**CS3103 2023/2024 Sem B**

**Assignment 1**

**Instructions**

1. **Due: 2024, March 3rd (Sunday), 23:59 PM HKT**
2. Please write your answer directly on this document and submit it to Canvas as *doc*, *docx* or *pdf* files. Answers are allowed in text only. Any form of image/snapshot is not allowed.
3. Before you start, please take the time to review the course policy on academic integrity at: <https://www.cityu.edu.hk/ah/academic_honesty.htm>.

Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

SID: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Section 1 (50 marks)**: Consider the following process A, B, C, and D with arrival times, processing times (i.e., burst time), and deadlines as given in the following table. The *deadline* is the time when a process is expected to complete. (Note: Answers are allowed in text only, any form of image/snapshot is not allowed.)

|  |  |  |  |
| --- | --- | --- | --- |
| Process Name | Arrival Time | Processing Time (Burst Time) | Deadline |
| A | 0 | 9 | 13 |
| B | 1 | 2 | 12 |
| C | 4 | 7 | 14 |
| D | 5 | 3 | 18 |

**Question 1.1 (18 marks)** Compute the finish time, response time, and turnaround time for each process for the following CPU scheduling algorithms. Please answer the finish time, response time, and turnaround time of each processor under the following scheduling algorithms by completing the provided table:

1. First-Come-First-Serve (FCFS)
2. Preemptive Shortest-Job-First (SJF)
3. Round-Robin (RR) with time quantum = 1 and round order A🡪B🡪C🡪D. (We assume that if a process arrives exactly when its round starts, this process can start execution immediately.)

**Answer:**

Table: The Finish/Response/Turnaround Time of scheduling algorithms

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | A | B | C | D |
| FCFS Finish | 9 | 11 | 18 | 21 |
| FCFS Response | 0 | 8 | 7 | 13 |
| FCFS Turnaround | 9 | 10 | 14 | 16 |
| SJF Finish | 14 | 3 | 21 | 8 |
| SJF Response | 0 | 0 | 10 | 0 |
| SJF Turnaround | 14 | 2 | 17 | 3 |
| RR Finish | 20 | 4 | 21 | 13 |
| RR Response | 0 | 0 | 1 | 1 |
| RR Turnaround | 20 | 3 | 17 | 8 |

Note for students:

About preemptive SJF:

