Previous class…

• Dealing with deadlock:
  – Prevention
  – Detection
  – Avoidance
Deadlock

• A set of processes is deadlocked when each process in the set is blocked awaiting an event that can only be triggered by another blocked process in the set.
Resource Allocation Graph describing the traffic jam
# Conditions for Deadlock

<table>
<thead>
<tr>
<th>Mutual Exclusion</th>
<th>Hold-and-Wait</th>
<th>No Pre-emption</th>
<th>Circular Wait</th>
</tr>
</thead>
<tbody>
<tr>
<td>A process cannot access a resource that has been allocated to another process</td>
<td>a process may hold allocated resources while awaiting assignment of others</td>
<td>no resource can be forcibly removed from a process holding it</td>
<td>a closed chain of processes exists, such that each process holds at least one resource needed by the next process in the chain</td>
</tr>
</tbody>
</table>
Dealing with Deadlock

- Three general approaches exist for dealing with deadlock:

  **Prevent Deadlock**
  - adopt a policy that eliminates one of the conditions

  **Avoid Deadlock**
  - make the appropriate dynamic choices based on the current state of resource allocation

  **Detect Deadlock**
  - attempt to detect the presence of deadlock and take action to recover
Dining Philosophers solution with numbered resources

Instead, number resources...

One philosopher can eat!

```c
#define N 5

void philosopher (int i) {
    while (TRUE) {
        think();
        take_fork(LOWER(i));
        take_fork(HIGHER(i));
        eat(); /* yummy */
        put_fork(LOWER(i));
        put_fork(HIGHER(i));
    }
}
```
Main Points

• Scheduling policy: what to do next, when there are multiple threads ready to run
  – Or multiple packets to send, or web requests to serve, or …

• Definitions
  – response time, throughput, predictability

• Uniprocessor policies
  – FIFO, Shortest Job First
  – round robin
  – multilevel feedback as approximation of optimal

• Multiprocessor policies

Some of the slides are courtesy of Dr. Thomas Anderson
Example

• You manage a web site, that suddenly becomes wildly popular. Do you?
  – Buy more hardware?
  – Turn away some users?
  – Implement a different scheduling policy?
Definitions

- Preemptive scheduler
  - If we can take resources away from a running task
- Response time
  - Time elapsed from the time of submission to response
- Turnaround time
  - Time elapsed from the time of submission to completion
- Throughput
  - # of tasks can be done per unit of time?
- Predictability
  - How consistent is the performance over time?
First In First Out (FIFO)

• Schedule tasks in the order they arrive
  – Continue running them until they complete or give up the processor
• Example: stores
• On what workloads is FIFO particularly bad?
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
  - Relies on estimates of time-to-completion
    - Can be inaccurate or falsified
  - Unsuitable for use in modern interactive systems
FIFO vs. SJF

**FIFO**

<table>
<thead>
<tr>
<th>Tasks</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1)</td>
<td></td>
</tr>
<tr>
<td>(2)</td>
<td></td>
</tr>
<tr>
<td>(3)</td>
<td></td>
</tr>
<tr>
<td>(4)</td>
<td></td>
</tr>
<tr>
<td>(5)</td>
<td></td>
</tr>
</tbody>
</table>

**SJF**

<table>
<thead>
<tr>
<th>Tasks</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1)</td>
<td></td>
</tr>
<tr>
<td>(2)</td>
<td></td>
</tr>
<tr>
<td>(3)</td>
<td></td>
</tr>
<tr>
<td>(4)</td>
<td></td>
</tr>
<tr>
<td>(5)</td>
<td></td>
</tr>
</tbody>
</table>
Question

• Does SJF have any downsides?
  – Starvation
  – Variance in response time
• Fastlane in supermarkets
Round Robin

• Each task gets resource for a fixed period of time (time quantum)
  – If task doesn’t complete, it goes back in line

• Need to pick a time quantum
  – What if time quantum is too long?
    • Becomes FIFO
  – What if time quantum is too short?
    • Large overhead for context switch
Round Robin

Round Robin (1 ms time slice)

Round Robin (100 ms time slice)
Round Robin = Fairness?

• Is Round Robin always fair?
Mixed Workload

Tasks

I/O Bound
- Issues
- I/O Request
- I/O Completes

CPU Bound

CPU Bound

Time
Multi-level Feedback Queue (MFQ)

• Goals:
  – Responsiveness
  – Low overhead
  – Starvation freedom
  – Some tasks are high/low priority
  – Fairness (among equal priority tasks)

• Not perfect at any of them!
  – Used in Linux (and probably Windows, MacOS)
MFQ

- Set of Round Robin queues
  - Each queue has a separate priority
- High priority queues have short time slices
  - Low priority queues have long time slices
- Scheduler picks first thread in highest priority queue
- Tasks start in highest priority queue
  - If time slice expires, task drops one level
  - If the process relinquishes the slice due to IO, it is kept in the current priority queue
## MFQ

<table>
<thead>
<tr>
<th>Priority</th>
<th>Time Slice (ms)</th>
<th>Round Robin Queues</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>80</td>
<td></td>
</tr>
</tbody>
</table>

- New or I/O Bound Task
- Time Slice Expiration
Uniprocessor Summary (1)

- FIFO is simple and minimizes overhead.
- If tasks are variable in size, then FIFO can have very poor average response time.
- If tasks are equal in size, FIFO is optimal in terms of average response time.
- Considering only the processor, SJF is optimal in terms of average response time.
- SJF leads to high variance in response time.
Uniprocessor Summary (2)

- If tasks are variable in size, Round Robin approximates SJF.
- If tasks are equal in size, Round Robin will have very poor average response time.
- Tasks that intermix processor do poorly under Round Robin.
- By manipulating the assignment of tasks to priority queues, an MFQ scheduler can achieve a balance between responsiveness, low overhead, and fairness.
Multiprocessor Scheduling

- What would happen if we used MFQ on a multiprocessor?
  - Contention for scheduler spinlock
  - Limited cache reuse: thread’s data from last time it ran is often still in its old cache
Per-Processor Affinity Scheduling

• Each processor has its own ready list
  – Protected by a per-processor spinlock
• When the system puts threads back on the ready list, they are put back where they had most recently run
• Exception: idle processors can steal work from other processors
Per-Processor Multi-level Feedback with Affinity Scheduling

Processor 1

Processor 2

Processor 3
Summary

• Scheduling policy: what to do next, when there are multiple threads ready to run
• Response time, throughput, predictability
• Uniprocessor policies
  – FIFO, Shortest Job First
  – round robin
  – multilevel feedback as approximation of optimal
• Multiprocessor policies