Chapter 8 – Processor Scheduling

Outline

Objectives

- After reading this chapter, you should understand:
  - the goals of processor scheduling.
  - preemptive vs. nonpreemptive scheduling.
  - the role of priorities in scheduling.
  - scheduling criteria.
  - common scheduling algorithms.
  - the notions of deadline scheduling and real-time scheduling.
  - Java thread scheduling.

8.1 Introduction

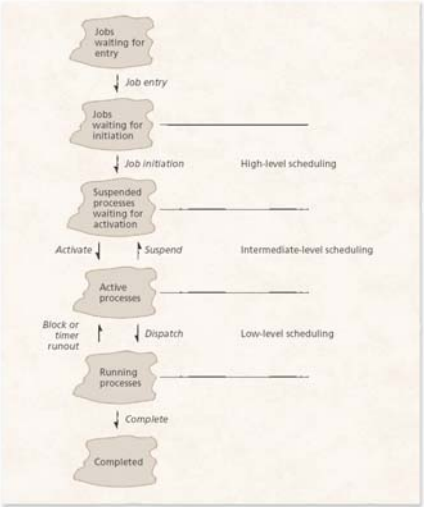
- Processor scheduling policy
  - Decides which process runs at given time
  - Different schedulers will have different goals
    - Maximize throughput
    - Minimize latency
    - Prevent indefinite postponement
    - Complete process by given deadline
    - Maximize processor utilization

8.2 Scheduling Levels

- High-level scheduling
  - Determines which jobs can compete for resources
  - Controls number of processes in system at one time
- Intermediate-level scheduling
  - Determines which processes can compete for processors
  - Responds to fluctuations in system load
- Low-level scheduling
  - Assigns priorities
  - Assigns processors to processes

8.2 Scheduling Levels

Figure 8.1 Scheduling levels.



8.3 Preemptive vs. Nonpreemptive Scheduling

- Preemptive processes
  - Can be removed from their current processor
  - Can lead to improved response times
  - Important for interactive environments
  - Preempted processes remain in memory
- Nonpreemptive processes
  - Run until completion or until they yield control of a processor
  - Unimportant processes can block important ones indefinitely

8.4 Priorities

- Static priorities
  - Priority assigned to a process does not change
  - Easy to implement
  - Low overhead
  - Not responsive to changes in environment
- Dynamic priorities
  - Responsive to change
  - Promote smooth interactivity
  - Incur more overhead than static priorities
    - Justified by increased responsiveness

8.5 Scheduling Objectives

- Different objectives depending on system
  - Maximize throughput
  - Maximize number of interactive processes receiving acceptable response times
  - Minimize resource utilization
  - Avoid indefinite postponement
  - Enforce priorities
  - Minimize overhead
  - Ensure predictability

8.5 Scheduling Objectives

- Several goals common to most schedulers
  - Fairness
  - Predictability
  - Scalability

8.6 Scheduling Criteria

- Processor-bound processes
  - Use all available processor time
- I/O-bound
  - Generates an I/O request quickly and relinquishes processor
- Batch processes
  - Contains work to be performed with no user interaction
- Interactive processes
  - Requires frequent user input

8.7 Scheduling Algorithms

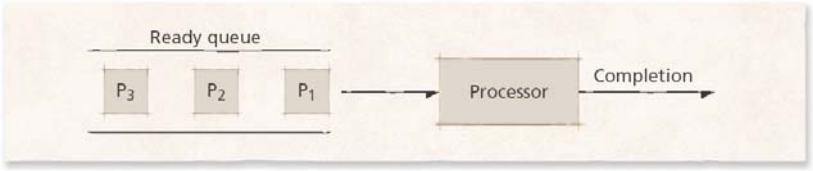
- Scheduling algorithms
  - Decide when and for how long each process runs
  - Make choices about
    - Preemptibility
    - Priority
    - Running time
    - Run-time-to-completion
    - fairness

8.7.1 First-In-First-Out (FIFO) Scheduling

- FIFO scheduling
  - Simplest scheme
  - Processes dispatched according to arrival time
  - Nonpreemptible
  - Rarely used as primary scheduling algorithm

8.7.1 First-In-First-Out (FIFO) Scheduling

Figure 8.2 First-in-first-out scheduling.

