CS 423
Operating System Design: Scheduling

Tianyin Xu

* Thanks for Prof. Adam Bates for the slides.
Scheduling

• A forever topic in Computer Systems and Life
  • Uniprocessor: 100 threads in the ready queue – which one to run next?
  • Multiprocessor: 400 threads in the ready queues of four cores – which one to run next on which core?
  • Cluster: 1000 MapReduce jobs – which one to run on which machine and on which core?
  • Datacenters: 10000 user request – which one to run on which datacenter on which cluster on which machine?
More complexity

• Jobs/requests are not created equal.
  • Some are more important than the others
• Jobs/requests could have deadlines
  • Finishing late means nothing but wasting resources.
• Jobs/requests have constraints
  • Affinity is important – same node and same PCIe switch for GPUs
• Workloads could be very different.
Scheduling

- Always an active research topic
  - Everyone wants run more jobs with less resources

- In this class, we are going to focus on the simplest setup – a uniprocessor
What Are Scheduling Goals?

• What are the goals of a scheduler?

• Scheduling Goals:
  ■ Generate illusion of concurrency
  ■ Maximize resource utilization (e.g., mix CPU and I/O bound processes appropriately)
  ■ Meet needs of both I/O-bound and CPU-bound processes
    ■ Give I/O-bound processes better interactive response
    ■ Do not starve CPU-bound processes
  ■ Support Real-Time (RT) applications
Definitions

- **Task/Job**
  - Something that needs CPU time: a thread associated with a process or with the kernel…
  - … a user request, e.g., mouse click, web request, shell command, …

- **Latency/response time**
  - How long does a task take to complete?

- **Throughput**
  - How many tasks can be done per unit of time?
Definitions

• Overhead
  • How much extra work is done by the scheduler?

• Fairness
  • How equal is the performance received by different users?

• Predictability
  • How consistent is the performance over time?

• Starvation
  • A task ‘never’ receives the resources it needs to complete
  • Not very fair: - (}
Definitions

• **Workload**
  - Set of tasks for system to perform

• **Work-conserving**
  - Resource is used whenever there is a task to run
  - For non-preemptive schedulers, work-conserving is not always better
Definitions

- **Non-preemptive scheduling:**
  - The running process keeps the CPU until it *voluntarily* gives up the CPU
  - process exits
  - switches to blocked state
  - 1 and 4 only (no 3)

- **Preemptive scheduling:**
  - The running process can be interrupted and must release the CPU (can be *forced* to give up CPU)
Definitions

• Scheduling algorithm
  • takes a workload as input
  • decides which tasks to do first
  • Performance metric (throughput, latency) as output
  • Only preemptive, work-conserving schedulers to be considered
• Schedule tasks in the order they arrive
  • Continue running them until they complete or give up the processor

• On what workloads would FIFO be particularly bad?
Shortest Job First (SJF)

• Always do the task that has the shortest remaining amount of work to do
  • Often called Shortest Remaining Time First (SRTF)

• Suppose we have five tasks arrive one right after each other, but the first one is much longer than the others
  • Which completes first in FIFO? Next?
  • Which completes first in SJF? Next?
FIFO vs. SJF

FIFO

Tasks

(1) 
(2) 
(3) 
(4) 
(5) 

SJF

Tasks

(1) 
(2) 
(3) 
(4) 
(5) 

Time
Round Robin (RR)

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

• Characteristics of scheduler change depending on the time quantum size
  • What if time quantum is too short?
    • One instruction?
  • What if time quantum is too long?
    • Infinite?
Round Robin (100 ms time slice)

Tasks

(1)  
(2)  
(3)  
(4)  
(5)  

Rest of Task 1

Time
Round Robin

Round Robin (100 ms time slice)

Tasks

(1)  
(2)  
(3)  
(4)  
(5)  

Rest of Task 1

Time

FIFO

Tasks

(1)  
(2)  
(3)  
(4)  
(5)  

CS 423: Operating Systems Design
Round Robin

Round Robin (1 ms time slice)

<table>
<thead>
<tr>
<th>Tasks</th>
<th>Rest of Task 1</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>
Round Robin

Round Robin (1 ms time slice)

Time

Tasks

(1)          Rest of Task 1
(2)
(3)
(4)
(5)

Tasks

(1)
(2)
(3)
(4)
(5)

SJF
Scheduling

- Basic scheduling algorithms
  - FIFO (FCFS)
  - Shortest job first
  - Round Robin
Scheduling

- Basic scheduling algorithms
  - FIFO (FCFS)
  - Shortest job first
  - Round Robin

- What is an optimal algorithm in the sense of maximizing the number of jobs finished (i.e., minimizing average response time)?
FIFO vs. SJF

FIFO

wait time for 2, 3, 4, 5 is BIG!

