**CISC2005 Exercise 05**

# Prerequisites

## Successful installation of \*nix terminal (e.g., WSL, cygwin) or a \*nix operating system (e.g., Linux, Mac).

For Windows system, I recommend you to install WSL. <https://learn.microsoft.com/en-us/windows/wsl/install>

## Successful installation of a C compiler, GCC is recommended.

## Successful installation of a text editor, VIM or Emacs is recommended.

# Tasks

In this section, students are required to complete the following codes to implement several basic operating system scheduling algorithms. For each program, students should capture a screenshot of the successful execution **with provided sample input**, verify that the output is the same as **sample output.** In the submission file, please attach the **added code** and **execution screenshot** in sequence.

**Q1**: Complete the code at the blank underline to implements the First-Come First-Served (FCFS) scheduling algorithm. The program reads in a set of processes, each of which includes a **Process ID**, an **Arrival Time(at)** and a **Service Time(st)**. The program should schedule the processes on a first-come, first-served basis based on arrival time and calculate the **Wait Time(wt)** and **Turnaround Time(tt)** for each process.

Input format.

The first line includes a positive integer n, indicating the number of processes. Each of the next n lines includes three positive integers indicating the process ID, arrival time, and service time, respectively.

Output format.

Outputs the **wait time** and **turnaround time** of each process in ascending order of process ID.

|  |  |
| --- | --- |
| Sample input | Sample output |
| 3 |  |
| 1 0 3 | 1 0 3 |
| 2 2 6 | 2 1 7 |
| 3 4 4 | 3 5 9 |

**Tips.**

* Turnaround time = completion time - arrival time.
* You can uncheck the "numbering" of the setting in Word to eliminate line numbers of codes and then copy codes to your text editor.

1. #include <stdio.h>
2. #include <stdlib.h>
4. **struct** process {
5. **int** pid;    // Process ID
6. **int** arrive; // Arrival Time
7. **int** service;    // Service Time
8. };
10. // Sort in ascending order of arrival time
11. **int** cmp(**const** **void** \*a, **const** **void** \*b) {
12. **struct** process \*p1 = (**struct** process \*) a;
13. **struct** process \*p2 = (**struct** process \*) b;
14. **return** p1->arrive - p2->arrive;
15. }
17. **int** main() {
18. **int** n, i;
19. printf("please enter the number of processes：");
20. scanf("%d", &n);
21. **struct** process \*proc = malloc(**sizeof**(**struct** process) \* n);
22. **int** \*wait\_time = malloc(**sizeof**(**int**) \* n);
23. **int** \*turnaround\_time = malloc(**sizeof**(**int**) \* n);
24. **int** current\_time = 0;
26. printf("please enter the info for each process, with three numbers on each line representing the process ID, arrival time, and service time(separated by Space).：\n");
27. **for** (i = 0; i < n; i++) {
28. scanf("%d %d %d", &proc[i].pid, &proc[i].arrive, &proc[i].service);
29. }
31. // Sort in ascending order of arrival time
32. qsort(proc, n, **sizeof**(**struct** process), cmp);
34. **for** (i = 0; i < n; i++) {
35. **if** (current\_time < proc[i].arrive) {
36. current\_time = proc[i].arrive;
37. }
38. wait\_time[i] = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;
39. current\_time += \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;
40. turnaround\_time[i] = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;
41. }
43. printf("The wait time and turnaround time for each process are as follows ：\n");
44. double avg\_wait\_time = 0;
45. for (i = 0; i < n; i++) {
46. avg\_wait\_time += wait\_time[i];
47. printf("%d %d %d\n", proc[i].pid, wait\_time[i], turnaround\_time[i]);
48. }
49. printf("avg. Wait Time:%f\n", avg\_wait\_time);
50. avg\_wait\_time /= n
51. free(proc);
52. free(wait\_time);
53. free(turnaround\_time);
54. **return** 0;
55. }

**Q2**: Complete the code at the blank underline to implements the Shortest Job First (SJF) scheduling algorithm and answer a questions. The program reads in a set of processes, each of which includes a **Process ID**, an **Arrival Time** and a **Service Time**. The program should schedule the processes on a Shortest Job First basis based on service time (arrival time should also be considered) and calculate the **Wait Time** and **Turnaround Time** for each process.

