Chapter 5
Process Scheduling

CPU Scheduling

- Objective:
  - Basic Scheduling Concepts
  - CPU Scheduling Algorithms

- Why Multiprogramming?
  - Maximize CPU/Resources Utilization
    (Based on Some Criteria)
CPU Scheduling

- Process Execution
  - CPU-bound programs tend to have a few very long CPU bursts.
  - IO-bound programs tend to have many very short CPU bursts.

The distribution can help in selecting an appropriate CPU-scheduling algorithms.

* All rights reserved, Tei-Wei Kuo, National Taiwan University, 2005.
**CPU Scheduling**

- CPU Scheduler – The Selection of Process for Execution
  - A short-term scheduler

![Diagram of CPU states]

**Nonpreemptive Scheduling**

- A running process keeps CPU until it volunteers to release CPU
  - E.g., I/O or termination

- Advantage
  - Easy to implement (at the cost of service response to other processes)
  - E.g., Windows 3.1
CPU Scheduling

- Preemptive Scheduling
  - Beside the instances for non-preemptive scheduling, CPU scheduling occurs whenever some process becomes ready or the running process leaves the running state!

- Issues involved:
  - Protection of Resources, such as I/O queues or shared data, especially for multiprocessor or real-time systems.
  - Synchronization
    - E.g., Interrupts and System calls

CPU Scheduling

- Dispatcher
  - Functionality:
    - Switching context
    - Switching to user mode
    - Restarting a user program

- Dispatch Latency:
  Must be fast
  Stop a process → Start a process
Scheduling Criteria

- Why?
  - Different scheduling algorithms may favor one class of processes over another!

- Criteria
  - CPU Utilization
  - Throughput
  - Turnaround Time: CompletionT-StartT
  - Waiting Time: Waiting in the ReadyQ
  - Response Time: FirstResponseTime

Scheduling Criteria

- How to Measure the Performance of CPU Scheduling Algorithms?

- Optimization of what?
  - General Consideration
    - Average Measure
    - Minimum or Maximum Values
  - Variance → Predictable Behavior
Scheduling Algorithms

- First-Come, First-Served Scheduling (FIFO)
- Shortest-Job-First Scheduling (SJF)
- Priority Scheduling
- Round-Robin Scheduling (RR)
- Multilevel Queue Scheduling
- Multilevel Feedback Queue Scheduling
- Multiple-Processor Scheduling

First-Come, First-Served Scheduling (FCFS)

- The process which requests the CPU first is allocated the CPU
- Properties:
  - Non-preemptive scheduling
  - CPU might be held for an extended period.

* All rights reserved, Tei-Wei Kuo, National Taiwan University, 2005.
First-Come, First-Served Scheduling (FCFS)

- Example

<table>
<thead>
<tr>
<th>Process</th>
<th>CPU Burst Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>24</td>
</tr>
<tr>
<td>P2</td>
<td>3</td>
</tr>
<tr>
<td>P3</td>
<td>3</td>
</tr>
</tbody>
</table>

**Gantt Chart**

- P1 24 27 30
- P2 3 6 3 0

Average waiting time for P1: 
\[(0+24+27)/3 = 17\]

Average waiting time for P2: 
\[(6+0+3)/3 = 3\]

*The average waiting time is highly affected by process CPU burst times !

Example: Convoy Effect
- One CPU-bound process + many I/O-bound processes

CPU

I/O device

ready queue

idle

All other processes wait for it to get off the CPU!
Shortest-Job-First Scheduling (SJF)

- **Non-Preemptive SJF**
  - Shortest next CPU burst first

<table>
<thead>
<tr>
<th>process</th>
<th>CPU burst time</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>6</td>
</tr>
<tr>
<td>P2</td>
<td>8</td>
</tr>
<tr>
<td>P3</td>
<td>7</td>
</tr>
<tr>
<td>P4</td>
<td>3</td>
</tr>
</tbody>
</table>

Average waiting time
= \(\frac{3+16+9+0}{4} = 7\)

Nonpreemptive SJF is optimal when processes are all ready at time 0
- The minimum average waiting time!
- Prediction of the next CPU burst time?
- Long-Term Scheduler
  - A specified amount at its submission time
- Short-Term Scheduler
  - Exponential average \((0\leq \alpha \leq 1)\)

\[
\tau_{n+1} = \alpha \tau_n + (1-\alpha) \tau_n
\]
Shortest-Job-First Scheduling (SJF)

- Preemptive SJF
- Shortest-remaining-time-first

<table>
<thead>
<tr>
<th>Process</th>
<th>CPU Burst Time</th>
<th>Arrival Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>P2</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>P3</td>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>P4</td>
<td>5</td>
<td>3</td>
</tr>
</tbody>
</table>

