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

git merge

git branch

* Lots of git tutorials and help available online

setting up repo

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

```
$ 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

```
devices/timer.c          |    42   +++++-
threads/fixed-point.h    |   120   ++++++++++++++++++
threads/synch.c          |    88   ++++++++++++-
threads/thread.c         |   196   ++++++++++++++++++++++++++++----
threads/thread.h         |    23   +++
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

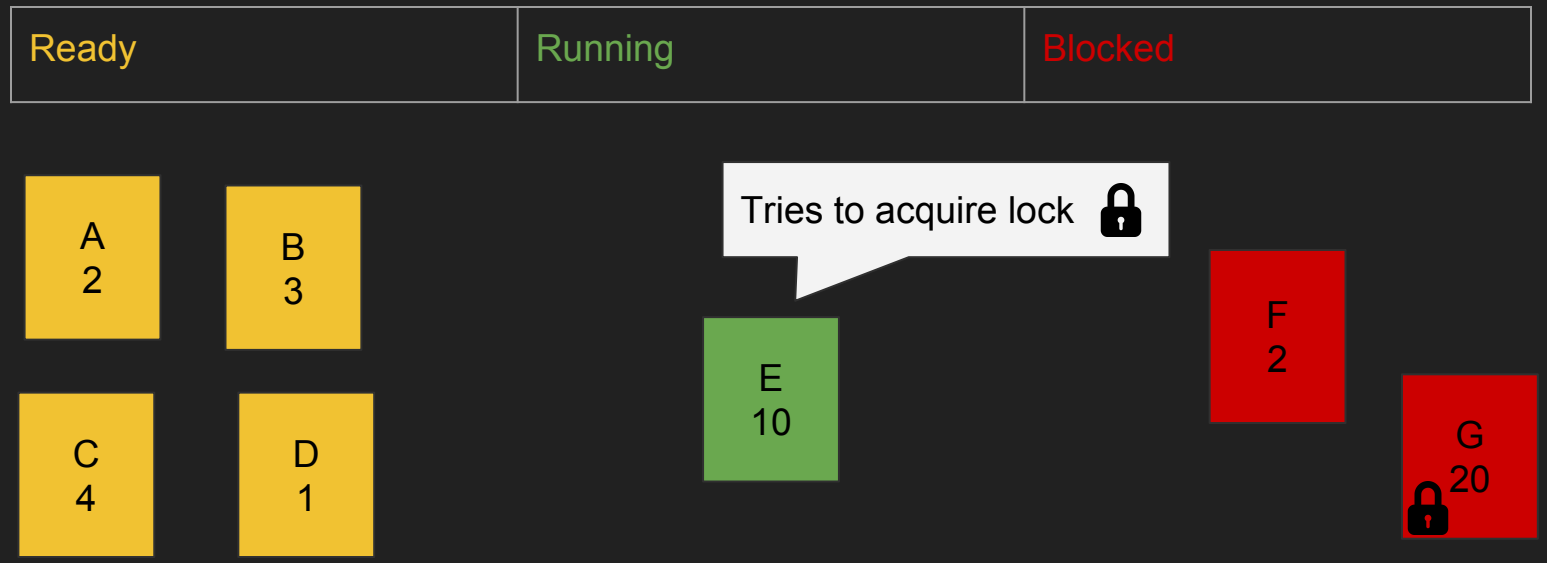
Priority scheduler

Basic scenario.

