CS 390
Chapter 6 Homework Solutions

6.1 `A CPU-scheduling algorithm determines ...`

Think of an ordered list with n elements in which we are to place n processes. The first element could be filled by any of the n processes. The second element could be filled by any of the remaining $n - 1$ processes. The third element by any of the remaining $n - 2$ processes, and so on. If we think of the each possible combination of processes in the list as a schedule, there are $n!$ schedules possible.

6.2 `Explain the difference between preemptive ...`

In non-preemptive scheduling, the scheduler only runs when a process voluntarily relinquishes the CPU (either by performing a system call or terminating). In preemptive scheduling, scheduling occurs at both of these times, as well as during interrupt service.

6.10 `Why is it important for ...`

Many scheduling algorithms require the scheduler to make an estimate of the length of a process's next CPU burst. At a minimum, this estimate should take into account the past behavior of the process. A crude estimate of this past behavior can be obtained if we can distinguish between I/O-bound and CPU-bound processes.

In addition, if we schedule I/O-bound processes before CPU-bound processes, interactive programs will show better response times, and we can use the time between user-responses to schedule CPU-bound processes.

6.11 `Discuss how the following pairs ...`

a. `CPU utilization and response time`

If we want to minimize response time, the scheduler needs to favor interactive processes (which tend to be IO-bound) over CPU-bound jobs. Because IO-bound jobs tend to have smaller CPU bursts, the scheduler will run more often, resulting in a lower CPU utilization, since the CPU will be running kernel code (and not user programs) more often. Note, however, that if

there are not enough CPU-bound jobs to keep the processor busy, CPU utilization will be low no matter how we schedule.

If we want to maximize CPU utilization, the scheduler should favor CPU-bound jobs over IO-bound jobs. This would increase response time.

b. `Average turnaround time and maximum waiting time`

In this question, our scheduling algorithm is interested is setting an upper limit on the waiting time of any process. That means we set an upper limit on the amount of time each process spends in the ready and wait queues.

If we wish to minimize average turnaround time, we should schedule short jobs before longer jobs. This means that longer jobs must sit in the ready queue, perhaps exceeding the maximum waiting time.

On the other hand, if we use the CPU timer to ensure that every job in the ready queue gets the CPU at least once in, say, the next 10 ms, then some short jobs will be pulled off the CPU and placed back on the ready queue, when they otherwise could have finished with more CPU time. This increases average turnaround time.

c. `I/O device utilization and CPU utilization`

If we want to maximize the utilization of some device, we need to make sure that there is always a process on the wait queue for the device. (Recall that each hardware device has a queue of processes that are waiting on the device. If we consider the CPU to be just another hardware device, then the ready queue is "the wait queue for the CPU device.") To maximize I/O device utilization, we need to schedule I/O bound jobs before CPU-bound jobs. However, whenever the system has all the jobs on the wait queue, CPU utilization will drop since there is no work for the CPU to do.

On the other hand, we might want to favor CPU utilization. To do so, the scheduler might always choose to schedule CPU-bound jobs before I/O-bound jobs. This will cause I/O-bound jobs to stack up in the ready queue, and cause the device queues to empty, reducing I/O device utilization.

Bonus question: Can you think of a type of computer system where favoring I/O device utilization over CPU utilization would be a desirable policy?

6.14 Consider the exponential average formula ...

a. $\alpha = 0$ and $\tau_0 = 100$ milliseconds

Setting the history parameter to 0 means that the most recent CPU-burst length will not effect our estimate of any future CPU-burst length. Thus, future estimates will depend solely on our past estimate. Since our initial estimate was 100 msecs, each of the future estimates will also be 100 msecs.

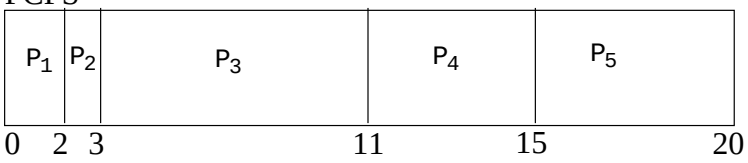
b. $\alpha = 0.99$ and $\tau_0 = 10$ milliseconds

Setting the history parameter to 0.99 means that our estimate of the next CPU-burst will be weighted very heavily by the length of the last CPU burst. In CPU-bound processes, this results in very accurate estimates, except when the process transits from a CPU-burst to an I/O burst or vice versa. In these cases, our estimate will be poor for a long time.

