CPU Scheduling

Practice Exercises

5.1 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 = $n \times n-1 \times n-2 \times \ldots \times 2 \times 1$).

5.2 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.

5.3 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.

Process	Arrival Time	Burst Time
P_1	0.0	8
P_2	0.4	4
P_3	1.0	1

a. What is the average turnaround time for these processes with the FCFS scheduling algorithm?

b. What is the average turnaround time for these processes with the SJF scheduling algorithm?

c. 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.

5.4 Consider the following set of processes, with the length of the CPU burst time given in milliseconds:

Process	Burst Time	Priority
P_1	2	2
P_2	1	1
P_3	8	4
P_4	4	2
P_5	5	3

The processes are assumed to have arrived in the order P_1, P_2, P_3, P_4, P_5, all at time 0.

a. Draw four Gantt charts that illustrate the execution of these processes using the following scheduling algorithms: FCFS, SJF, nonpreemptive priority (a larger priority number implies a higher priority), and RR (quantum = 2).

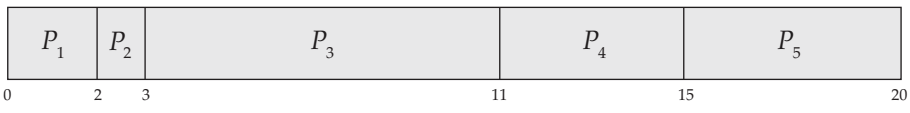
b. What is the turnaround time of each process for each of the scheduling algorithms in part a?

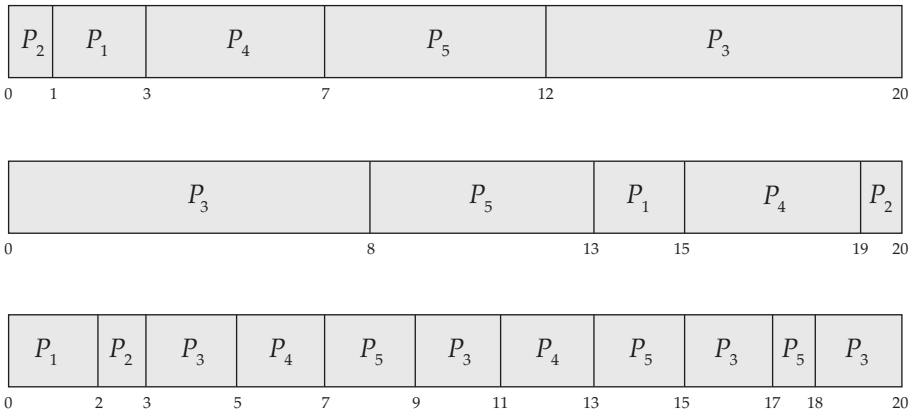
c. What is the waiting time of each process for each of these scheduling algorithms?

d. Which of the algorithms results in the minimum average waiting time (over all processes)?

Answer:

a. The four Gantt charts:





b. Turnaround time:

	FCFS	SJF	Priority	RR
P_1	2	3	15	2
P_2	3	1	20	3
P_3	11	20	8	20
P_4	15	7	19	13
P_5	20	12	13	18

c. Waiting time (turnaround time minus burst time):

	FCFS	SJF	Priority	RR
P_1	0	1	13	0
P_2	2	0	19	2
P_3	3	12	0	12
P_4	11	3	15	9
P_5	15	7	8	13

d. SJF has the shortest wait time.

5.5 The following processes are being scheduled using a preemptive, round-robin scheduling algorithm.

Process	Priority	Burst	Arrival
P_1	40	20	0
P_2	30	25	25
P_3	30	25	30
P_4	35	15	60
P_5	5	10	100
P_6	10	10	105

Each process is assigned a numerical priority, with a higher number indicating a higher relative priority. In addition to the processes listed below, the system also has an **idle task** (which consumes no CPU resources and

is identified as P_{idle}). This task has priority 0 and is scheduled whenever the system has no other available processes to run. The length of a time quantum is 10 units. If a process is preempted by a higher-priority process, the preempted process is placed at the end of the queue.

a. Show the scheduling order of the processes using a Gantt chart.

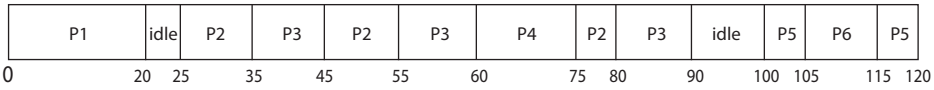
b. What is the turnaround time for each process?

c. What is the waiting time for each process?

d. What is the CPU utilization rate?