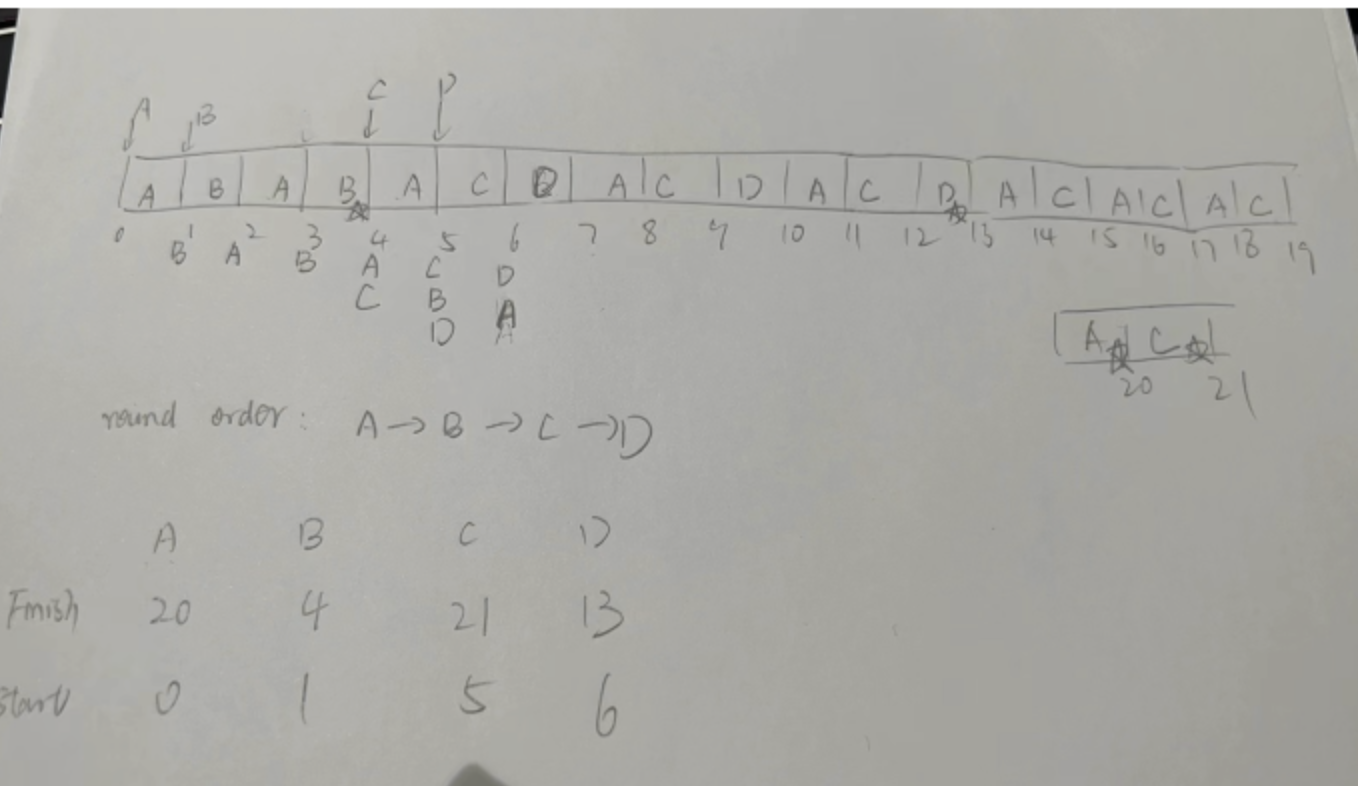
At t=4, the remaining execution times of A and C are both 7, and the correct answer should be that A continues to execute. The reason is: In the preemptive SJF scheduling policy, processes with shorter remaining time will actively preempt other processes. Since A and C have the same execution time, A will not actively give up the CPU, and C will not preempt A at this time, so A continues to execute.

About RR:

At t=1, the correct answer should choose B to execute. The reason is: the question stem gives the following condition “We assume that if a process arrives exactly when its round starts, this process can start execution immediately.”.When the running time of process A is up at a certain moment, B arrives at this time, and the ready queue is empty. At this time, the scheduler needs to decide who to schedule in the next time slice period and who to insert ready. In the queue, if this condition is not given, both A and B can use the CPU in the next time period. Given this condition, the scheduler chooses to schedule B, and A is inserted into the ready queue.

However, during the actual grading, the grading here was relatively loose, and both situations of RR were counted as correct answers. When t=1, when we choose whether to execute A or B, both situations are considered correct.

At t=4, the correct answer should choose A to execute. The reason is: According to the round robin scheduling principle, new tasks need to be placed at the end of the queue (refer to the round robin slide). At t=4, there is already A in the ready queue, and the new C can only be placed at the end of the queue. Therefore, at the moment t=4, A is to be executed.



Full-point: 18 points

* 0.5 point for each blank

**Question 1.2 (4 marks)** Is it possible to schedule these four processes so that they all meet their deadlines? If yes, please explain how to schedule them. If not, please explain why.

**Answer:**

Impossible, because the total amount of processing time is larger than the latest deadline.

Full-point: 4 points

* 2 points for answer: “impossible”
* 2 points for explanation

**Question 1.3 (6 marks)** Considering the following four algorithms: FCFS, non-preemptive SJF, preemptive SJF and RR). Please answer the following questions. (Each question may have more than one answer).

1. Which algorithm(s) has/have the maximum CPU utilization (suppose the context switch overhead can be ignored)?
2. Which algorithm(s) has/have the minimum average waiting time?
3. Which algorithms(s) is/are the fairest?

**Answer:**

* 1. All are the same (if do not consider context switching)
  2. Preemptive-SJF
  3. RR

Full-point: 6 points

* 2 points for each sub-question

**Question 1.4 (16 marks)** Suppose a process has the following four states

* Not Arrived
* Ready
* Waiting
* Running
* Finished

Under Round-Robin (RR) scheduling, what are the states of the four processes at time points 2.1, 3.4, 8.2, and 15.4 respectively? Please explain how did you get your answer.

