CS140 Project 1: Threads

Fall 2015

*Based off Winter 14 review slides*
overview

- git / setup
- alarm clock
- priority scheduler
- advanced scheduler
- general advice
git basics

git add

git commit

git pull

git push

* Lots of git tutorials and help available online

git merge

git branch
setting up repo

- Don’t use a public repo
- Suggest you set up a repo on your AFS space

```bash
$ git clone --bare http://cs140.stanford.edu/pintos.git

// for each group member
$ fsr sa pintos.git <sunet_id> all

// each group member (including owner of common repo)
$ cd <desired location of pintos work tree>

$ git clone <path to pinots.git clone>
// i.e. ‘/afs/ir.stanford.edu/users/c/o/conoreby/140/pintos.git’
```
project overview

- project documentation will guide you through starting with pintos
- work in two dirs: threads and devices

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>devices/timer.c</td>
<td>42</td>
<td>++++</td>
</tr>
<tr>
<td>threads/fixed-point.h</td>
<td>120</td>
<td>-------------------</td>
</tr>
<tr>
<td>threads/synch.c</td>
<td>88</td>
<td>++++++++</td>
</tr>
<tr>
<td>threads/thread.c</td>
<td>196</td>
<td>++++++++++++++++++++</td>
</tr>
<tr>
<td>threads/thread.h</td>
<td>23</td>
<td>+++</td>
</tr>
</tbody>
</table>

5 files changed, 440 insertions(+), 29 deletions(-)
Alarm clock

- `timer_sleep(num_ticks)`
  - original implementation just busy waits
  - **You need to:**
    - put the thread to sleep for `num_ticks`
    - wake thread up after `num_ticks` (put on ready queue)
  - `timer_usleep()`, `timer_msleep()`, etc call `timer_sleep()` and do not need to be modified
- Pay attention to whether or not you are in an interrupt context
Alarm clock

- **Disabling Interrupts** Turns off thread preemption, so only one thread can run.
  - only should be used when necessary and for the least amount of time necessary

- When should interrupts be disabled?
  - you should never sleep within an interrupt context
  - pay attention to whether or not you are in an interrupt context
Priority Scheduler

- current scheduler does round robin
- **You need to:** implement a priority scheduler
  - thread with highest priority is always running (unless it is blocked)
  - highest priority waiting thread should be woken first
# Priority scheduler

## Basic scenario.

<table>
<thead>
<tr>
<th>Ready</th>
<th>Running</th>
<th>Blocked</th>
</tr>
</thead>
<tbody>
<tr>
<td>A 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C 4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E 10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>20</td>
<td></td>
</tr>
</tbody>
</table>
### Priority scheduler

#### Basic scenario.

<table>
<thead>
<tr>
<th>Ready</th>
<th>Running</th>
<th>Blocked</th>
</tr>
</thead>
<tbody>
<tr>
<td>A 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C 4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E 10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>G 20</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- E Tries to acquire lock
## Priority scheduler

### Basic scenario.

<table>
<thead>
<tr>
<th>Ready</th>
<th>Running</th>
<th>Blocked</th>
</tr>
</thead>
<tbody>
<tr>
<td>A 2</td>
<td>C 4</td>
<td>F 2</td>
</tr>
<tr>
<td>B 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D 1</td>
<td></td>
<td>G 20</td>
</tr>
</tbody>
</table>

The diagram illustrates the states of tasks in a priority scheduler scenario. Tasks are categorized into Ready, Running, and Blocked. The numbers inside the rectangles represent priorities.
Priority scheduler

Another Basic scenario.

<table>
<thead>
<tr>
<th>Ready</th>
<th>Running</th>
<th>Blocked</th>
</tr>
</thead>
<tbody>
<tr>
<td>A 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C 4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E 10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F 2</td>
<td></td>
<td>Waiting on</td>
</tr>
<tr>
<td>G 20</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Priority scheduler

Another Basic scenario.

<table>
<thead>
<tr>
<th>Ready</th>
<th>Running</th>
<th>Blocked</th>
</tr>
</thead>
<tbody>
<tr>
<td>A 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C 4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E 10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>G 20</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Releases

Waiting on
Priority scheduler

Another Basic scenario.

<table>
<thead>
<tr>
<th>Ready</th>
<th>Running</th>
<th>Blocked</th>
</tr>
</thead>
<tbody>
<tr>
<td>A 2</td>
<td>B 3</td>
<td></td>
</tr>
<tr>
<td>C 4</td>
<td>D 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A releases B, B releases C, C releases D, D releases E, E releases F, F releases G.

Waiting on G.
Priority scheduler

Another Basic scenario.

