#include <stdio.h>

#include <limits.h>

struct Process {

int id;

int arrivalTime;

int burstTime;

int remainingTime;

int waitingTime;

int turnAroundTime;

int completionTime;

};

int main() {

int n;

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

scanf("%d", &n);

struct Process processes[n];

int totalWaitingTime = 0, totalTurnAroundTime = 0;

// Input Arrival and Burst Times

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

processes[i].id = i;

printf("Enter Arrival Time and Burst Time for Process P%d: ", i);

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

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

}

int completed = 0, currentTime = 0, shortest = -1;

int minRemainingTime = INT\_MAX;

// Flags to track if all processes are completed

while (completed < n) {

// Find the process with the shortest remaining time at the current time

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

if (processes[i].arrivalTime <= currentTime && processes[i].remainingTime > 0)

{

if (processes[i].remainingTime < minRemainingTime) {

minRemainingTime = processes[i].remainingTime;

shortest = i;

}

if (processes[i].remainingTime == minRemainingTime) {

// Tie-breaker: Choose the process with earlier arrival time

if (processes[i].arrivalTime < processes[shortest].arrivalTime) {

shortest = i;

}

}

}

}

if (shortest == -1) {

currentTime++; // No process available, increment time

continue;

}

// Execute the process with the shortest remaining time

processes[shortest].remainingTime--;

currentTime++;

minRemainingTime = processes[shortest].remainingTime;

// If the process is completed

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

processes[shortest].completionTime = currentTime;

processes[shortest].turnAroundTime = processes[shortest].completionTime - processes[shortest].arrivalTime;

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

totalWaitingTime += processes[shortest].waitingTime;

totalTurnAroundTime += processes[shortest].turnAroundTime;

completed++;

shortest = -1;

minRemainingTime = INT\_MAX;

}

}

// Output Results

printf("\nPROCESS\tARRIVAL TIME\tBURST TIME\tWAITING TIME\tTURNAROUND TIME\n");

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

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

processes[i].id,

processes[i].arrivalTime,

processes[i].burstTime,

processes[i].waitingTime,

processes[i].turnAroundTime);

}

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

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

return 0;

}

### **Pseudocode for SRTF Algorithm**

1. **Input**:
   * n: Number of processes.
   * For each process P[i], take input:
     + arrivalTime[i]: Arrival time of the process.
     + burstTime[i]: Burst time of the process.
2. **Initialize**:
   * Set remainingTime[i] = burstTime[i] for all processes.
   * Set waitingTime[i] = 0, turnAroundTime[i] = 0 for all processes.
   * Set currentTime = 0, completed = 0.
   * Set minRemainingTime = infinity and shortest = -1.
3. **While completed < n**:
   * **Step 3.1**: Find the process with the shortest remaining time:
     + For each process P[i]:
       - If arrivalTime[i] <= currentTime and remainingTime[i] > 0:
         * If remainingTime[i] < minRemainingTime, update:

minRemainingTime = remainingTime[i]

shortest = i.

* + - * + If remainingTime[i] == minRemainingTime:

Break ties by selecting the process with the smaller arrivalTime.

* + **Step 3.2**: Check if no process is available:
    - If shortest == -1:
      * Increment currentTime and continue.
  + **Step 3.3**: Execute the process with the shortest remaining time:
    - Decrement remainingTime[shortest] by 1.
    - Increment currentTime by 1.
  + **Step 3.4**: Check if the process is completed:
    - If remainingTime[shortest] == 0:
      * Mark the process as completed:
        + completed += 1.
        + Set completionTime[shortest] = currentTime.
        + Compute:

turnAroundTime[shortest] = completionTime[shortest] - arrivalTime[shortest].

waitingTime[shortest] = turnAroundTime[shortest] - burstTime[shortest].

* + - * Reset minRemainingTime = infinity and shortest = -1.

1. **Compute Results**:
   * Calculate:
     + totalWaitingTime = sum(waitingTime[i] for all i).
     + totalTurnAroundTime = sum(turnAroundTime[i] for all i).
   * Compute averages:
     + averageWaitingTime = totalWaitingTime / n.
     + averageTurnAroundTime = totalTurnAroundTime / n.
2. **Output Results**:
   * Print:
     + Process ID, Arrival Time, Burst Time, Waiting Time, and Turnaround Time for each process.
   * Print:
     + Average Waiting Time.
     + Average Turnaround Time.