CPU Scheduling
CSCI 444/544 Operating Systems
Fall 2008

Agenda
- What is scheduling?
  - scheduling vs. allocation
- Scheduling criteria
- Scheduling algorithms
  - What are FCFS, SJF, RR and priority-based scheduling policies?
- Multi-level queue scheduling

Types of Resources
Resources can be classified into one of two groups
- non-preemptible resources
- preemptible resources
Type of resource determines how the OS manages it

Non-preemptible Resources
- Once given resource, cannot be reused until voluntarily relinquished
- Resource has complex or costly state associated with it
- Need many instances of this resource
- Example: Blocks on disk
- OS management: allocation
  - Decide which process gets which resource
Preemptible Resources

- Can take resource away, give it back later
- Resource has little state associated with it
- May only have one of this resource
- Example: CPU
- OS management: scheduling
  - Decide order in which requests are serviced
  - Decide how long process keeps resource

Levels of CPU Management

Dispatcher
- Low-level mechanism
- Performs context-switch
  - Save execution state of old process in PCB
  - Add PCB to appropriate queue (ready or waiting)
  - Load state of next process from PCB to registers
  - Switch from kernel to user mode
  - Jump to instruction in user process

Scheduler
- Policy to determine which process gets CPU and when
- Maximize CPU utilization with multiprogramming

CPU and I/O Bursts

CPU Workload Model

Workload contains a collection of jobs (processes)

Job model
- Job alternates between CPU and I/O bursts (i.e., moves between ready and waiting queues)
- CPU-bound job: Long CPU bursts
- I/O-bound job: Short CPU bursts

Do not know type of job before it executes
- Do not know duration of CPU or I/O burst

Need job scheduling for each ready job
- Schedule each CPU burst
Scheduling Metrics

CPU utilization

Throughput
- the number of completed processes per second

Turnaround time
- the time interval from submission of a process to its completion

Waiting time
- the sum of the periods spent waiting in the ready queue

Response time
- the time it takes to start responding
- the time interval from submission of a process to the first response

Scheduling Goals

Maximize resource utilization
- Keep expensive devices (e.g., CPU) busy

Maximize throughput
- Want many jobs to complete per unit of time

Minimize turnaround time
- Do not want to wait long for a job to complete

Minimize waiting time
- Do not want to spend much time in the Ready queue

Minimize response time
- Schedule interactive jobs promptly so users see output quickly

Minimize overhead
- Reduce number of context switches

Maximize fairness
- All jobs get the same amount of CPU over some time interval

Design Space

Two dimensions
- Selection algorithm
  - Which of the ready jobs should be run next?
- Preemption
  - Preemptive: currently running job may be interrupted and moved to Ready state
  - Non-preemptive: once a process is in the Running state, it continues to execute until it terminates or it blocks for I/O

Gantt Chart

Illustrates how jobs are scheduled over time on the CPU

Example:

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>0</td>
<td>10</td>
<td>12</td>
</tr>
</tbody>
</table>

Time
First-Come-First-Served (FCFS)

Idea: Maintain FIFO list of jobs as they arrive
- Non-preemptive policy
- Allocate CPU to job at head of list

<table>
<thead>
<tr>
<th>Job</th>
<th>Arrival</th>
<th>CPU burst</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>C</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>

Average wait time: 
\[ \frac{(0 + (10-1) + (12-2))}{3}=6.33 \]
Average turnaround time: 
\[ \frac{(10 + (12-1) + (16-2))}{3}=11.67 \]

FCFS Discussion

Advantage: Very simple implementation
Disadvantage
- Waiting time depends on arrival order
- Potentially long wait for jobs that arrive later
- Convoy effect: Short jobs stuck waiting for long jobs
  - Hurts waiting time of short jobs
  - Reduces utilization of I/O devices
  - Example: 1 mostly CPU-bound job, 3 mostly I/O-bound jobs

Shortest-Job-First (SJF)

Associate with each process the length of its next CPU burst. Use these lengths to schedule the process with the shortest time
- Two schemes:
  - non-preemptive – once CPU given to the process it cannot be preempted until completes its CPU burst
  - preemptive – if a new process arrives with CPU burst length less than remaining time of current executing process, preempt. This scheme is known as the Shortest-Remaining-Time-First (SRTF)

SJF is optimal – gives minimum average waiting time for a given set of processes

Example of Non-Preemptive SJF

<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>7</td>
</tr>
<tr>
<td>P₂</td>
<td>2.0</td>
<td>4</td>
</tr>
<tr>
<td>P₃</td>
<td>4.0</td>
<td>1</td>
</tr>
<tr>
<td>P₄</td>
<td>5.0</td>
<td>4</td>
</tr>
</tbody>
</table>

SJF (non-preemptive)
- use FCFS if jobs are of same length

Average waiting time = \( \frac{(0 + 6 + 3 + 7)}{4} = 4 \)
Example of Preemptive SJF

<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>7</td>
</tr>
<tr>
<td>P₂</td>
<td>2.0</td>
<td>4</td>
</tr>
<tr>
<td>P₃</td>
<td>4.0</td>
<td>1</td>
</tr>
<tr>
<td>P₄</td>
<td>5.0</td>
<td>4</td>
</tr>
</tbody>
</table>

Average waiting time = \((9 + 1 + 0 + 2)/4\) = 3

SJF Discussion

**Advantages**
- Provably optimal for minimizing average wait time
  - Moving shorter job before longer job improves waiting time of short job more than it harms waiting time of long job
- Helps keep I/O devices busy

**Disadvantages**
- Not practical: Cannot predict future CPU burst time
  - OS solution: Use past behavior to predict future behavior
- Starvation: Long jobs may never be scheduled

Round-Robin (RR)

Idea: Run each job for a time-slice and then move to back of FIFO queue
  - Preempt job if still running at end of time-slice

<table>
<thead>
<tr>
<th>Job</th>
<th>Arrival</th>
<th>CPU burst</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>C</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>

Average wait time:
\((6 + 2 + 4)/3\) = 4

RR Discussion

**Advantages**
- Jobs get fair share of CPU
- Shortest jobs finish relatively quickly

**Disadvantages**
- Poor average waiting time with similar job lengths
  - Example: 10 jobs each requiring 10 time slices
  - RR: All complete after about 100 time slices
  - FCFS performs better!
- Performance depends on length of time-slice
  - If time-slice too short, pay overhead of context switch
  - If time-slice too long, degenerate to FCFS
**RR Time-Slice**

IF time-slice too long, degenerate to FCFS

- Example:
  - Job A with 1 ms compute and 10 ms I/O
  - Job B always computes
  - Time-slice is 50 ms

**Priority-Based**

Idea: Each job is assigned a priority

- Schedule highest priority ready job
- May be preemptive or non-preemptive
- Priority may be static or dynamic

**Advantages**

- Static priorities work well for real-time systems
- Dynamic priorities work well for general workloads

**Disadvantages**

- Low priority jobs can starve
- How to choose priority of each job?

Goal: Adjust priority of job to match CPU burst

- Approximate SRTF by giving short jobs high priority

**Multilevel Queue**

Ready Queue is partitioned into separate queues

- foreground (interactive)
- background (batch)

Each queue has its own scheduling algorithm

- foreground — RR
- background — FCFS

Scheduling must be done between queues

- fixed priority scheduling
- Time slice

**Multilevel Queue Scheduling**