CS2106 Introduction to Operating Systems
Semester 2 2023/2024
Solutions
Tutorial 3: **Process Scheduling**

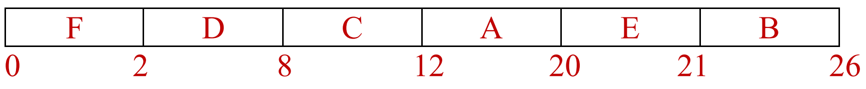
1. **[Understanding scheduling algorithm]**(Adapted from Andrew.T) Six batch jobs. *A* through *F*, arrive at a computer center at almost the same time. The estimated running time and priorities are as follows:

Process Name	**Running time (ms)**	**Priority (1 is highest priority)**
A	8	4
B	5	6
C	4	3
D	6	2
E	1	5
F	2	1

For each of the following scheduling algorithms, determine the mean process turnaround time. Ignore process switching overhead. Assume all processes are **CPU-bound and that only one job at a time runs,** until it finishes (i.e., non-preemptive).

(a) Priority scheduling.
(b) First Come First Serve (run in the order A through F)
(c) Shortest Job First

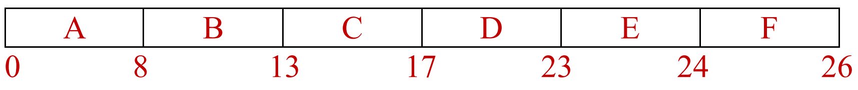
ANS:
(a) Priority scheduling : Order of process scheduling is F->D->C->A->E->B



Mean = (2+8+12+20+21+26)/6 = 14.8

111

(b) First Come First Serve : Order of process scheduling is A->B->C->D->E->F



Mean = (8+13+17+23+24+26)/6 = 18.5

(c) Shortest job first : Order of process scheduling is E->F->C->B->D->A

E	F	C	B	D	A

0 1 3 7 12 18 26

Mean = (1+3+7+12+18+26)/6 = 11.1

2. **[Walking through Scheduling Algorithms]** Consider the following execution scenario:

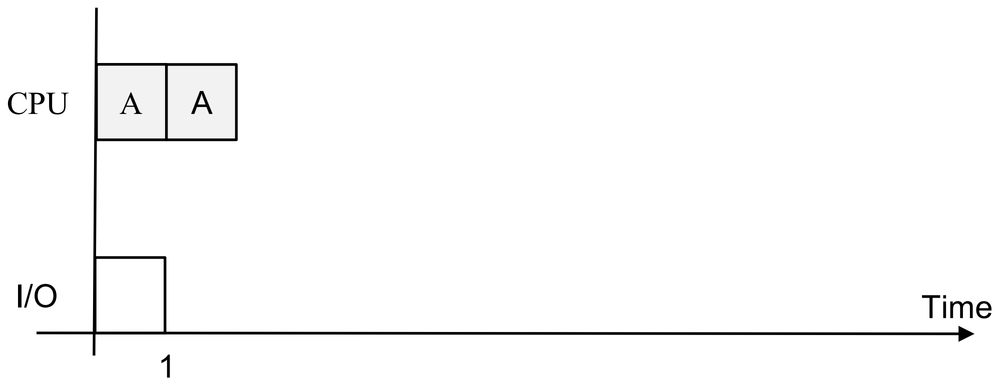
Program A, Arrives at time 0
Behavior (C**X** = Compute for **X** Time Units, IO**X** = I/O for **X** Time Units): C**3**, IO**1**, C**3**, IO**1**

Program B, Arrives at time 0
Behavior: C**1**, IO**2**, C**1**, IO**2,** C**2**, IO**1**

Program C, Arrives at time 3
Behavior: C**2**

a) Show the scheduling time chart with First-Come-First-Serve algorithm. For simplicity, we assume all tasks block on the **same I/O resource**.