**Process Timeline:**

- P1: 0-1
- P2: 1-5
- P4: 5-10
- P1: 10-17
- P3: 17-26

**Average Waiting Time:**

\[
\text{Average Waiting Time} = \frac{(10-1) + (1-1) + (17-2) + (5-3))}{4} = \frac{26}{4} = 6.5
\]

**Non-preemptive AWT:**

\[
\text{AWT} = \frac{0 + (10-1)}{2} = \frac{9}{2} = 4.5
\]

**Preemptive AWT:**

\[
\text{AWT} = \frac{(2-1) + 0}{2} = 0.5
\]

* Context switching cost ~ modeling & analysis

*All rights reserved, Tei-Wei Kuo, National Taiwan University, 2005.*
Priority Scheduling

- CPU is assigned to the process with the highest priority – A framework for various scheduling algorithms:
  - FCFS: Equal-Priority with Tie-Breaking by FCFS
  - SFJ: Priority = 1 / next CPU burst length

<table>
<thead>
<tr>
<th>Process</th>
<th>CPU Burst Time</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>10</td>
<td>3</td>
</tr>
<tr>
<td>P2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>P3</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>P4</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>P5</td>
<td>5</td>
<td>2</td>
</tr>
</tbody>
</table>

Gantt Graph

Average waiting time = (6+0+16+18+1)/5 = 8.2
Priority Scheduling

- Priority Assignment
  - Internally defined – use some measurable quantity, such as the # of open files, Average CPU Burst, Average I/O Burst
  - Externally defined – set by criteria external to the OS, such as the criticality levels of jobs.

- Preemptive or Non-Preemptive?
  - Preemptive scheduling – CPU scheduling is invoked whenever a process arrives at the ready queue, or the running process relinquishes the CPU.
  - Non-preemptive scheduling – CPU scheduling is invoked only when the running process relinquishes the CPU.
Priority Scheduling

- Major Problem
  - Indefinite Blocking (/Starvation)
    - Low-priority processes could starve to death!
  - A Solution: Aging
    - A technique that increases the priority of processes waiting in the system for a long time.

Round-Robin Scheduling (RR)

- RR is similar to FCFS except that preemption is added to switch between processes.
  - Interrupt at every time quantum (time slice)
  - Goal: Fairness – Time Sharing

The quantum is used up!
Round-Robin Scheduling (RR)

<table>
<thead>
<tr>
<th>Process</th>
<th>CPU Burst Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>24</td>
</tr>
<tr>
<td>P2</td>
<td>3</td>
</tr>
<tr>
<td>P3</td>
<td>3</td>
</tr>
</tbody>
</table>

Time slice = 4

```
0 4 7 14 18 22 26 30
P1 P2 P3 P1 P1 P1 P1
```

AWT = ((10-4) + (4-0) + (7-0))/3
= 17/3 = 5.66

Round-Robin Scheduling (RR)

- Service Size and Interval
  - Time quantum = q → Service interval <= (n-1)*q if n processes are ready.
  - IF q = ∞, then RR → FCFS.
  - IF q = ε, then RR → processor sharing. The # of context switchings increases!

<table>
<thead>
<tr>
<th>process</th>
<th>quantum</th>
<th>context switch #</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>10</td>
<td>9</td>
</tr>
</tbody>
</table>

If context switch cost
time quantum = 10% => 1/11 of CPU is wasted!

* All rights reserved, Tei-Wei Kuo, National Taiwan University, 2005.
Round-Robin Scheduling (RR)

- **Turnaround Time**

<table>
<thead>
<tr>
<th>Process (10ms)</th>
<th>Quantum = 10</th>
<th>Quantum = 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>0-10</td>
<td>0-10-20-30</td>
</tr>
<tr>
<td>P2</td>
<td>10-20</td>
<td>10-20-30</td>
</tr>
<tr>
<td>P3</td>
<td>20-30</td>
<td>20-30-30</td>
</tr>
</tbody>
</table>

Average Turnaround Time

\[
\text{ATT} = \frac{(28+29+30)}{3} = 29
\]

\(=> 80\% \text{ CPU Burst} < \text{time slice}\)

Multilevel Queue Scheduling

- Partition the ready queue into several separate queues \(=>\)
- Processes can be classified into different groups and permanently assigned to one queue.

- **System Processes**
- **Interactive Processes**
- **Batch Processes**

* All rights reserved, Tei-Wei Kuo, National Taiwan University, 2005.*
Multilevel Queue Scheduling

- Intra-queue scheduling
  - Independent choice of scheduling algorithms.

- Inter-queue scheduling
  - Fixed-priority preemptive scheduling
    - e.g., foreground queues always have absolute priority over the background queues.
  - Time slice between queues
    - e.g., 80% CPU is given to foreground processes, and 20% CPU to background processes.
  - More??