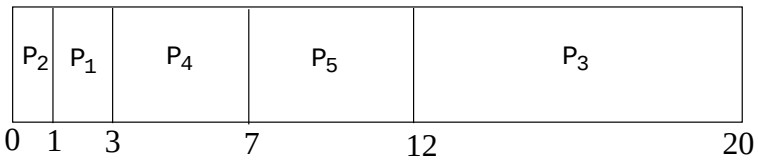
6.16 Consider the following set of ...

a. Draw four Gantt charts that ...

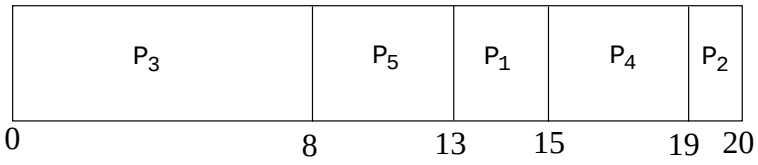
FCFS



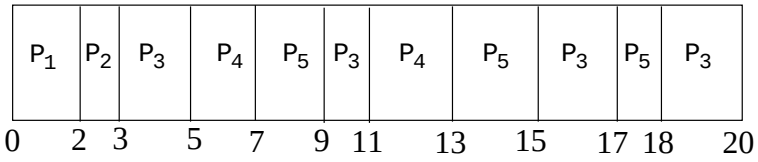
SJF



Priority



RR (Q = 2)



b. What is the turnaround time ...

Turnaround Time				
	Scheduling Algorithm			
Process	FCFS	SJF	Priority	RR
1	2	3	15	2
2	3	1	20	3
3	11	20	8	20
4	15	7	19	13
5	20	12	13	18

c. What is the waiting time ...

Waiting Time				
	Scheduling Algorithm			
Process	FCFS	SJF	Priority	RR (Q = 2)
1	0	1	13	0
2	2	0	19	2
3	3	12	0	12
4	11	3	15	9
5	15	7	8	13

d. `Which of the algorithms in ...`

- FCFS: 6.2 msecs average waiting time
- SJF: 4.6 msecs average waiting time
- Priority: 11 msecs average waiting time
- RR: 7.2 msecs average waiting time

Thus, SJF results in the minimum average waiting time, as would be expected since SJF is provably optimal when we wish to minimize waiting time and turnaround time.

6.19 `Which of the following scheduling ...`

a. `First-come, first-served`

No starvation

b. `Shortest job first`

Could result in starvation

c. `Round robin`

No starvation

d. `Priority`

Could result in starvation

6.23 `Consider a preemptive priority scheduling ...`

a. `What is the algorithm that results from` $\beta > \alpha > 0$`?`

Since $\beta > \alpha > 0$, the priorities of all processes in the system (running and on the ready queue) will increase for as long as they remain in the system. Processes that arrive to the system later will have lower priorities than those that arrive sooner. In

addition, running processes will always have higher priorities than ready processes. In this case, processes that have been in the system longest will have the highest priorities, and the running process will have the highest priority of all. This is FCFS.

b. What is the algorithm that results from $\alpha < \beta < 0$?

Since $\alpha < \beta$, processes on the ready queue will have lower priorities than the running process, and any process that enters the ready queue will have a higher priority than the running process. Thus, processes that arrive later will have a higher priority than processes that arrived earlier. The scheduling algorithm will treat the ready queue like a stack.

6.24 Explain the differences in how ...

a. FCFS

No benefit or penalty accrues to short processes.

b. RR

Short processes will finish before longer processes.

c. Multilevel feedback queues

It depends on the exact algorithm used to move processes between queues, but in general, short processes will finish before longer processes. If the scheduling algorithm tends to move IO-bound processes to higher priority queues (a reasonable assumption), then short IO-bound processes will be favored over short CPU-bound processes, both of which will be favored over longer processes.