Chapter 8 – Processor Scheduling

Outline
8.1 Introduction
8.2 Scheduling Levels
8.3 Preemptive vs. Nonpreemptive Scheduling
8.4 Priorities
8.5 Scheduling Objectives
8.6 Scheduling Criteria
8.7 Scheduling Algorithms
8.7.1 First-In-First-Out (FIFO) Scheduling
8.7.2 Round-Robin (RR) Scheduling
8.7.3 Shortest-Process-First (SPF) Scheduling
8.7.4 Highest-Response-Ratio-Next (HRRN) Scheduling
8.7.5 Shortest-Remaining-Time (SRT) Scheduling
8.7.6 Multilevel Feedback Queues
8.7.7 Fair Share Scheduling
8.8 Deadline Scheduling
8.9 Real-Time Scheduling
8.10 Java Thread Scheduling

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.
Process and Thread Scheduling

- Process scheduling
  - Decision making policies to determine the order in which active processes should compete for the use of processors
- Process dispatcher
  - The actual binding of a selected process to a processor
  - Remove the process from the ready queue, change its status, load the processor state

- Jointly we refer to both components as the scheduler

Schedulers

- Long-term scheduler (or job scheduler)
  - selects which processes should be brought into the ready queue.
  - (Caution) A process may have not been loaded into the memory yet
  - invoked very infrequently (second, minutes)
  - Degree of multiprogramming
  - Good process mix (I/O bound vs CPU bound)
- Short-term scheduler (or CPU scheduler)
  - selects which process should be executed next and allocates CPU.
  - Schedule only those processes already in the memory
  - invoked very frequently (millisecond)
  - CPU utilization

Organization of Schedulers

- Embedded
  - Called as a function at the end of kernel call
  - Runs as part of the calling process
  - Shared by the processes
  - May have several different OSs by having a group processes belong to different schedulers
- Autonomous
  - A separate process
  - May have dedicated CPU on a multiprocessor for scheduling
    - preferable in multiprocessor or distributed systems
  - On single-processor, run at every quantum:
    - scheduler and other processes alternate
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

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
Context Switch

- When CPU switches to another process, the system must save the state of the old process and load the saved state for the new process.
- Context-switch time is overhead
  - 100 ms CPU time, 10 ms scheduling & context switching (10% overhead)
- Time dependent on hardware support:
  - Memory speed, number of registers, existence of special instructions
  - 1 – 1000 microseconds

Dispatcher

- Gives control of the CPU to the process selected by the short-term scheduler
  - switching context
  - switching to user mode
  - jumping to the proper location in the user program to restart that program
- Dispatch latency – time it takes for the dispatcher to stop one process and start another running.

Scheduling Algorithm Design

- The number of processes usually exceed the number of available processors (CPUs)
- Scheduler perform resource allocation
  - Employ different strategies for different types of processes
  - (eg) batch jobs vs. interactive jobs
  - I/O bound vs. CPU bound
  - system process vs. user process
  - A general framework to handle broad spectrum of scheduling strategies
- Basic questions
  - When to schedule: governed by decision mode
  - Who to schedule:
    - Governed by priority function and arbitration mode
    - How to choose a process among the currently active processes
Scheduling Methods

• When is scheduler invoked?
  – Decision mode
    • Nonpreemptive
      – Scheduler called when process terminates or blocks
      – More economical
      – Not adequate for real-time and time-shared systems
    • Preemptive
      – Scheduler called periodically (quantum-oriented) or when
        system state changes (create or unblock)
      – More costly than non-preemptive
        » Process switching time
        » Additional logic in the scheduler
        » Possible swapping of programs and data between
          main memory and secondary memory

• Most OSs combine both strategies
  – Kernel routine: non-preemptive
  – Use process: preemptive

• How does it select highest priority process?
  – Priority function: \( P = \text{Priority}(p) \)
    • For all active processes, evaluate priority functions
    • Arbitration rule: break ties when there is more than one process
      with the same priority

