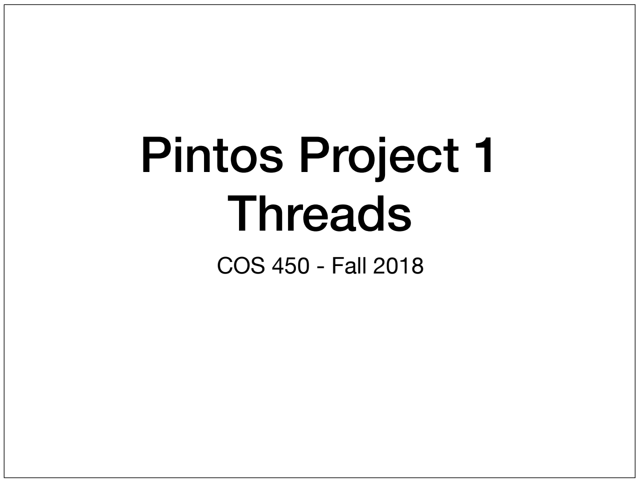
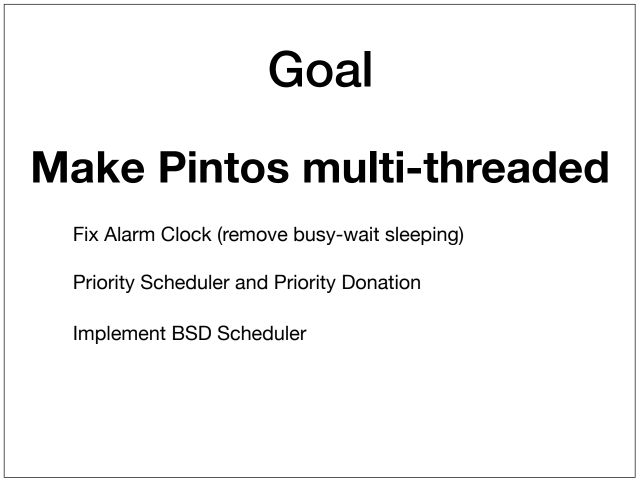
1



Pintos Project 1
Threads

COS 450 - Fall 2018

2



Goal

Make Pintos multi-threaded

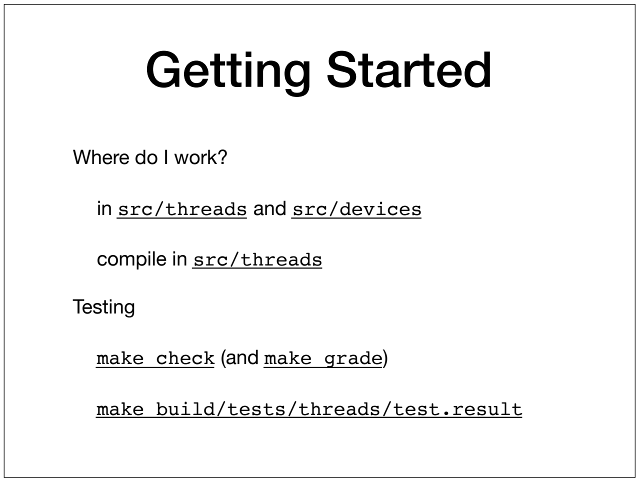
Fix Alarm Clock (remove busy-wait sleeping)

Priority Scheduler and Priority Donation

Implement BSD Scheduler

There are three fundamental parts to this project. Though the priority scheduler and priority donation can be done sequentially. They are (somewhat) in order of increasing difficulty.