Multilevel Feedback Queue Scheduling

- Different from Multilevel Queue Scheduling by Allowing Processes to Migrate Among Queues.

- Configurable Parameters:
  - # of queues
  - The scheduling algorithm for each queue
  - The method to determine when to upgrade a process to a higher priority queue.
  - The method to determine when to demote a process to a lower priority queue.
  - The method to determine which queue a newly ready process will enter.

*Inter-queue scheduling: Fixed-priority preemptive?!
Multilevel Feedback Queue Scheduling

- Example

quantum = 8

quantum = 16

FCFS

Idea: Separate processes with different CPU-burst characteristics!

Multiple-Processor Scheduling

- CPU scheduling in a system with multiple CPUs
- A Homogeneous System
  - Processes are identical in terms of their functionality.
    ➔ Can processes run on any processor?
- A Heterogeneous System
  - Programs must be compiled for instructions on proper processors.
Multiple-Processor Scheduling

- Load Sharing – Load Balancing!!
  - A queue for each processor
    - Self-Scheduling – Symmetric Multiprocessing
  - A common ready queue for all processors.
    - Self-Scheduling
      - Need synchronization to access common data structure, e.g., queues.
    - Master-Slave – Asymmetric Multiprocessing
      - One processor accesses the system structures → no need for data sharing

Multiple-Processor Scheduling

- Load Balancing
  - Push migration: A specific task periodically checks for imbalance and migrate tasks
  - Pull migration: An idle processor pulls a waiting task from a busy processor
  - Linux and FreeBSD do both!
- Processor Affinity
  - The system might avoid process migration because of the cost in invalidating or repopulating caches
  - Soft or hard affinity
Multiple-Processor Scheduling

- Symmetric Multithreading (SMT), i.e., Hyperthreading
  - A feature provided by the hardware
  - Several logical processors per physical processor
    - Each has its own architecture state, including registers.
- Issues: Process Synchronization

Multiple-Processor Scheduling – SMT

Superscalar

occupied issue slot

unused issue slot

\[ \text{time} \]

\[ \text{: 13} \]

\[ \text{: 10} \]

\[ \text{: 11} \]
Thread Scheduling

- Two Scopes:
  - Process Contention Scope (PCS): m:1 or m:m
    - Priority-Driven
  - System-Contention Scope (SCS): 1:1

- Pthread Scheduling
  - PCS and SCS
    
    
    ```
    Pthread_attr_setscope(pthread_attr_t *attr, int scope)
    Pthread_attr_getscope(pthread_attr_t *attr, int *scope)
    ```

Operating System Examples

- Process Local Scheduling
  - E.g., those for user-level threads
  - Thread scheduling is done locally to each application.

- System Global Scheduling
  - E.g., those for kernel-level threads
  - The kernel decides which thread to run.
Operating System Examples – Solaris

- Priority-Based Process Scheduling
  - Real-Time
  - System
    - Kernel-service processes
  - Time-Sharing
    - A default class
  - Interactive
- Each LWP inherits its class from its parent process

- Real-Time
  - A guaranteed response
- System
  - The priorities of system processes are fixed.
- Time-Sharing
  - Multilevel feedback queue scheduling – priorities inversely proportional to time slices
- Interactive
  - Prefer windowing process

* All rights reserved, Tei-Wei Kuo, National Taiwan University, 2005.
The selected thread runs until one of the following occurs:

- It blocks.
- It uses its time slice (if it is not a system thread).
- It is preempted by a higher-priority thread.
- RR is used when several threads have the same priority.
Operating System Examples – Solaris

- Two New Classes in Solaris 9
  - Fixed Priority
    - Non-adjusted priorities in the range of the time-sharing class
  - Fair Sharing
    - CPU shares, instead of priorities

Operating System Examples – Windows XP

- Priority-Based Preemptive Scheduling
  - Priority Class/Relationship: 0..31
  - Dispatcher: A process runs until
    - It is preempted by a higher-priority process.
    - It terminates
    - Its time quantum ends
    - It calls a blocking system call
  - Idle thread
  - A queue per priority level
Each thread has a base priority that represents a value in the priority range of its class.