Below is a sample sketch up to time 1:



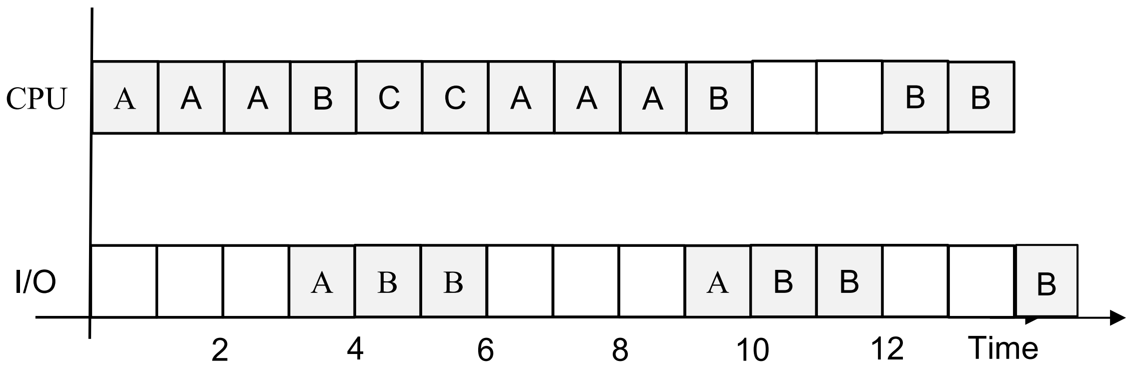
b) What are the turnaround time and the waiting time for program A, B and C? In this case, waiting time includes all time units where the program is ready for execution but could not get the CPU.

c) Use the Round **Robin** algorithm to schedule the same set of tasks. Assume a time quantum of **2 time units**. Draw out the scheduling time chart. State any assumptions you may have.

d) What is the response time for tasks A, B and C? In this case, we define response time as the time difference between the arrival time and the first time when the task receives CPU time.

ANS:

a.

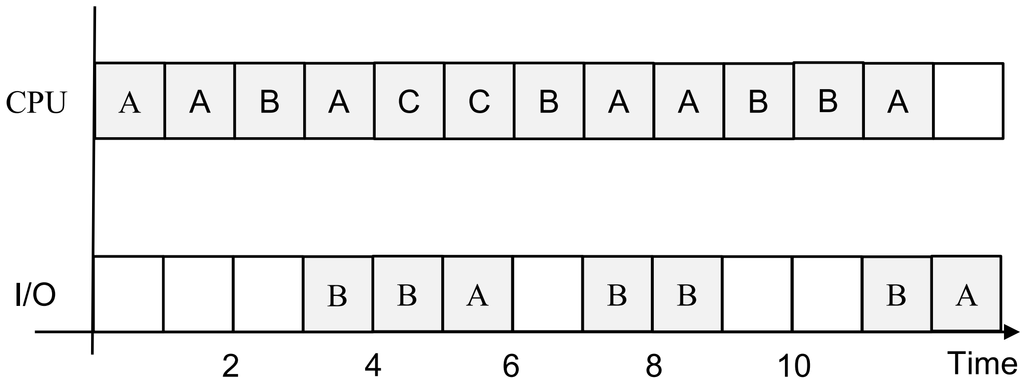


b.

	Turnaround Time	Waiting Time (turnaround time – work done- IO waiting time)
A	**10**	**10 – 8 - 0 = 2**
B	**15**	**15 – 9 - 0 = 6**
C	**6 – 3 = 3**	**3 – 2 - 0 = 1**

Note: work done here includes both CPU and IO execution time.

c.



d.

Task	Response Time
A	0
B	2 – 0 = 2
C	4 – 3 = 1

3. **[MLFQ]** As discussed in the lecture, the simple MLFQ has a few shortcomings. Describe the scheduling behavior for the following two cases.

   1. (Change of heart) A process with a lengthy CPU-intensive phase followed by I/O-intensive phase.

2. (Gaming the system) A process repeatedly gives up CPU just before the time quantum lapses.

The following are two simple tweaks. For each of the rules, identify which case (1 or 2 above) it is designed to solve, then briefly describe the new scheduling behavior.

i. (Rule – Accounting matters) The CPU usage of a process is now accumulated across time quanta. Once the CPU usage exceeds a single time quantum, the priority of the task will be decremented.

ii. (Rule – Timely boost) All processes in the system will be moved to the highest priority level periodically.