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
Scheduling Algorithms

• Problem Description
  - Two processes P1 and P2
  - P1: arrives at 0, needs five CPU time units
  - P2: arrives at 2, needs two CPU time units

FIFO (First Come First Out)
- Criterion:
  - According to the arrival time (real-time r)
  - \( P = r \)
  - The earlier the arrival, the higher the priority
- Non-preemptive

SJF (Shortest Job First)
- Criterion:
  - According to the total service time (t)
  - \( P = t \)
  - The larger the value of t, the lower the priority
- Non-preemptive
Scheduling Algorithms

• Shortest Remaining Time (SRT)
  - Criterion:
    • According to the remaining service time (t-a)
    • \( P = \frac{(t-a)}{t} \)
    • Highest priority to the process that will need the least amount of time to complete
  - Preemptive
  - Dynamic version of SJF

[Diagram showing P1 and P2]

• Round Robin (RR)
  - Criterion:
    • Fixed time quantum (time slice) \( q \)
    • Used by most time sharing systems
  - Preemptive

[Diagram showing P1, P2, P1, P2, P1]

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

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

- 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

Scheduling Algorithms

- Multilevel Priority
  - A fixed set of priorities
    - Within the same priority queue, either Round Robin or FIFO scheduling can be used
  - Preemptive
    - Higher priority processes always preempt lower priority processes
  - Possibility of starvation, Solaris 2
Priority Scheduling

- Problem
  - Starvation
  - low priority processes may never execute.
  - (eg) When they shut down the IBM 7094 at MIT in 1973, they found a low-priority process that had been submitted in 1967 and had not yet been run
- Solution
  - Aging – as time progresses increase the priority of the process.
  - Decrement the priority of a waiting process by 1 every 15 minutes

MLF (Multilevel Feedback)

- Like ML, but priority changes dynamically
- A process can move between the various queues
- Every process enters at highest level $n$
- Each level $P$ prescribes maximum time $t_P$
- $t_P$ increases as $P$ decreases
- Typically:
  - $t_n = T$ (constant)
  - $t_P = 2 \times t_{P+1}$
Example of Multilevel Feedback Queue

- Three queues:
  - $Q_0$ – time quantum 8 milliseconds
  - $Q_1$ – time quantum 16 milliseconds
  - $Q_2$ – FCFS

- Scheduling
  - A new job enters queue $Q_0$ which is served FCFS. When it gains CPU, job receives 8 milliseconds. If it does not finish in 8 milliseconds, job is moved to queue $Q_1$.
  - A job at $Q_1$ is again served FCFS and receives 16 additional milliseconds. If it still does not complete, it is preempted and moved to queue $Q_2$.

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
  - 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

8.7.7 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.)

8.8 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

- 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

- 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
Comparison of Methods

- Real-time systems
  - P1: t=1, d=4 (t: time needed, d: period)
  - P2: t=3, d=5

Real-time Scheduling Algorithms

Rate Monotonic (RM):
- Intended for periodic (real-time) processes
- If a process has a period d, it is activated every d units of time
- Computation must be completed before the start of the next period
- Preemptive
- Highest priority: shortest period

Earliest Deadline First (EDF):
- Intended for periodic (real-time) processes
- Preemptive
- Highest priority: shortest remaining time to the next deadline
  - Highest priority: the shorter, the higher

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 non-timesliced systems
  - Threads waiting to run are called waiting, sleeping or blocked

Comparison of Methods

- FIFO, SJF, SRT
  - Developed primarily for batch processing systems
  - SJF and SRT are preferable to FIFO if the total service times known in that the shorter jobs are serviced first
    - Fast checkout line in the supermarket!
  - Reduced average turnaround time
    - The order in which processes are executed has no effect on the total time to complete all processes
    - Depending on the scheduling policy, the turnaround time can be reduced
    - Average turnaround time
      - \( \frac{(r_1 + r_2 + \ldots + r_n)}{n} \)
      - where \( r_i \) is the real time for process \( i \) and \( n \) = total number of processes
Comparison of Methods