Answer:

a. The Gantt chart:



b. P1: 20-0 - 20, P2: 80-25 = 55, P3: 90 - 30 = 60, P4: 75-60 = 15, P5: 120-100 = 20, P6: 115-105 = 10

c. P1: 0, p2: 40, P3: 35, P4: 0, P5: 10, P6: 0

d. 105/120 = 87.5 percent.

5.6 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.

5.7 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 that has been in existence the longest.

d. None.

5.8 Suppose that a CPU scheduling algorithm 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 bursts requested 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.

5.9 Distinguish between PCS and SCS scheduling.

Answer:

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

5.10 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:

Priority = (recent CPU usage / 2) + base

where 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 will be 80, 69, and 65, respectively. The scheduler lowers the relative priority of CPU-bound processes.

Exercises

5.11 Of these two types of programs:

a. I/O-bound

b. CPU-bound

which is more likely to have voluntary context switches, and which is more likely to have nonvoluntary context switches? Explain your answer.

Answer:

a. I/O-bound—Voluntary, as the program is more likely to voluntarily relinquish the CPU as it waits for I/O to become available.

b. CPU-bound—Nonvoluntary, as the program is likely to use the CPU for its entire time quantum.

5.12 One technique for implementing **lottery scheduling** works by assigning processes lottery tickets, which are used for allocating CPU time. Whenever a scheduling decision has to be made, a lottery ticket is chosen at random, and the process holding that ticket gets the CPU. The BTV operating system implements lottery scheduling by holding a lottery 50 times each second, with each lottery winner getting 20 milliseconds of CPU time (20 milliseconds × 50 = 1 second). Describe how the BTV scheduler can ensure that higher-priority threads receive more attention from the CPU than lower-priority threads.

Answer:
By assigning more lottery tickets to higher-priority processes.

5.13 Consider the exponential average formula used to predict the length of the next CPU burst. What are the implications of assigning the following values to the parameters used by the algorithm?

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

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

Answer:
When $\alpha = 0$ and $\tau_0 = 100$ milliseconds, the formula always makes a prediction of 100 milliseconds for the next CPU burst. When $\alpha = 0.99$ and $\tau_0 = 10$ milliseconds, the most recent behavior of the process is given much more weight than the past history associated with the process. Consequently, the scheduling algorithm is almost memoryless and simply predicts the length of the previous burst for the next quantum of CPU execution.

5.14 A variation of the round-robin scheduler is the **regressive round-robin** scheduler. This scheduler assigns each process a time quantum and a priority. The initial value of a time quantum is 50 milliseconds. However, every time a process has been allocated the CPU and uses its entire time quantum (does not block for I/O), 10 milliseconds is added to its time quantum, and its priority level is boosted. (The time quantum for a

process can be increased to a maximum of 100 milliseconds.) When a process blocks before using its entire time quantum, its time quantum is reduced by 5 milliseconds, but its priority remains the same. What type of process (CPU-bound or I/O-bound) does the regressive round-robin scheduler favor? Explain.

Answer:
This scheduler would favor CPU-bound processes, as they are rewarded with a longer time quantum as well as a priority boost whenever they consume an entire time quantum. This scheduler does not penalize I/O-bound processes, as they are likely to block for I/O before consuming their entire time quantum, but their priority remains the same.

5.15 The following processes are being scheduled using a preemptive, priority-based, round-robin scheduling algorithm.

Process	Priority	Burst	Arrival
P_1	8	15	0
P_2	3	20	0
P_3	4	20	20
P_4	4	20	25
P_5	5	5	45
P_6	5	15	55

Each process is assigned a numerical priority, with a higher number indicating a higher relative priority. The scheduler will execute the highest-priority process. For processes with the same priority, a round-robin scheduler will be used with a time quantum of 10 units. If a process is preempted by a higher-priority process, the preempted process is placed at the end of the queue.

a. Show the scheduling order of the processes using a Gantt chart.

b. What is the turnaround time for each process?

c. What is the waiting time for each process?

Answer:

a. No Answer

b. P1 = 15, P2 = 95, P3 = 55, P4 = 55, P5 = 5, P6 = 15

c. P1 = 0, P2 = 75, P3 = 35, P4 = 35, P5 = 0, P6 = 0

5.16 Which of the following scheduling algorithms could result in starvation?

a. First-come, first-served

b. Shortest job first

c. Round robin

d. Priority

Answer:
Shortest-job-first and priority-based scheduling algorithms could result in starvation.