3



Getting Started

Where do I work?

in `src/threads` and `src/devices`

compile in `src/threads`

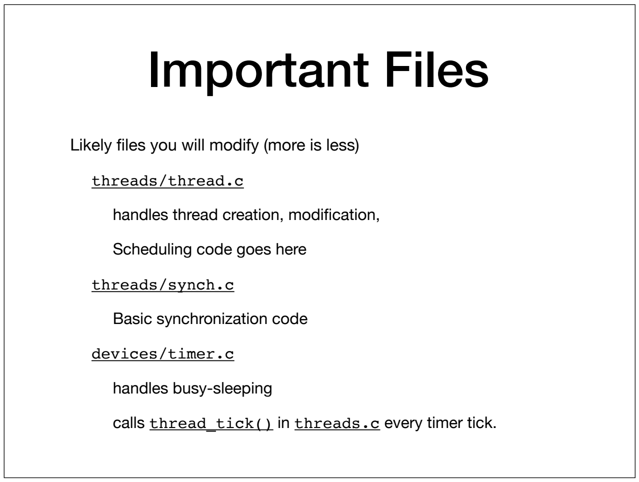
Testing

`make check` (and `make grade`)

`make build/tests/threads/test.result`

If you find yourself outside of these locations you might be getting off track. Most, if not all solutions, don’t require any code outside of threads and devices.

make grade will give you an idea of what score you will get on the code portion of the project. Run it multiple times to ensure you get consistent results.



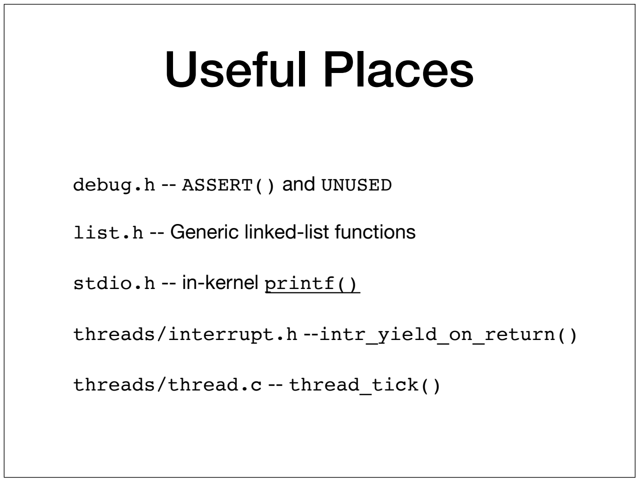
Important Files

Likely files you will modify (more is less)

- `threads/thread.c`
  - handles thread creation, modification,
  - Scheduling code goes here
- `threads/synch.c`
  - Basic synchronization code
- `devices/timer.c`
  - handles busy-sleeping
  - calls `thread_tick()` in `threads.c` every timer tick.

4

As with the locations, most solutions only need to modify these files. If you go too far outside of these (except for the BSD scheduler maybe) you might be off track.



Useful Places

`debug.h` -- `ASSERT()` and `UNUSED`

`list.h` -- Generic linked-list functions

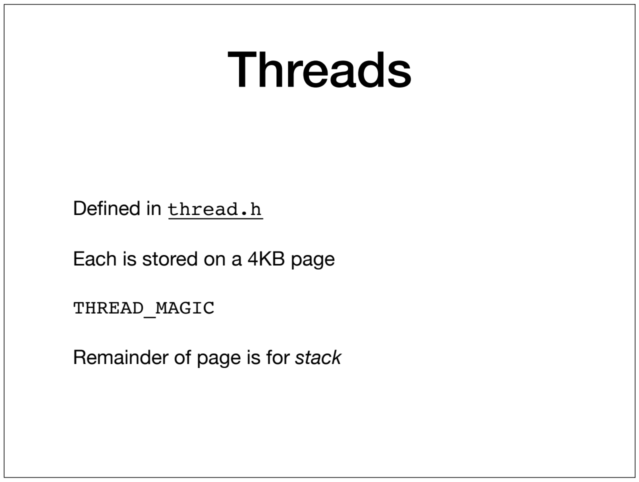
`stdio.h` -- in-kernel `printf()`

`threads/interrupt.h` --`intr_yield_on_return()`

`threads/thread.c` -- `thread_tick()`

5

In these locations you will find utility functions, defines, and routines that you should use in your code. Don't invent your own list when one already exists in the system. Learn how to use what's provided.



