

# CPU Scheduling



## Practice Exercises

- 6.1 A CPU-scheduling algorithm determines an order for the execution of its scheduled processes. Given  $n$  processes to be scheduled on one processor, how many different schedules are possible? Give a formula in terms of  $n$ .

**Answer:**

$n!$  ( $n$  factorial =  $n \times n - 1 \times n - 2 \times \dots \times 2 \times 1$ ).

- 6.2 Explain the difference between preemptive and nonpreemptive scheduling.

**Answer:**

Preemptive scheduling allows a process to be interrupted in the midst of its execution, taking the CPU away and allocating it to another process. Nonpreemptive scheduling ensures that a process relinquishes control of the CPU only when it finishes with its current CPU burst.

- 6.3 Suppose that the following processes arrive for execution at the times indicated. Each process will run for the amount of time listed. In answering the questions, use nonpreemptive scheduling, and base all decisions on the information you have at the time the decision must be made.

<u>Process</u>	<u>Arrival Time</u>	<u>Burst Time</u>
$P_1$	0.0	8
$P_2$	0.4	4
$P_3$	1.0	1

- What is the average turnaround time for these processes with the FCFS scheduling algorithm?
- What is the average turnaround time for these processes with the SJF scheduling algorithm?
- The SJF algorithm is supposed to improve performance, but notice that we chose to run process  $P_1$  at time 0 because we did not know

that two shorter processes would arrive soon. Compute what the average turnaround time will be if the CPU is left idle for the first 1 unit and then SJF scheduling is used. Remember that processes  $P_1$  and  $P_2$  are waiting during this idle time, so their waiting time may increase. This algorithm could be known as future-knowledge scheduling.

**Answer:**

- a. 10.53
- b. 9.53
- c. 6.86

Remember that turnaround time is finishing time minus arrival time, so you have to subtract the arrival times to compute the turnaround times. FCFS is 11 if you forget to subtract arrival time.

- 6.4 What advantage is there in having different time-quantum sizes at different levels of a multilevel queueing system?

**Answer:**

Processes that need more frequent servicing, for instance, interactive processes such as editors, can be in a queue with a small time quantum. Processes with no need for frequent servicing can be in a queue with a larger quantum, requiring fewer context switches to complete the processing, and thus making more efficient use of the computer.

- 6.5 Many CPU-scheduling algorithms are parameterized. For example, the RR algorithm requires a parameter to indicate the time slice. Multilevel feedback queues require parameters to define the number of queues, the scheduling algorithms for each queue, the criteria used to move processes between queues, and so on.

These algorithms are thus really sets of algorithms (for example, the set of RR algorithms for all time slices, and so on). One set of algorithms may include another (for example, the FCFS algorithm is the RR algorithm with an infinite time quantum). What (if any) relation holds between the following pairs of algorithm sets?

- a. Priority and SJF
- b. Multilevel feedback queues and FCFS
- c. Priority and FCFS
- d. RR and SJF

**Answer:**

- a. The shortest job has the highest priority.
- b. The lowest level of MLFQ is FCFS.
- c. FCFS gives the highest priority to the job having been in existence the longest.
- d. None.

- 6.6 Suppose that a scheduling algorithm (at the level of short-term CPU scheduling) favors those processes that have used the least processor time in the recent past. Why will this algorithm favor I/O-bound programs and yet not permanently starve CPU-bound programs?

**Answer:**

It will favor the I/O-bound programs because of the relatively short CPU burst request by them; however, the CPU-bound programs will not starve because the I/O-bound programs will relinquish the CPU relatively often to do their I/O.

- 6.7 Distinguish between PCS and SCS scheduling.

**Answer:**

PCS scheduling is done local to the process. It is how the thread library schedules threads onto available LWPs. SCS scheduling is the situation where the operating system schedules kernel threads. On systems using either many-to-one or many-to-many, the two scheduling models are fundamentally different. On systems using one-to-one, PCS and SCS are the same.

- 6.8 Assume that an operating system maps user-level threads to the kernel using the many-to-many model and that the mapping is done through the use of LWPs. Furthermore, the system allows program developers to create real-time threads. Is it necessary to bind a real-time thread to an LWP?

**Answer:**

Yes, otherwise a user thread may have to compete for an available LWP prior to being actually scheduled. By binding the user thread to an LWP, there is no latency while waiting for an available LWP; the real-time user thread can be scheduled immediately.

- 6.9 The traditional UNIX scheduler enforces an inverse relationship between priority numbers and priorities: the higher the number, the lower the priority. The scheduler recalculates process priorities once per second using the following function:

$$\text{Priority} = (\text{recent CPU usage} / 2) + \text{base}$$

where  $\text{base} = 60$  and *recent CPU usage* refers to a value indicating how often a process has used the CPU since priorities were last recalculated.

Assume that recent CPU usage for process  $P_1$  is 40, for process  $P_2$  is 18, and for process  $P_3$  is 10. What will be the new priorities for these three processes when priorities are recalculated? Based on this information, does the traditional UNIX scheduler raise or lower the relative priority of a CPU-bound process?

**Answer:**

The priorities assigned to the processes are 80, 69, and 65 respectively. The scheduler lowers the relative priority of CPU-bound processes.