<table>
<thead>
<tr>
<th>Ready</th>
<th>Running</th>
<th>Blocked</th>
</tr>
</thead>
<tbody>
<tr>
<td>E 10</td>
<td>D 1</td>
<td></td>
</tr>
<tr>
<td>C 4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>G 20</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Priority scheduler

Priority inversion

<table>
<thead>
<tr>
<th>Ready</th>
<th>Running</th>
<th>Blocked</th>
</tr>
</thead>
</table>

A 1
B 2
C 3
Priority scheduler

Priority inversion

<table>
<thead>
<tr>
<th>Ready</th>
<th>Running</th>
<th>Blocked</th>
</tr>
</thead>
</table>

A
B
C

Tries to acquire
Priority scheduler

Priority inversion

<table>
<thead>
<tr>
<th>Ready</th>
<th>Running</th>
<th>Blocked</th>
</tr>
</thead>
<tbody>
<tr>
<td>A 1</td>
<td>B 2</td>
<td>C 3</td>
</tr>
</tbody>
</table>

- A: Ready
- B: Running
- C: Blocked
Priority scheduler

Priority inversion

<table>
<thead>
<tr>
<th>Ready</th>
<th>Running</th>
<th>Blocked</th>
</tr>
</thead>
</table>

Problem!
Priority scheduler

Solution: Priority Donation (C donates its priority to A until A releases the lock)
Priority scheduler

Solution: Priority Donation (C donates its priority to A until A releases the lock)
Priority scheduler

Solution: Priority Donation (C donates its priority to A until A releases the lock)
Priority Donation

- **Multiple Donation**
  - a thread is holding two locks and different higher priority threads are waiting on each lock
    - take the maximum donated priority

- **Nested Donation**
  - A high priority thread is waiting on a lower priority thread to release a lock which is in turn waiting on an even lower priority thread to release a lock
    - donate the effective priority through

- Remember to reset the priority correctly after unblocking the higher priority threads
Priority Donation

- **void thread_set_priority** (int new_priority)
  - sets the current thread’s priority to new_priority. Yields if the thread no longer has the highest priority

- **int thread_get_priority ()**
  - Returns the current thread’s priority. With priority donation returns the higher (donated) priority
Advanced Scheduler

- No priority donation
- add -mlfqs kernel option
  - your code must allow us to choose a scheduling algorithm at startup time
    - add to parse_options()
- Priority is calculated, and not set by the thread itself
  - Priority argument in thread_create() and calls to thread_set_priority() should be ignored
  - thread_get_priority() returns the priority calculated by the scheduler
Advanced Scheduler

- mlfqs = multilevel feedback queue scheduler
  - scheduler maintains multiple ready queues
  - threads in the same queue have the same priority
  - pick the queue with the highest priority threads and round robin schedule them
  - priorities are updated periodically
    - need to make sure that priorities are updated before scheduling decisions are made
Advanced Scheduler

- Priority is calculated based off of certain metrics maintained
  - niceness
    - 0 -> does not change thread priority
    - >0 -> decreases the priority of a thread (max of 20)
    - <0 -> tends to take CPU time from other threads (min of -20)
    - initial thread starts with nice value of 0
    - other threads inherit nice value from their parent thread
  - recent_cpu
  - load_avg
Advanced Scheduler

- Priorities range from 0 to 63
  - calculated every 4th tick
  - recalculate priority for every thread where recent_cpu has changed

priority = PRI_MAX - (recent_cpu / 4) - (nice * 2)

recent_cpu = (2*load_avg) / (2*load_avg + 1) * recent_cpu + nice

load_avg = (59/60)*load_avg + (1/60)*ready_threads
Advanced Scheduler

- `int thread_get_recent_cpu()`
  - Returns 100 times the current thread's recent_cpu value, rounded to the nearest integer.

- `int thread_get_load_avg()`
  - Returns 100 times the current system load average, rounded to the nearest integer.

- Pay attention to when you should update these values.

- `int thread_get_nice()`
  - Returns the current thread's nice value.

- `void thread_set_nice(int new_nice)`
  - Sets the current thread's nice value to new_nice and recalculates the thread's priority based on the new value. If the running thread no longer has the highest priority, yields.
Advanced Scheduler

- Fixed-Point arithmetic
  - recent_cpu and load_avg are real numbers
  - Pintos **does not** support floating point already
  - You need to:
    - implement floating point arithmetic to calculate recent_cpu and load_avg
General tips

- Start early
- Use version control (git)
- Design before you code
- Don’t rush the Design Doc
- Integrate team code early and often
- Match code style (don’t lose easy points)
Questions?