[minIndex].remainingTime--;

currentTime++;

// If process is completed

if (processes[minIndex].remainingTime == 0) {

processes[minIndex].finishTime = currentTime;

processes[minIndex].turnAroundTime = processes[minIndex].finishTime - processes[minIndex].arrivalTime;

processes[minIndex].waitingTime = processes[minIndex].turnAroundTime - processes[minIndex].burstTime;

processes[minIndex].isCompleted = true;

completed++;

}

} else {

currentTime++;

}

}

}

void displayResults(struct Process processes[], int n) {

printf("PID\tArrival\tBurst\tPriority\tFinish\tTurnaround\tWaiting\n");

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

printf("%d\t%d\t%d\t%d\t\t%d\t%d\t\t%d\n",

processes[i].pid,

processes[i].arrivalTime,

processes[i].burstTime,

processes[i].priority,

processes[i].finishTime,

processes[i].turnAroundTime,

processes[i].waitingTime);

}

float totalTurnAroundTime = 0, totalWaitingTime = 0;

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

totalTurnAroundTime += processes[i].turnAroundTime;

totalWaitingTime += processes[i].waitingTime;

}

printf("Average Turnaround Time: %.2f\n", totalTurnAroundTime / n);

printf("Average Waiting Time: %.2f\n", totalWaitingTime / n);

}

int main() {

int n, choice;

printf("Enter the number of processes: ");

scanf("%d", &n);

struct Process processes[n];

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

processes[i].pid = i + 1;

printf("Enter arrival time, burst time, and priority for process %d: ", processes[i].pid);

scanf("%d %d %d", &processes[i].arrivalTime, &processes[i].burstTime, &processes[i].priority);

processes[i].remainingTime = processes[i].burstTime;

processes[i].isCompleted = false;

}

printf("Choose Scheduling:\n1. Non-Preemptive Priority\n2. Preemptive Priority\n");

scanf("%d", &choice);

if (choice == 1) {

priorityNonPreemptive(processes, n);

} else if (choice == 2) {

priorityPreemptive(processes, n);

} else {

printf("Invalid choice!\n");

return 0;

}

displayResults(processes, n);

return 0;

}