The FCFS algorithm is nonpreemptive; the RR algorithm is preemptive. The SJF and priority algorithms may be either preemptive or nonpreemptive.

Multilevel queue algorithms allow different algorithms to be used for different classes of processes. The most common model includes a foreground interactive queue that uses RR scheduling and a background batch queue that uses FCFS scheduling. Multilevel feedback queues allow processes to move from one queue to another.

Many contemporary computer systems support multiple processors and allow each processor to schedule itself independently. Typically, each processor maintains its own private queue of processes (or threads), all of which are available to run. Additional issues related to multiprocessor scheduling include processor affinity, load balancing, and multicore processing.

A real-time computer system requires that results arrive within a deadline period; results arriving after the deadline has passed are useless. Hard real-time systems must guarantee that real-time tasks are serviced within their deadline periods. Soft real-time systems are less restrictive, assigning real-time tasks higher scheduling priority than other tasks.

Real-time scheduling algorithms include rate-monotonic and earliest-deadline-first scheduling. Rate-monotonic scheduling assigns tasks that require the CPU more often a higher priority than tasks that require the CPU less often. Earliest-deadline-first scheduling assigns priority according to upcoming deadlines—the earlier the deadline, the higher the priority. Proportional share scheduling divides up processor time into shares and assigning each process a number of shares, thus guaranteeing each process a proportional share of CPU time. The POSIX Pthread API provides various features for scheduling real-time threads as well.

Operating systems supporting threads at the kernel level must schedule threads—not processes—for execution. This is the case with Solaris and Windows. Both of these systems schedule threads using preemptive, priority-based scheduling algorithms, including support for real-time threads. The Linux process scheduler uses a priority-based algorithm with real-time support as well. The scheduling algorithms for these three operating systems typically favor interactive over CPU-bound processes.

The wide variety of scheduling algorithms demands that we have methods to select among algorithms. Analytic methods use mathematical analysis to determine the performance of an algorithm. Simulation methods determine performance by imitating the scheduling algorithm on a “representative” sample of processes and computing the resulting performance. However, simulation can at best provide an approximation of actual system performance. The only reliable technique for evaluating a scheduling algorithm is to implement the algorithm on an actual system and monitor its performance in a “real-world” environment.

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 $n$. 
6.2 Explain the difference between preemptive and nonpreemptive scheduling.

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.

<table>
<thead>
<tr>
<th>Process</th>
<th>Arrival Time</th>
<th>Burst Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>P₁</td>
<td>0.0</td>
<td>8</td>
</tr>
<tr>
<td>P₂</td>
<td>0.4</td>
<td>4</td>
</tr>
<tr>
<td>P₃</td>
<td>1.0</td>
<td>1</td>
</tr>
</tbody>
</table>

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 P₁ 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 P₁ and P₂ are waiting during this idle time, so their waiting time may increase. This algorithm could be called future-knowledge scheduling.

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

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 algorithm 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

6.6 Suppose that a scheduling algorithm (at the level of short-term CPU scheduling) favors those processes that have used the least processor
6.7 Distinguish between PCS and SCS scheduling

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?

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 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 is 40 for process $P_1$, 18 for process $P_2$, and 10 for process $P_3$. 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?

### Exercises

6.10 Why is it important for the scheduler to distinguish I/O-bound programs from CPU-bound programs?

6.11 Discuss how the following pairs of scheduling criteria conflict in certain settings.

   a. CPU utilization and response time
   b. Average turnaround time and maximum waiting time
   c. I/O device utilization and CPU utilization

6.12 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 BTW 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 $\times$ 50 $= 1$ second). Describe how the BTW scheduler can ensure that higher-priority threads receive more attention from the CPU than lower-priority threads.

6.13 In Chapter 5, we discussed possible race conditions on various kernel data structures. Most scheduling algorithms maintain a run queue, which lists processes eligible to run on a processor. On multicore systems, there are two general options: (1) each processing core has its own run
queue, or (2) a single run queue is shared by all processing cores. What are the advantages and disadvantages of each of these approaches?

6.14 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

6.15 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.

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

<table>
<thead>
<tr>
<th>Process</th>
<th>Burst Time</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>$P_2$</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$P_3$</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>$P_4$</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>$P_5$</td>
<td>5</td>
<td>3</td>
</tr>
</tbody>
</table>

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)?