Threads

Defined in `thread.h`

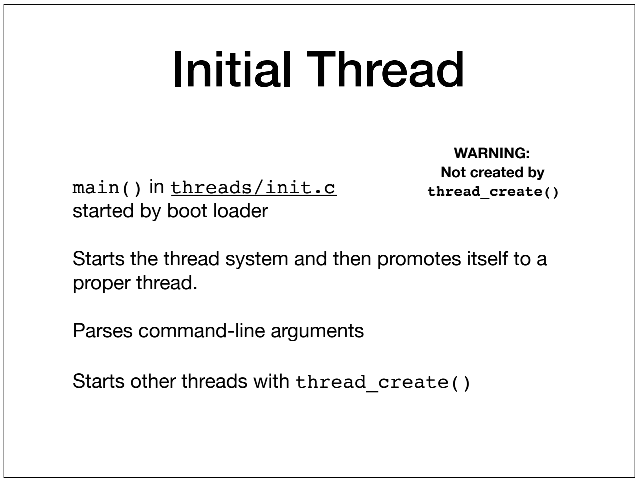
Each is stored on a 4KB page

`THREAD_MAGIC`

Remainder of page is for *stack*

6

The `thread` is the fundamental thing that Pintos manages, our text refers to it as a process. thread.h defines the structure that Pintos uses to maintain the state of a thread and all the related information it needs to manage it. We will be changing this in just about all our projects.



Initial Thread

WARNING:
Not created by
`thread_create()`

`main()` in `threads/init.c`
started by boot loader

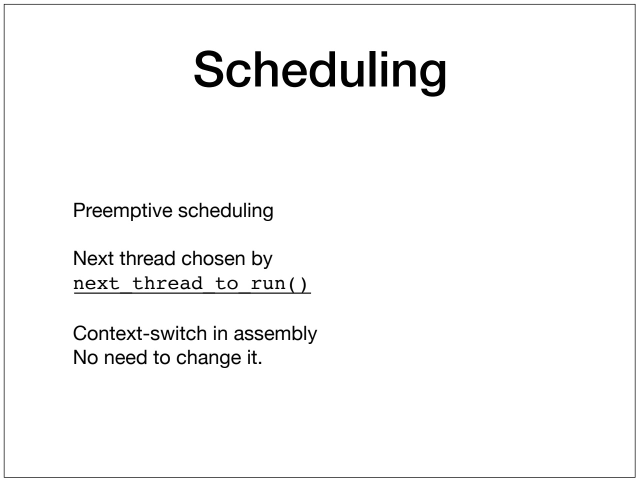
Starts the thread system and then promotes itself to a proper thread.

Parses command-line arguments

Starts other threads with `thread_create()`

7

The first thread in the system is special. It’s not created by thread_create(). thread_create() effectively clones the current thread into a new one. When the system is starting there’s no thread to clone so Pintos has to fake it.



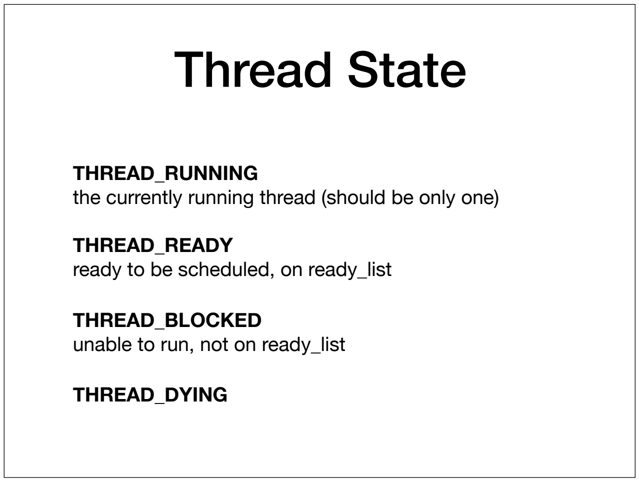
Scheduling

Preemptive scheduling

Next thread chosen by
`next_thread_to_run()`

Context-switch in assembly
No need to change it.