1. #include <stdio.h>
3. **struct** process {
4. **int** id;
5. **int** arrival\_time;
6. **int** service\_time;
7. **int** wait\_time;
8. **int** turnaround\_time;
9. **int** service\_save;
10. };
12. **void** calculate\_times(**struct** process \*p, **int** n) {
13. **int** i, j;
14. **int** current\_time = 0;
16. **for** (i = 0; i < n; i++) {
17. // Find the process with the shortest execution time among all processes whose arrival time is less than or equal to the current time
18. **int** shortest\_index = -1;
19. **int** shortest\_time = -1;
20. **for** (j = 0; j < n; j++) {
21. **if** (p[j].arrival\_time <= \_\_\_\_\_\_\_\_\_\_\_\_ && p[j].service\_time \_\_\_\_\_\_\_) {
22. **if** (shortest\_time == -1 || p[j].service\_time < shortest\_time) {
23. shortest\_index = j;
24. shortest\_time = p[j].service\_time;
25. }
26. }
27. }
29. **if** (shortest\_index == -1) {
30. current\_time++;
31. i--;
32. **continue**;
33. }
35. // Update the wait time and current time
36. p[shortest\_index].wait\_time = \_\_\_\_\_\_\_\_\_\_\_\_ - \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;
37. current\_time += \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;
38. p[shortest\_index].turnaround\_time = current\_time - p[shortest\_index].arrival\_time;
39. p[shortest\_index].service\_time = 0; //
40. }
41. }
43. **int** main() {
44. **int** i, n;
45. printf("please enter the number of processes:");
46. scanf("%d", &n);
47. **struct** process p[n];
49. // Input process information
50. **for** (i = 0; i < n; i++) {
51. printf("Please enter the arrival time and service time of process %d: ", i + 1);
52. scanf("%d%d", &p[i].arrival\_time, &p[i].service\_time);
53. p[i].service\_save = p[i].service\_time;
54. p[i].id = i + 1;
55. p[i].wait\_time = 0;
56. p[i].turnaround\_time = 0;}
58. calculate\_times(p, n);
60. // calculate average wait time.
62. **double** avg\_wait\_time = 0;
63. **for** (i = 0; i < n; i++) {
64. avg\_wait\_time += p[i].wait\_time;
65. }
66. avg\_wait\_time /= n;
68. // Ouput results
69. printf("Process\tArrival Time\tService Time\tWait Time\tTurnaround Time\n");
70. **for** (i = 0; i < n; i++) {
71. printf("P%d\t%d\t\t%d\t\t%d\t\t%d\n", p[i].id, p[i].arrival\_time, p[i].service\_save, p[i].wait\_time, p[i].turnaround\_time);
72. }
73. printf("avg. Wait Time:%f\n", avg\_wait\_time);
74. **return** 0;
75. }

|  |  |
| --- | --- |
| Sample input1(at and st) | Sample output1(wt and tt) |
| 3 |  |
| 0 3 | 0 3 |
| 2 6 | 5 11 |
| 3 4 | 0 4 |

Question: Comment out lines 29 to 33 of the code, run the program, and enter sample input2. What happens? Explain why.

|  |
| --- |
| Sample input2(at and st) |
| 3 |
| 2 3 |
| 0 1 |
| 2 2 |

## Q3: I have provided the code for Preemptive Shortest Job First(PSJF) below. You can run the programs for non-preemptive and preemptive SJF with the same inputs several times and compare the average waiting times of the two algorithms. If you're interested, you can study the code in detail.

|  |  |  |
| --- | --- | --- |
| Input(at and st) | Avg.Wait\_time(SJF) | Avg.Wait\_time(PSJF) |
| [0 1, 2 7, 3 4] | 2 | 1.33 |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

#include <stdio.h>

#include <stdlib.h>

struct process {

    int id;

    int arrival\_time;

    int service\_time;

    int wait\_time;

    int turnaround\_time;

};

void calculate\_times(struct process \*p, int n) {

    int i, j, time = 0, completed = 0;

    int \*remaining\_time = (int\*)malloc(n \* sizeof(int));

    for (i = 0; i < n; i++) {

        remaining\_time[i] = p[i].service\_time;

    }

    while (completed != n) {

        int shortest\_index = -1;

        int shortest\_time = -1;

        for (j = 0; j < n; j++) {

            if (p[j].arrival\_time <= time && remaining\_time[j] > 0) {

                if (shortest\_time == -1 || remaining\_time[j] < shortest\_time) {

                    shortest\_index = j;

                    shortest\_time = remaining\_time[j];

                }

            }

        }

        if (shortest\_index == -1) {

            time++;

            continue;

        }

        remaining\_time[shortest\_index]--;

        if (remaining\_time[shortest\_index] == 0) {

            completed++;

            int k = shortest\_index;

            p[k].wait\_time = time - p[k].arrival\_time - p[k].service\_time + 1;

            if (p[k].wait\_time < 0) {

                p[k].wait\_time = 0;

            }

            p[k].turnaround\_time = time - p[k].arrival\_time;

        }

        time++;

    }

    free(remaining\_time);

}

int main() {

    int i, n;

    printf("please enter the number of processes:");

    scanf("%d", &n);

    struct process p[n];

    for (i = 0; i < n; i++) {

        printf("Please enter the arrival time and execution time of process %d: ", i + 1);

        scanf("%d%d", &p[i].arrival\_time, &p[i].service\_time);

        p[i].id = i + 1;

        p[i].wait\_time = 0;

        p[i].turnaround\_time = 0;

    }

    calculate\_times(p, n);

    printf("Process\tArrival Time\tService Time\tWait Time\tTurnaround Time\n");

    int total\_wait\_time = 0;

    int total\_turnaround\_time = 0;

    for (i = 0; i < n; i++) {

        printf("P%d\t%d\t\t%d\t\t%d\t\t%d\n", p[i].id, p[i].arrival\_time, p[i].service\_time, p[i].wait\_time, p[i].turnaround\_time);

        total\_wait\_time += p[i].wait\_time;

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

    }

    printf("avg. Wait Time:%.2f\n", (float)total\_wait\_time/n);

    printf("avg. Turnaround Time:%.2f\n", (float)total\_turnaround\_time/n);

    return 0;

}