Ready	Running	Blocked
A 2 B 3 C 4 D 1	E 10	F 2 G 20

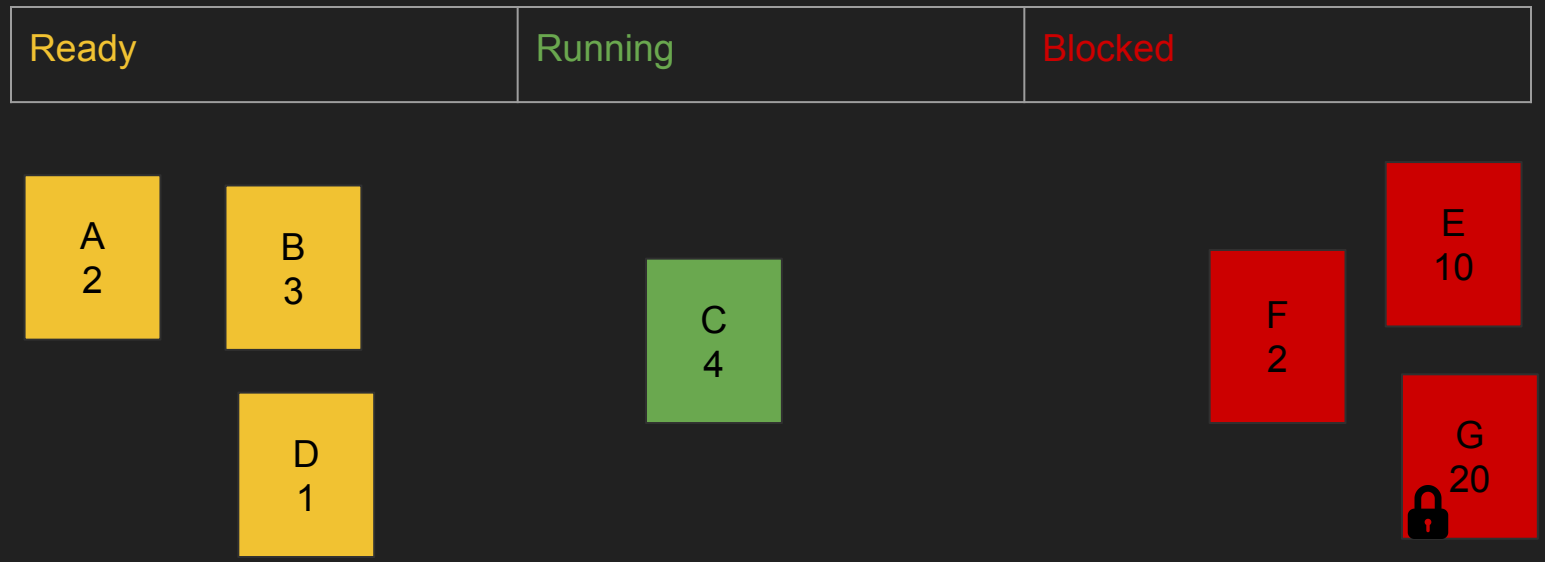
Priority scheduler

Basic scenario.



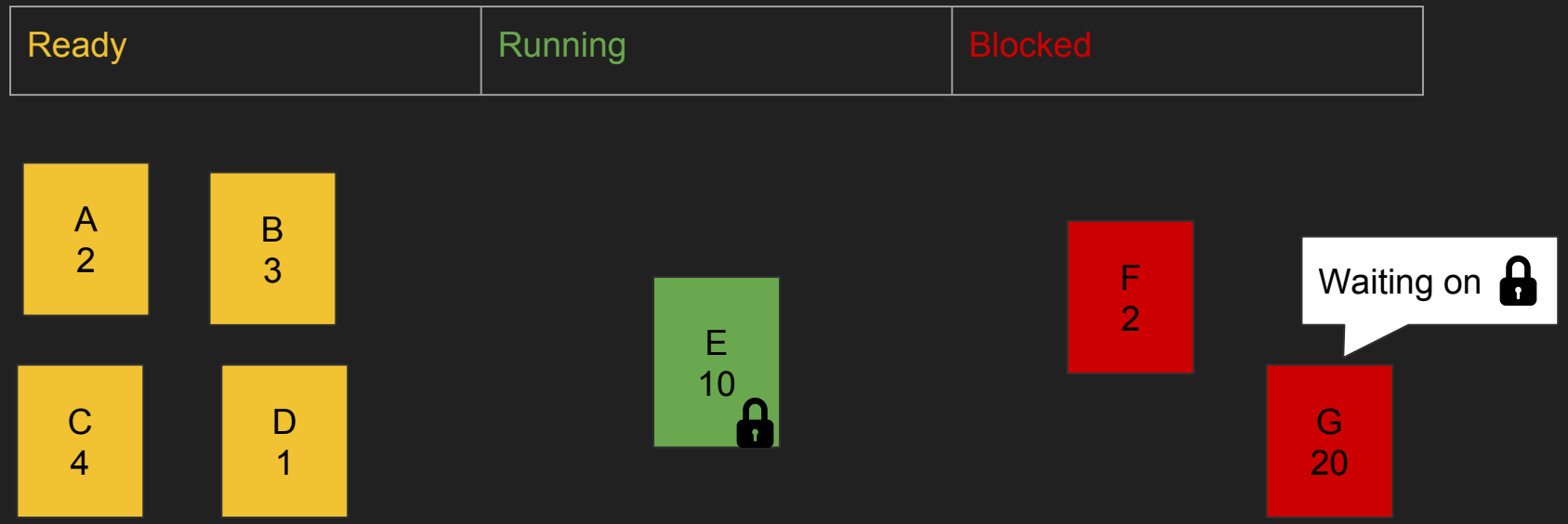
Priority scheduler

Basic scenario.



Priority scheduler

Another Basic scenario.