5.17 Consider a system running ten I/O-bound tasks and one CPU-bound task. Assume that the I/O-bound tasks issue an I/O operation once for every millisecond of CPU computing and that each I/O operation takes 10 milliseconds to complete. Also assume that the context-switching overhead is 0.1 millisecond and that all processes are long-running tasks. Describe the CPU utilization for a round-robin scheduler when:

a. The time quantum is 1 millisecond

b. The time quantum is 10 milliseconds

Answer:

a. The time quantum is 1 millisecond: Irrespective of which process is scheduled, the scheduler incurs a 0.1 millisecond context-switching cost for every context switch. This results in a CPU utilization of 1/1.1 * 100 = 91%.

b. The time quantum is 10 milliseconds: The I/O-bound tasks incur a context switch after using up only 1 millisecond of the time quantum. The time required to cycle through all the processes is therefore 10*1.1 + 10.1 (as each I/O-bound task executes for 1 millisecond and then incurs a context switch, whereas the CPU-bound task executes for 10 milliseconds before incurring a context switch). The CPU utilization is therefore 20/21.1 * 100 = 94%.

5.18 Explain the how the following scheduling algorithms discriminate either in favor of or against short processes:

a. FCFS

b. RR

c. Multilevel feedback queues

Answer:

a. FCFS—discriminates against short jobs, since any short job arriving after a long job will have a longer waiting time.

b. RR—treats all jobs equally (giving them equal bursts of CPU time), so short jobs will be able to leave the system faster, since they will finish first.

c. Multilevel feedback queues work similarly to the RR algorithm—they discriminate favorably toward short jobs.

5.19 Assume that an SMP system has private, per-processor run queues. When a new process is created, it can be placed in either the same queue as the parent process or a separate queue.

a. What are the benefits of placing the new process in the same queue as its parent?

b. What are the benefits of placing the new process in a different queue?

Answer:

a. If the child process accesses the same memory as its parent, it may be able to take advantage of the contents of the processor's cache memory.

b. The new process could be placed in a queue with a small workload, thereby providing load balancing as new processes are created.

5.20 Using the Windows scheduling algorithm, determine the numeric priority of each of the following threads.

a. A thread in the REALTIME_PRIORITY_CLASS with a relative priority of NORMAL

b. A thread in the ABOVE_NORMAL_PRIORITY_CLASS with a relative priority of HIGHEST

c. A thread in the BELOW_NORMAL_PRIORITY_CLASS with a relative priority of ABOVE_NORMAL

Answer:

a. 26

b. 8

c. 14

5.21 Assume that two tasks, *A* and *B*, are running on a Linux system. The nice values of *A* and *B* are −5 and +5, respectively. Using the CFS scheduler as a guide, describe how the respective values of `vruntime` vary between the two processes given each of the following scenarios:

- Both *A* and *B* are CPU-bound.
- *A* is I/O-bound, and *B* is CPU-bound.
- *A* is CPU-bound, and *B* is I/O-bound.

Answer:

- Since *A* has a higher priority than *B*, `vruntime` will move more slowly for *A* than for *B*. If both *A* and *B* are CPU-bound (that is, both use the CPU for as long as it is allocated to them), `vruntime` will generally be smaller for *A* than for *B*, and hence *A* will have a higher priority to run than *B*.

- In this situation, `vruntime` will be much smaller for *A* than for *B*, as (1) `vruntime` will move more slowly for *A* than for *B* due to priority differences, and (2) *A* will require less CPU time, as it is I/O-bound.
- This situation is not as clear, and it is possible that *B* may end up running in favor of *A*, as it will be using the processor less than *A* and its value of `vruntime` may, in fact, be less than the value of `vruntime` for *A*.

5.22 Explain why interrupt and dispatch latency times must be bounded in a hard real-time system.

Answer: Stopping one process and starting another requires the following tasks: save the currently executing instruction, determine the type of interrupt, save the current process state, and then invoke the appropriate interrupt service routine. Dispatch latency is the cost associated with stopping one process and starting another. Both interrupt and dispatch latency times need to be minimized in order to ensure that real-time tasks receive immediate attention. Furthermore, sometimes interrupts are disabled when kernel data structures are being modified, so an interrupt is not serviced immediately. For hard real-time systems, the time period for which interrupts are disabled must be bounded in order to guarantee the desired quality of service.