- FIFO, SJF, SRT
  - FIFO simplest, SJF & SRT have better average turnaround times:
  - (eg) two processes, p1 and p2
    p1: total service time t1 during its lifetime
    p2: total service time t2 during its lifetime
  - SJF: p1 → p2 therefore, average turnaround time = \((t1 + (t1+t2))/2\)
  - It’s the best performance in that the other scheduling leads to
    \((t2+(t2+t1))/n\)

Comparison of Methods

<table>
<thead>
<tr>
<th></th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIFO</td>
<td>0  + 4</td>
<td>4  + 2</td>
<td>3  + 1</td>
<td>14/3 = 4.66</td>
</tr>
<tr>
<td>SJF</td>
<td>2  + 4</td>
<td>0  + 2</td>
<td>3  + 1</td>
<td>12/3 = 4.00</td>
</tr>
<tr>
<td>SRT</td>
<td>3  + 4</td>
<td>0  + 2</td>
<td>0  + 1</td>
<td>10/3 = 3.33</td>
</tr>
</tbody>
</table>

Average Turnaround Time
Scheduling Optimization Criteria

- CPU utilization
  - keep the CPU as busy as possible
- Throughput
  - # of processes that complete their execution per time unit
- Turnaround time
  - amount of time to execute a particular process
- Waiting time
  - amount of time a process has been waiting in the ready queue
- Response time
  - amount of time it takes from when a request was submitted until the first response is produced, not output (for time-sharing environment)

Optimization Criteria

- Maximize CPU utilization
- Maximize throughput
- Minimize turnaround time
- Minimize waiting time
- Minimize response time

Comparison of Methods

- Time-sharing systems
  - Response time is critical
  - RR or MLF with RR within each queue are suitable
  - Choice of quantum determines overhead
    - When \( q \to \infty \), RR approaches FIFO
    - When \( q \to 0 \), context switch overhead \( \to 100\% \)
      - No work is done
    - When \( q \gg \) context switch overhead, \( n \) processes run concurrently at \( 1/n \) CPU speed
Exercise 1 (Average Waiting Time)

- Five processes at time \( t \)

<table>
<thead>
<tr>
<th>Process</th>
<th>CPU Burst Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>10</td>
</tr>
<tr>
<td>P2</td>
<td>29</td>
</tr>
<tr>
<td>P3</td>
<td>3</td>
</tr>
<tr>
<td>P4</td>
<td>7</td>
</tr>
<tr>
<td>P5</td>
<td>12</td>
</tr>
</tbody>
</table>

Which algorithm would give the minimum average waiting time? FCFS, SJF, RR (quantum=10 ms)

Answer

- FCFS : \((0 + 10 + 39 + 42 + 49) / 5 = 28\)
- SJF : \((0 + 3 + 10 + 20 + 32) / 5 = 13\)
- RR : \((0 + (10+3+7+10+2) + 20 + 23 + (30+10))/ 5 = 23\)

Exercise 2 (Turnaround Time)

- Three processes

<table>
<thead>
<tr>
<th>Process</th>
<th>Arrival Time</th>
<th>Burst Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>0</td>
<td>15</td>
</tr>
<tr>
<td>P2</td>
<td>4</td>
<td>15</td>
</tr>
<tr>
<td>P3</td>
<td>10</td>
<td>10</td>
</tr>
</tbody>
</table>

What is the average turnaround time for these processes with

1. FCFS scheduling algorithm
2. RR (quantum 10)
Exercise 2 (Turnaround Time)

- FCFS

<table>
<thead>
<tr>
<th>Process</th>
<th>Arrival Time</th>
<th>Burst Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>0</td>
<td>15</td>
</tr>
<tr>
<td>P2</td>
<td>4</td>
<td>15</td>
</tr>
<tr>
<td>P3</td>
<td>10</td>
<td>10</td>
</tr>
</tbody>
</table>

P1 : 15, P2 : 21, P3 : 30
Average turnaround time = (15+21+30)/3 = 22