ANS:

a. The process can sink to the lowest priority during the CPU intensive phase. With the low priority, the process may not receive CPU time in a timely fashion, which degrades the responsiveness. The general shape of the timing chart is the same as the example 1 shown in lecture.

b. If a process gives up / blocks before the time quantum lapses, it will retain its priority. Since all processes enter the system with the highest priority, a process can keep its high priority indefinitely by using this trick and receive disproportionately more CPU time than other processes.

Amendments to MLFQ

i. This tweak fixes case (b). The trick in (b) works because the scheduler is "memory-less", i.e., the CPU usage is counted from fresh every time a process receives a time quantum. If the CPU usage is accumulated, then a CPU intensive process will still exhaust the allowed time quantum and be demoted in priority. This will prevent the process from hogging the CPU.

ii. This tweak is for case (a). By periodically boosting the priority of all processes (essentially treat all process as "new" and hence have highest priority), a process with different behavior phases may get a chance to be treated correctly even after it has sunk to the lowest priority.

---

4. [Adapted from AY1920S1 Midterm – Evaluating scheduling algorithms]

Briefly answer each of the following questions regarding process scheduling, stating your assumptions, if any.

a. Under what conditions does round-robin (RR) scheduling behave identically to FIFO?

b. Under what conditions does RR scheduling perform poorly compared to FIFO?

c. Under what conditions does FCFS (FIFO) scheduling result in the shortest possible average response time?

ANS:

a. RR behaves identically to FIFO if the job lengths are shorter than the time quantum, since it is essentially a pre-emptive variant of FIFO.

b. There are multiple accepted responses, depending on the criteria of evaluation for RR scheduling. Some possible responses include:
   - When the job lengths are all the same and much greater than the time quantum, RR performs poorly in average turnaround time
   - When there are many jobs and the job lengths exceed the time quantum, RR results in reduced throughput due to greater overhead from the OS incurred due to context-switches when jobs are pre-empted

c. FIFO minimizes the average response time if the jobs arrive in the ready queue in order of increasing job lengths. This avoids short jobs arriving later from waiting substantially for an earlier longer job.

Questions for your own exploration (will not be discussed in the tutorial)

5. [Putting it together] Take a look at the given mysterious program **`Behavior.c`**. This program takes in one integer command line argument **D**, which is used as a **delay** to control the amount of computation work done in the program. For the part (a) and (b), use ideas you have learned from **Lecture 4: Process Scheduling** to explain the program behavior.

   Use the command `taskset --cpu-list 0 ./Behaviors D`
   This restricts the process to run on only one core.
   Warning: you may not have the `taskset` command on your Linux system. If so, install the `util-linux` package using your package manager (apt, yum, etc).

   Note: If you are using Windows Subsystem for Linux (WSL), make sure that you are using WSL2 kernel instead of WSL1. You can check by running `wsl -l -v` and upgrade using `wsl --set-version <distro-name> 2`

   a. **D** = 1.

b. **D** = 100,000,000 (note: don’t type in the "," ☺️)

c. Now, find the **smallest D** that gives you the following interleaving output pattern:

Interleaving Output Pattern

```
[6183]: Step 0
[6184]: Step 0
[6183]: Step 1
[6184]: Step 1
[6183]: Step 2
[6184]: Step 2
[6183]: Step 3
[6184]: Step 3
[6183]: Step 4
[6184]: Step 4
[6184] Child Done!
[6183] Parent Done!
```

What do you think "**D**" represents?

*Note: "**D**" is machine dependent, you may get very different value from your friends'.*

ANS:

a. It is likely you see all steps from one process get printed before another. When the delay is very small, the total work done across the 5 iterations is less than the time quantum given for a process. Hence, the process can finish all iterations before get swapped out.
b. It is likely you see interleaving pattern similar to (c). Each iteration in DoWork() now takes (multiple) time quanta to finish. Since each process will be swapped out once the time quantum expires, the printing will be in an interleaved pattern.
   [Note to instructor: Ask what happens if we increase the D further. Ensure they see that it could be **multiple time quanta** for each iteration]
c. The amount of time to loop D times and the cost of the printing is likely to be the time quantum used on your machine. Typical time quantum value is 10ms to 100ms.

Instructors can use `taskset --cpu-list 0 sudo perf stat ./Behaviors 2000000` (specifically the sudo perf stat portion) to show students the number of context switches, just to demonstrate that the theory is in line with practice.