cs 140 project 1: threads

9 January 2015
The basics:

- `git clone`
- `git add`
- `git commit`
- `git branch`
- `git merge`
- `git stash`
- `git pull`
- `git push`
- `git rebase`
Some guidelines & ideas:

- Write helpful commit and stash messages. They exist only for you and your team!

Read or skim Pro Git\(^1\) for fuller advice.

\(^1\)http://git-scm.com/book/en/v2
Some guidelines & ideas:

- Write helpful commit and stash messages. They exist only for you and your team!
- Host your code on Github or Bitbucket as a “master” copy. **Use a private repository!**

Read or skim Pro Git\(^1\) for fuller advice.

---

\(^1\)http://git-scm.com/book/en/v2
Some guidelines & ideas:

▶ Write helpful commit and stash messages. They exist only for you and your team!

▶ Host your code on Github or Bitbucket as a “master” copy. **Use a private repository!**

▶ Create per-assignment branches. Work on topic branches; merge into assignment branches and delete once the topic is “done”.

Read or skim Pro Git\(^1\) for fuller advice.

\(^1\)http://git-scm.com/book/en/v2
Some guidelines & ideas:

▶ Write helpful commit and stash messages. They exist only for you and your team!

▶ Host your code on Github or Bitbucket as a “master” copy. **Use a private repository!**

▶ Create per-assignment branches. Work on topic branches; merge into assignment branches and delete once the topic is “done”.

▶ Stay synchronized with your team: fetch and push often.

Read or skim Pro Git\(^1\) for fuller advice.

---

Some guidelines & ideas:

▶ Write helpful commit and stash messages. They exist only for you and your team!

▶ Host your code on Github or Bitbucket as a “master” copy. **Use a private repository!**

▶ Create per-assignment branches. Work on topic branches; merge into assignment branches and delete once the topic is “done”.

▶ Stay synchronized with your team: fetch and push often.

▶ Commit often. Use `git bisect` to find regression bugs.

Read or skim [Pro Git](http://git-scm.com/book/en/v2) for fuller advice.
Synchronization

Serializing access to shared resource.
Synchronization

Serializing access to shared resource.

Disabling interrupts  Turns off thread preëmption, so only one thread can run. Undesirable unless absolutely necessary.
Synchronization

Serializing access to shared resource.

**Disabling interrupts**  Turns off thread preemption, so only one thread can run. Undesirable unless absolutely necessary.

**Synchronization primitives**  In threads/synch.h
  - Semaphores
  - Locks
  - Condition variables
Thread basics

```c
struct thread
{
    tid_t tid;
    enum thread_status status;
    char name[16];
    uint8_t *stack;
    int priority;
    struct list_elem allelem;
    struct list_elem elem;

    #ifdef USERPROG
    uint32_t *pagedir;
    #endif

    unsigned magic;
};
```
Implementing void timer_sleep (int64_t ticks)
Implementing `void timer_sleep (int64_t ticks)`

- Remove busy waiting implementation.
Implementing `void timer_sleep (int64_t ticks)`

- Remove busy waiting implementation.
- What to do with a `struct thread` if you don’t want to touch it again until after time passes?
Replace round-robin scheduler with a priority-based scheduler.

Key points:

- Most code will be in thread.
- When scheduling, pick the highest priority thread.
- When lowering thread’s priority, it should yield if another thread has higher priority.
- When a higher priority thread wakes up from alarm clock or a lock, it should preempt the current thread.
Replace round-robin scheduler with a priority-based scheduler.

Key points:

- Most code will be in thread. [hc].
Priority scheduling

Replace round-robin scheduler with a priority-based scheduler.

Key points:

- Most code will be in thread. [hc].
- When scheduling, pick the highest priority thread.
Replace round-robin scheduler with a priority-based scheduler. Key points:

- Most code will be in thread $[hc]$. 
- When scheduling, pick the highest priority thread.
- When lowering thread’s priority, it should yield if another thread has higher priority.
Priority scheduling

Replace round-robin scheduler with a priority-based scheduler. Key points:

- Most code will be in thread. [hc].
- When scheduling, pick the highest priority thread.
- When lowering thread’s priority, it should yield if another thread has higher priority.
- When a higher priority thread wakes up from alarm clock or a lock, it should preempt the current thread.
Priority inversion

If the lowest priority thread holds a lock that a high priority thread wants, the high priority thread blocks until every other thread finishes running.

Solution: **priority donation**. Things to consider:

- To how many threads can a donor donate its priority? From how many threads may a donee receive priority?
- What happens when a priority recipient donates to another thread?
Advanced scheduler

- BSD scheduler computes thread CPU usage statistics to calculate thread priorities.
- `thread_set_priority` ignored in BSD scheduler mode.
- No priority donation.
- Will require you to write a simple fixed-point arithmetic library.
- Global Boolean variable `thread_mlfqs` indicates which mode to use.

Fullest information available in Pintos handbook.²

Advanced scheduler

- BSD scheduler computes thread CPU usage statistics to calculate thread priorities.

Fullest information available in Pintos handbook.²

Advanced scheduler

- BSD scheduler computes thread CPU usage statistics to calculate thread priorities.
- `thread_set_priority` ignored in BSD scheduler mode.

Fullest information available in Pintos handbook.²

Advanced scheduler

- BSD scheduler computes thread CPU usage statistics to calculate thread priorities.
- `thread_set_priority` ignored in BSD scheduler mode.
- No priority donation.

Fullest information available in Pintos handbook.²

Advanced scheduler

- BSD scheduler computes thread CPU usage statistics to calculate thread priorities.
- `thread_set_priority` ignored in BSD scheduler mode.
- No priority donation.
- Will require you to write a simple fixed-point arithmetic library.

Fullest information available in Pintos handbook.²

²[http://www.scs.stanford.edu/15wi-cs140/pintos/pintos_7.html]
Advanced scheduler

- BSD scheduler computes thread CPU usage statistics to calculate thread priorities.
- `thread_set_priority` ignored in BSD scheduler mode.
- No priority donation.
- Will require you to write a simple fixed-point arithmetic library.
- Global Boolean variable `thread_mlfqs` indicates which mode to use.

Fullest information available in Pintos handbook.²

Match existing code style; don’t stick out. Debug with `gdb`, not with `printf`. `bochs` is reproducible. Use the "jitter" flag `-j` to generate alternate reproducible runs. Read the design document template first and work on it as you write code and debug. Design your solution, data structures, and synchronization scheme before writing code.
- Match existing code style; don’t stick out.
- Match existing code style; don’t stick out.
- Debug with `gdb`, not with `printf`. 
- Match existing code style; don’t stick out.
- Debug with gdb, not with printf.
- bochs is *reproducible*. Use the “jitter” flag -j to generate alternate reproducible runs.
• Match existing code style; don’t stick out.
• Debug with gdb, not with printf.
• bochs is *reproducible*. Use the “jitter” flag -j to generate alternate reproducible runs.
• Read the design document template *first* and work on it as you write code and debug.
Match existing code style; don’t stick out.

Debug with gdb, not with printf.

bochs is *reproducible*. Use the “jitter” flag `-j` to generate alternate reproducible runs.

Read the design document template *first* and work on it as you write code and debug.

Design your solution, data structures, and synchronization scheme *before* writing code.