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

<table>
<thead>
<tr>
<th>Thread</th>
<th>Priority</th>
<th>Burst</th>
<th>Arrival</th>
</tr>
</thead>
<tbody>
<tr>
<td>( P_1 )</td>
<td>40</td>
<td>20</td>
<td>0</td>
</tr>
<tr>
<td>( P_2 )</td>
<td>30</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>( P_3 )</td>
<td>30</td>
<td>25</td>
<td>30</td>
</tr>
<tr>
<td>( P_4 )</td>
<td>35</td>
<td>15</td>
<td>60</td>
</tr>
<tr>
<td>( P_5 )</td>
<td>5</td>
<td>10</td>
<td>100</td>
</tr>
<tr>
<td>( P_6 )</td>
<td>10</td>
<td>10</td>
<td>105</td>
</tr>
</tbody>
</table>

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?

6.18 The `nice` command is used to set the nice value of a process on Linux, as well as on other UNIX systems. Explain why some systems may allow any user to assign a process a nice value \( \geq 0 \) yet allow only the root user to assign nice values \( < 0 \).

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

a. First-come, first-served
b. Shortest job first
c. Round robin
d. Priority

6.20 Consider a variant of the RR scheduling algorithm in which the entries in the ready queue are pointers to the PCBs.

a. What would be the effect of putting two pointers to the same process in the ready queue?

b. What would be two major advantages and two disadvantages of this scheme?

c. How would you modify the basic RR algorithm to achieve the same effect without the duplicate pointers?

6.21 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:
6.22 Consider a system implementing multilevel queue scheduling. What strategy can a computer user employ to maximize the amount of CPU time allocated to the user's process?

6.23 Consider a preemptive priority scheduling algorithm based on dynamically changing priorities. Larger priority numbers imply higher priority. When a process is waiting for the CPU (in the ready queue, but not running), its priority changes at a rate $\alpha$. When it is running, its priority changes at a rate $\beta$. All processes are given a priority of 0 when they enter the ready queue. The parameters $\alpha$ and $\beta$ can be set to give many different scheduling algorithms.

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

6.24 Explain the differences in how much the following scheduling algorithms discriminate in favor of short processes:

a. FCFS
b. RR
c. Multilevel feedback queues

6.25 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

6.26 Assuming that no threads belong to the REALTIME_PRIORITY_CLASS and that none may be assigned a TIME_CRITICAL priority, what combination of priority class and priority corresponds to the highest possible relative priority in Windows scheduling?

6.27 Consider the scheduling algorithm in the Solaris operating system for time-sharing threads.

a. What is the time quantum (in milliseconds) for a thread with priority 15? With priority 40?
b. Assume that a thread with priority 50 has used its entire time quantum without blocking. What new priority will the scheduler assign this thread?
c. Assume that a thread with priority 20 blocks for I/O before its time quantum has expired. What new priority will the scheduler assign this thread?
6.28 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 $\text{vrun.time}$ 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.

6.29 Discuss ways in which the priority inversion problem could be addressed in a real-time system. Also discuss whether the solutions could be implemented within the context of a proportional share scheduler.

6.30 Under what circumstances is rate-monotonic scheduling inferior to earliest-deadline-first scheduling in meeting the deadlines associated with processes?

6.31 Consider two processes, $P_1$ and $P_2$, where $p_1 = 50$, $t_1 = 25$, $p_2 = 75$, and $t_2 = 30$.

a. Can these two processes be scheduled using rate-monotonic scheduling? Illustrate your answer using a Gantt chart such as the ones in Figure 6.16–Figure 6.19.

b. Illustrate the scheduling of these two processes using earliest-deadline-first (EDF) scheduling.

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

**Bibliographical Notes**

Feedback queues were originally implemented on the CTSS system described in [Corbato et al. (1962)]. This feedback queue scheduling system was analyzed by [Schrage (1967)]. The preemptive priority scheduling algorithm of Exercise 6.23 was suggested by [Kleinrock (1975)]. The scheduling algorithms for hard real-time systems, such as rate monotonic scheduling and earliest-deadline-first scheduling, are presented in [Liu and Layland (1973)].

[Anderson et al. (1989)], [Lewis and Berg (1998)], and [Philbin et al. (1996)] discuss thread scheduling. Multicore scheduling is examined in [McNairy and Bhatia (2005)] and [Kongetira et al. (2005)].

[Fisher (1981)], [Hall et al. (1996)], and [Lowney et al. (1993)] describe scheduling techniques that take into account information regarding process execution times from previous runs.


Scheduling policies used in the UNIX V operating system are described by [Bach (1987)]; those for UNIX FreeBSD 5.2 are presented by [McKusick and Neville-Neil (2005)]; and those for the Mach operating system are discussed by [Black (1990)]. [Love (2010)] and [Mauerer (2008)] cover scheduling in