Priority scheduler

Another Basic scenario.

Priority scheduler

Another Basic scenario.

Priority scheduler

Another Basic scenario.

Ready	Running	Blocked
A 2 B 3 E 10 C 4 D 1	G 20	F 2

Priority scheduler

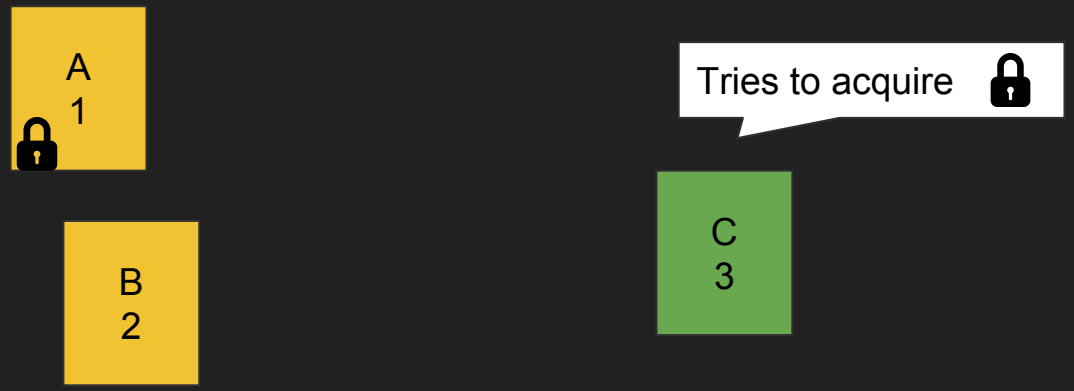
Priority inversion

Ready	Running	Blocked
A 1	C 3	
B 2		

Priority scheduler

Priority inversion

Ready	Running	Blocked



Priority scheduler

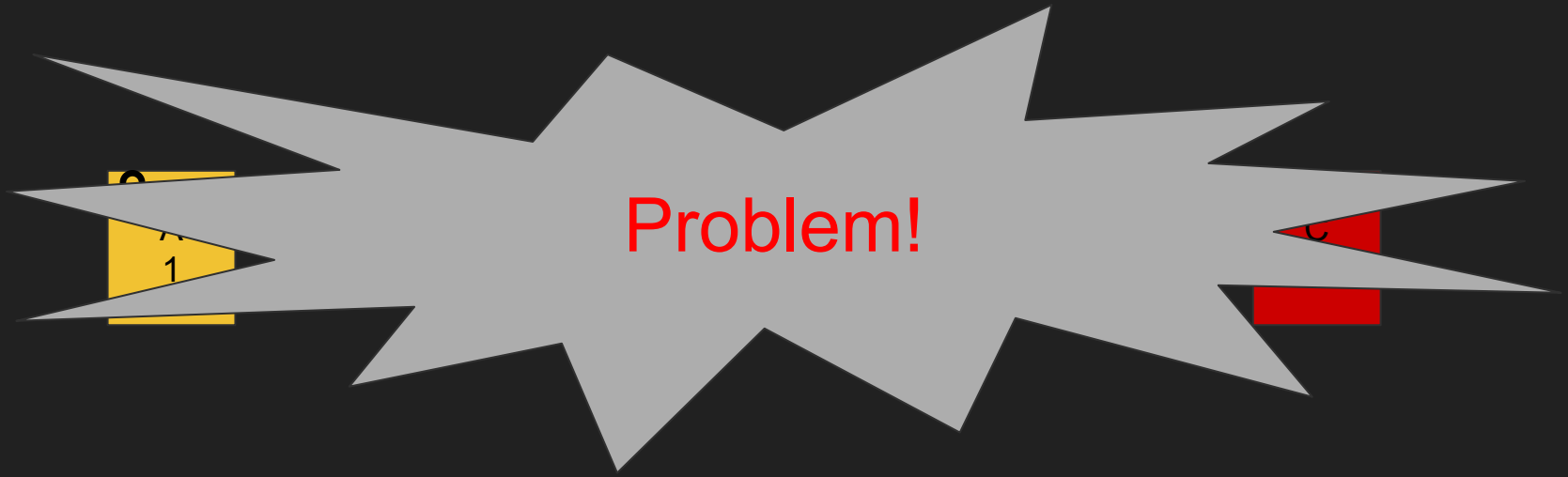
Priority inversion

Ready	Running	Blocked
A 1	B 2	C 3

Priority scheduler

Priority inversion

Ready	Running	Blocked



Priority scheduler

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

Ready	Running	Blocked
B 2	A (3)	C 3

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)

Ready	Running	Blocked

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?