**Answer:**

At time point 2.1:

* A: running
* B: ready
* C: not arrived
* D: not arrived

At time point 3.4:

* A: ready
* B: running
* C: not arrived
* D: not arrived

At time point 8.2:

* A: ready
* B: finished
* C: running
* D: ready

At time point 15.4:

* A: running
* B: finished
* C: ready
* D: finished

Full-point: 16 points

* Total 16 status, 1 point for each status

**Question 1.5 (6 marks)** Regarding the Round Robin algorithm mentioned in Question 1.4, during time interval [2.1, 8.6], how many times does the system switches from the user mode to the kernel mode, and how many times does the system switches from the kernel mode to the user mode? Please explain how did you get your answer.

**Answer:**

For RR, both 6 times. Because there are 6 switches among different processes. In each process switching, the system first switches from the user mode to the kernel mode, then switches from the kernel mode back to the user mode.

Full-point: 6 points

2 sub-questions: 3 points/each sub-question. For each sub-question:

* 1 point for correct times, 2 points for explanation.

**Section 2 (24 marks):** Note: Answers are allowed in text only, any form of image/snapshot is not allowed.

**Question 2.1 (12 marks)**: Run process-run.py with flags in tutorial 2. One important behavior is what to do when an I/O completes. With -I IO\_RUN\_LATER, when an I/O completes, the process that issued it is not necessarily run right away; rather, whatever was running at the time keeps running. What happens when you run this combination of processes? (-l 3:0,5:100,5:100,5:100 -S SWITCH\_ON\_IO -I IO\_RUN\_LATER -c -p) Are system resources being effectively utilized?

**Answer:**

No. The I/O job is not working until all other tasks finished, and that waste some time. (The utilization of CPU is only 66.67%.)

Full-point: 12 points

2 sub-questions: 6 points/each sub-question.

For first sub-question:

* 6 point for “what happens”

For second sub-question:

* 2 points for “not effectively utilized”
* 4 points for explanation

**Question 2.2 (12 marks)**: Now run the same processes process-run.py, but with -I IO\_RUN\_IMMEDIATE set, which immediately runs the process that issued the I/O. How does this behavior differ? Why might running a process that just completed an I/O again be a good idea?

**Answer:**

This time the utilization of CPU becomes 100%, which means we are not wasting the CPU time. System resource (CPU) is effectively utilized. Because all these three instructions issued by process 0 are I/O operations.

After issuing an I/O operation, we have to wait for it to complete. Other processes can be scheduled to run simultaneously. So it would be better if we could run the process that just completed an I/O again. It will issue another I/O instruction, then release CPU for other processes.

Full-point: 12 points

2 sub-questions: 6 points/each sub-question.

* 6 points for sub-question 1
* 6 points for sub-question 2

**Section 3 (26 marks):** Note: Answers are allowed in text only, any form of image/snapshot is not allowed.

**Question 3.1 (13 marks):** Run x86.py in tutorial 3 like this: ./x86.py -p loop.s -t 2 -i 3 -r -a dx=3,dx=3 -R dx. This makes the interrupt interval small/random; use different seeds (-s 1, -s 2, etc) to see different interleavings. Does the interrupt frequency change anything?

**Answer:**

For different seeds, the interleavings are different. But interrupt frequency does not affect the result. Because there are no shared variable accesses in this code. Both threads only change the value of register %dx. Recall that each thread when running has its own private registers; the registers are virtualized by the context-switch code that saves and restores them.

Full-point: 13 points

**Question 3.2 (13 marks):** Run with multiple iterations/threads: ./x86.py -p looping-race-nolock.s -t 2 -a bx=3 -M 2000. Why does each thread loop three times? What is final value of value?

**Answer:**

This piece of code just like a do while loop in C: do {some work; %bx--} while (%bx >0); The initial and final value of bx are 3 and 0 respectively, so the iterations is 3;

The value in memory is initialized to 0 by default, and for each thread, this value is added by 1 for 3 times. For two threads, the final value in memory is 6.

Full-point: 13 points

* 7 points for sub-question 1
* 6 points for sub-question 2