SJF

wait time for 2, 3, 4, 5 is SMALL!
Basic scheduling algorithms
- FIFO (FCFS)
- Shortest job first
- Round Robin

Assuming zero-cost to time slicing, is Round Robin always better than FIFO?
RR v. FIFO (fixed size tasks)

**Round Robin (1 ms time slice)**

- Tasks 1, 2, 3, 4, 5...

**FIFO and SJF**

- Tasks 1, 2, 3, 4, 5...

**Time**
Suppose you want to compare two scheduling algorithms

- Create some infinite sequence of arriving tasks
- Start measuring
- Stop at some point
- Compute average response time as the average for completed tasks between start and stop

Is this valid or invalid?
Sample Bias Solutions

• Measure for long enough that # of completed tasks >> # of uncompleted tasks
  • For both systems!

• Start and stop system in idle periods
  • Idle period: no work to do
  • If algorithms are work-conserving, both will complete the same tasks
Is Round Robin the fairest possible algorithm?

What is fair?

- FIFO?
- Equal share of the CPU?
- What if some tasks don’t need their full share?
- Minimize worst case divergence?
- Time task would take if no one else was running
- Time task takes under scheduling algorithm
Fairness needs to be defined.

- 4 kids share a cake.
  - Each gets 25% of the cake.
  - Quite fair!

- There is one little kid and the kid can only eat 10% of the cake.
  - We either force her to eat the 25% -- to be fair
  - Or we give 15% remaining to the other 3 kids.
    - Min-max fairness
Max-Min Fairness

• The *least* demanding one *will* get its fair share *first*

• After this, the *next least* demanding one *will* get its fair share *first*

• And so on...
Max-Min Fairness

- Kid 1: 20%
- Kid 2: 26%
- Kid 3: 40%
- Kid 4: 50%

- 100% -> 25% each kid
  - 20% -> 5% left -> 1.6666666% to the other three
    - 25%
    - 25%
    - 25%
Max-Min Fairness

• Kid 1: 20%
• Kid 2: 26%
• Kid 3: 40%
• Kid 4: 50%

• 100% -> 25% each kid
  • 20%
  • 26%
  • 27%
  • 27%
Max-Min Fairness

• How do we balance a mixture of repeating tasks?
  • Some I/O bound, need only a little CPU
  • Some compute bound, can use as much CPU as they are assigned

• One approach: maximize the minimum allocation given to a task
  • If any task needs less than an equal share, schedule the smallest of these first
  • Split the remaining time using max-min
  • If all remaining tasks need at least equal share, split evenly
Mixed Workloads??

Tasks

I/O Bound

Issues I/O Request

I/O Completes

I/O Completes

Issues I/O Request

CPU Bound

Time
Multi-Level Feedback Queue

• **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)
Multi-Level Feedback Queue

- 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
<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><img src="image" alt="Queue" /></td>
</tr>
<tr>
<td>2</td>
<td>20</td>
<td><img src="image" alt="Queue" /></td>
</tr>
<tr>
<td>3</td>
<td>40</td>
<td><img src="image" alt="Queue" /></td>
</tr>
<tr>
<td>4</td>
<td>80</td>
<td><img src="image" alt="Queue" /></td>
</tr>
</tbody>
</table>

- **New or I/O Bound Task**
- **Time Slice Expiration**
Summary

- 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 is pessimal in terms of variance in response time.
Summary

• 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 and I/O benefit from SJF and can do poorly under Round Robin.
• Max-Min fairness can improve response time for I/O-bound tasks.
• Round Robin and Max-Min fairness both avoid starvation.
• By manipulating the assignment of tasks to priority queues, an MFQ scheduler can achieve a balance between responsiveness, low overhead, and fairness.
• Is MFQ optimally fair??