A typical class – Normal_Priority_Class

Time quantum – thread
- Increased after some waiting
  - Different for I/O devices.
- Decreased after some computation
  - The priority is never lowered below the base priority.
- Favor foreground processes (more time quantum)

<table>
<thead>
<tr>
<th>Base Priority</th>
<th>Real-time</th>
<th>High</th>
<th>Above normal</th>
<th>Normal</th>
<th>Below normal</th>
<th>Idle priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time-critical</td>
<td>31</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>Highest</td>
<td>26</td>
<td>15</td>
<td>12</td>
<td>10</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>Above normal</td>
<td>25</td>
<td>14</td>
<td>11</td>
<td>9</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>Normal</td>
<td>24</td>
<td>13</td>
<td>10</td>
<td>8</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>Below normal</td>
<td>23</td>
<td>12</td>
<td>9</td>
<td>7</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>Lowest</td>
<td>22</td>
<td>11</td>
<td>8</td>
<td>6</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>Idle</td>
<td>16</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

* All rights reserved, Tei-Wei Kuo, National Taiwan University, 2005.
Operating System Examples – Linux Ver. 2.5+

- **Scheduling Algorithm**
  - O(1)
  - SMP, load balancing, and processor affinity
  - Fairness and support for interactive tasks
- **Priorities**
  - Real-time: 0..99
  - Nice: 100..140

<table>
<thead>
<tr>
<th>Numeric Priority</th>
<th>Time Quantum</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>200ms</td>
</tr>
<tr>
<td>99</td>
<td>10ms</td>
</tr>
<tr>
<td>100</td>
<td>10ms</td>
</tr>
<tr>
<td>140</td>
<td></td>
</tr>
</tbody>
</table>

- **Real-Time Tasks**
- **Other Tasks**

Operating System Examples – Linux Ver. 2.5+

- Each processor has a runqueue
  - An active array and an expired array
  - Switching of the two arrays when all processes in the active array have their quantum expired.
- **Priority-Driven Scheduling**
  - Fixed Priority – Real-Time
  - Dynamic Priority – nice ± x, for x <= 5
    - Interactive tasks are favored.
    - The dynamic priority of a task is recalculated when its quantum is expired.
Algorithm Evaluation

- A General Procedure
  - Select criteria that may include several measures, e.g., maximize CPU utilization while confining the maximum response time to 1 second
  - Evaluate various algorithms
- Evaluation Methods:
  - Deterministic modeling
  - Queuing models
  - Simulation
  - Implementation

Deterministic Modeling

- A Typical Type of Analytic Evaluation
  - Take a particular predetermined workload and defines the performance of each algorithm for that workload
- Properties
  - Simple and fast
  - Through excessive executions of a number of examples, trends might be identified
  - But it needs exact numbers for inputs, and its answers only apply to those cases
    - Being too specific and requires too exact knowledge to be useful!
Deterministic Modeling

**Process** | **CPU Burst time**
--- | ---
P1 | 10
P2 | 29
P3 | 3
P4 | 7
P5 | 12

**Nonpreemptive Shortest Job First**

| P3 | P4 | P1 | P5 | P2 |
--- | --- | --- | --- | ---
0 | 3 | 10 | 20 | 32 |

Average Waiting Time (AWT) = (10+32+0+3+20)/5 = 13

**Round Robin (quantum =10)**

| P1 | P2 | P3 | P4 | P5 | P2 | P5 | P2 |
--- | --- | --- | --- | --- | --- | --- | ---
0 | 10 | 20 | 23 | 30 | 40 | 50 | 52 |

AWT = (0+(10+20+2)+20+23+(30+10))/5 = 23

Queueing Models

- **Motivation:**
  - Workloads vary, and there is no static set of processes
- **Models (~ Queueing-Network Analysis)**
  - **Workload:**
    - a. Arrival rate: the distribution of times when processes arrive.
    - b. The distributions of CPU & I/O bursts
  - **Service rate**
Queueing Models

- Model a computer system as a network of servers. Each server has a queue of waiting processes
  - Compute average queue length, waiting time, and so on.
- Properties:
  - Generally useful but with limited application to the classes of algorithms & distributions
  - Assumptions are made to make problems solvable => inaccurate results

**Example: Little’s formula**

\[ n = \lambda \times w \]

- \( n \) = # of processes in the queue
- \( \lambda \) = arrival rate
- \( w \) = average waiting time in the queue

- If \( n = 14 \) & \( \lambda = 7 \) processes/sec, then \( w = 2 \) seconds.
Simulation

- Motivation:
  - Get a more accurate evaluation.

- Procedures:
  - Program a model of the computer system
  - Drive the simulation with various data sets
    - Randomly generated according to some probability distributions
      => inaccuracy occurs because of only the occurrence frequency of events. Miss the order & the relationships of events.
  - Trace tapes: monitor the real system & record the sequence of actual events.

Properties:
- Accurate results can be gotten, but it could be expensive in terms of computation time and storage space.
- The coding, design, and debugging of a simulator can be a big job.
Implementation

- Motivation:
  - Get more accurate results than a simulation!

- Procedure:
  - Code scheduling algorithms
  - Put them in the OS
  - Evaluate the real behaviors

- Difficulties:
  - Cost in coding algorithms and modifying the OS
  - Reaction of users to a constantly changing the OS
  - The environment in which algorithms are used will change
    - For example, users may adjust their behaviors according to the selected algorithms
  => Separation of the policy and mechanism!