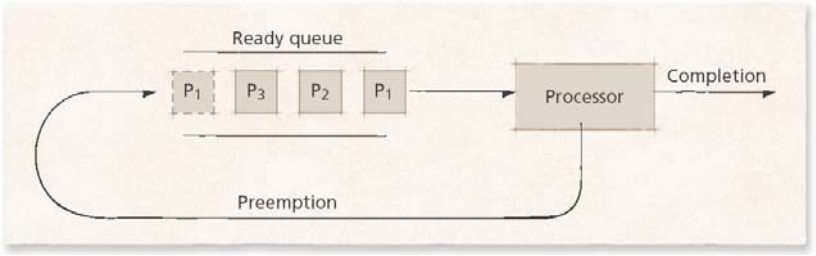


8.7.2 Round-Robin (RR) Scheduling

- Round-robin scheduling
  - Based on FIFO
  - Processes run only for a limited amount of time called a time slice or quantum
  - Preemptible
  - Requires the system to maintain several processes in memory to minimize overhead
  - Often used as part of more complex algorithms

8.7.2 Round-Robin (RR) Scheduling

Figure 8.3 Round-robin scheduling.



8.7.2 Round-Robin (RR) Scheduling

- Selfish round-robin scheduling
  - Increases priority as process ages
  - Two queues
    - Active
    - Holding
  - Favors older processes to avoids unreasonable delays

8.7.2 Round-Robin (RR) Scheduling

- Quantum size
  - Determines response time to interactive requests
  - Very large quantum size
    - Processes run for long periods
    - Degenerates to FIFO
  - Very small quantum size
    - System spends more time context switching than running processes
  - Middle-ground
    - Long enough for interactive processes to issue I/O request
    - Batch processes still get majority of processor time

8.7.3 Shortest-Process-First (SPF) Scheduling

- Scheduler selects process with smallest time to finish
  - Lower average wait time than FIFO
    - Reduces the number of waiting processes
  - Potentially large variance in wait times
  - Nonpreemptive
    - Results in slow response times to arriving interactive requests
  - Relies on estimates of time-to-completion
    - Can be inaccurate or falsified
  - Unsuitable for use in modern interactive systems

8.7.4 Highest-Response-Ratio-Next (HRRN) Scheduling

- HRRN scheduling
  - Improves upon SPF scheduling
  - Still nonpreemptive
  - Considers how long process has been waiting
  - Prevents indefinite postponement

8.7.5 Shortest-Remaining-Time (SRT) Scheduling

- SRT scheduling
  - Preemptive version of SPF
  - Shorter arriving processes preempt a running process
  - Very large variance of response times: long processes wait even longer than under SPF
  - Not always optimal
    - Short incoming process can preempt a running process that is near completion
    - Context-switching overhead can become significant

8.7.6 Multilevel Feedback Queues

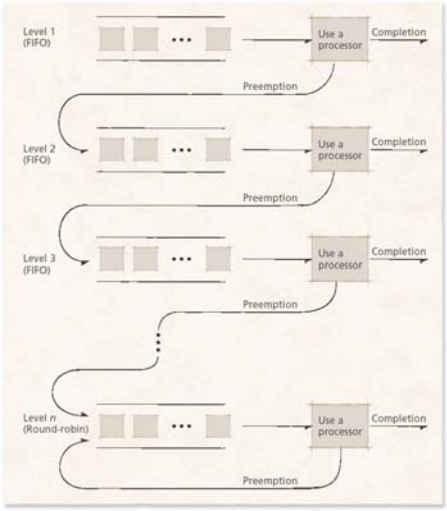
- Different processes have different needs
  - Short I/O-bound interactive processes should generally run before processor-bound batch processes
  - Behavior patterns not immediately obvious to the scheduler
- Multilevel feedback queues
  - Arriving processes enter the highest-level queue and execute with higher priority than processes in lower queues
  - Long processes repeatedly descend into lower levels
    - Gives short processes and I/O-bound processes higher priority
    - Long processes will run when short and I/O-bound processes terminate
  - Processes in each queue are serviced using round-robin
    - Process entering a higher-level queue preempt running processes

8.7.6 Multilevel Feedback Queues

- Algorithm must respond to changes in environment
  - Move processes to different queues as they alternate between interactive and batch behavior
- Example of an adaptive mechanism
  - Adaptive mechanisms incur overhead that often is offset by increased sensitivity to process behavior

8.7.6 Multilevel Feedback Queues

Figure 8.4 Multilevel feedback queues.

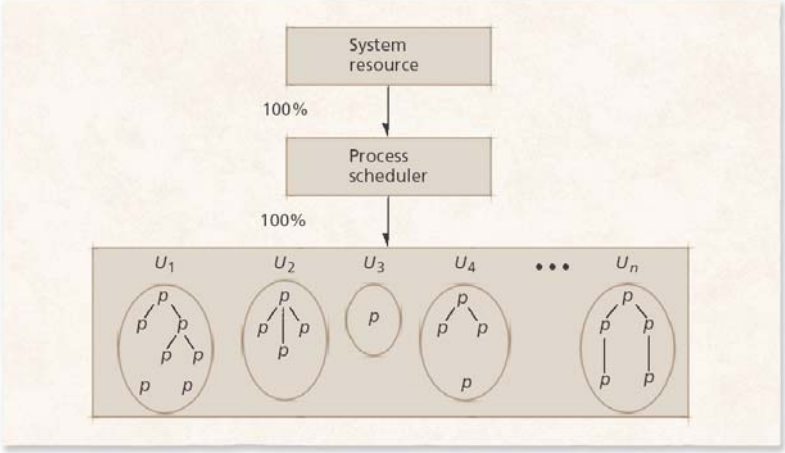


8.7.7 Fair Share Scheduling

- FSS controls users' access to system resources
  - Some user groups more important than others
  - Ensures that less important groups cannot monopolize resources
  - Unused resources distributed according to the proportion of resources each group has been allocated
  - Groups not meeting resource-utilization goals get higher priority

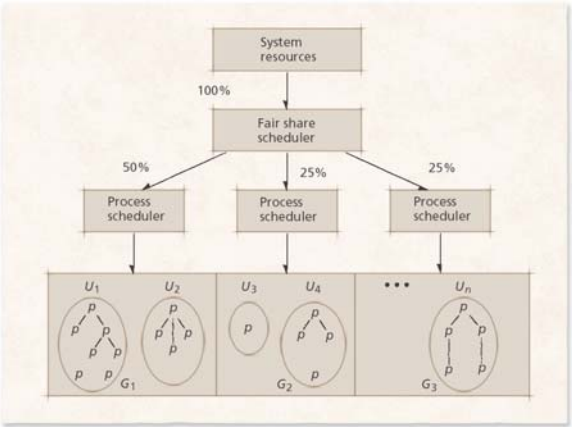
8.7.7 Fair Share Scheduling

Figure 8.5 Standard UNIX process scheduler. The scheduler grants the processor to users, each of whom may have many processes. (Property of AT&T Archives. Reprinted with permission of AT&T.)



8.7.7 Fair Share Scheduling

Figure 8.6 Fair share scheduler. The fair share scheduler divides system resource capacity into portions, which are then allocated by process schedulers assigned to various fair share groups. (Property of AT&T Archives. Reprinted with permission of AT&T.)



8.8 Deadline Scheduling

- Deadline scheduling
  - Process must complete by specific time
  - Used when results would be useless if not delivered on-time
  - Difficult to implement
    - Must plan resource requirements in advance
    - Incurs significant overhead
    - Service provided to other processes can degrade

8.9 Real-Time Scheduling

- Real-time scheduling
  - Related to deadline scheduling
  - Processes have timing constraints
  - Also encompasses tasks that execute periodically
- Two categories
  - Soft real-time scheduling
    - Does not guarantee that timing constraints will be met
    - For example, multimedia playback
  - Hard real-time scheduling
    - Timing constraints will always be met
    - Failure to meet deadline might have catastrophic results
    - For example, air traffic control

8.9 Real-Time Scheduling

- Static real-time scheduling
  - Does not adjust priorities over time
  - Low overhead
  - Suitable for systems where conditions rarely change
    - Hard real-time schedulers
  - Rate-monotonic (RM) scheduling
    - Process priority increases monotonically with the frequency with which it must execute
  - Deadline RM scheduling
    - Useful for a process that has a deadline that is not equal to its period

8.9 Real-Time Scheduling

- Dynamic real-time scheduling
  - Adjusts priorities in response to changing conditions
  - Can incur significant overhead, but must ensure that the overhead does not result in increased missed deadlines
  - Priorities are usually based on processes' deadlines
    - Earliest-deadline-first (EDF)
      - Preemptive, always dispatch the process with the earliest deadline
    - Minimum-laxity-first
      - Similar to EDF, but bases priority on laxity, which is based on the process's deadline and its remaining run-time-to-completion

8.10 Java Thread Scheduling

- Operating systems provide varying thread scheduling support
  - User-level threads
    - Implemented by each program independently
    - Operating system unaware of threads
  - Kernel-level threads
    - Implemented at kernel level
    - Scheduler must consider how to allocate processor time to a process's threads

8.10 Java Thread Scheduling

- Java threading scheduler
  - Uses kernel-level threads if available
  - User-mode threads implement timeslicing
    - Each thread is allowed to execute for at most one quantum before preemption
  - Threads can `yield` to others of equal priority
    - Only necessary on nontimesliced systems
    - Threads waiting to run are called waiting, sleeping or blocked

8.10 Java Thread Scheduling

Figure 8.7 Java thread priority scheduling.