8



Thread State

THREAD_RUNNING
the currently running thread (should be only one)

THREAD_READY
ready to be scheduled, on ready_list

THREAD_BLOCKED
unable to run, not on ready_list

THREAD_DYING

9

Synchronization

Interrupt Disabling
Can affect performance, use sparingly

Semaphores

Locks

Monitors

NOTE:
Interrupt disabling is used in the kernel to synchronize interrupt handlers

10

Pintos provides several synchronization mechanisms already. You should not invent your own, use these to implement your solution.

Requirements

Non busy sleep (Alarm Clock)

Priority Scheduling

- Allow processes to modify their priority

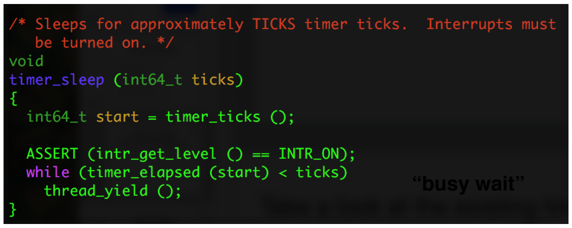
Priority Donation (for locks)

BSD Scheduler (fixed-point math)

11

Alarm Clock

Current state of affairs…



```
/* Sleeps for approximately TICKS timer ticks.  Interrupts must
   be turned on. */
void
timer_sleep (int64_t ticks)
{
  int64_t start = timer_ticks ();

  ASSERT (intr_get_level () == INTR_ON);
  while (timer_elapsed (start) < ticks)
    thread_yield ();
}
```

“busy wait”

12

Take a look at the existing timer_sleep() code in devices/timer.c

13

Alarm Clock

When a thread is sleeping,
it should not consume CPU time

`timer_sleep()…`

block the calling thread

allow other threads to execute

unblock after number of timer ticks

NOTE: multiple threads may call timer_sleep()

Take a look at the existing timer_sleep() code in devices/timer.c

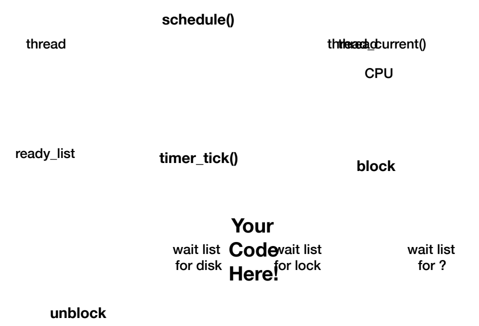
14

Alarm Clock

WARNING:
Part of your code
Will be in an
interrupt handler
pay close attention to
concurrent data access

15

Threads and Scheduling



schedule()

thread

thread thread_current()

CPU

ready_list

timer_tick()

block

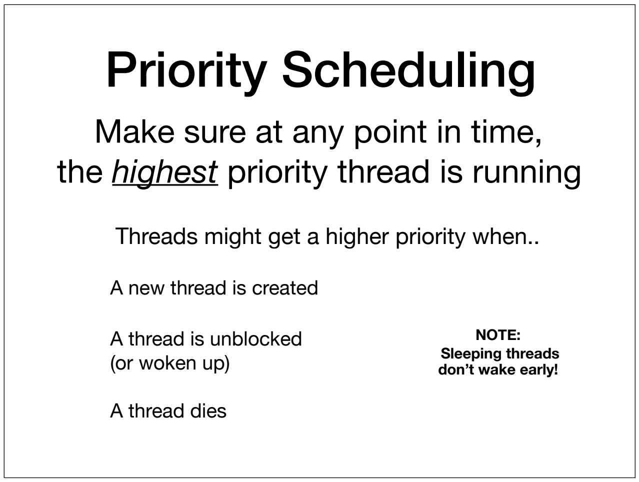
Your Code Here!

wait list for disk

wait list for lock

wait list for ?

unblock



16

Priority Scheduling

Make sure at any point in time,
the <u>*highest*</u> priority thread is running

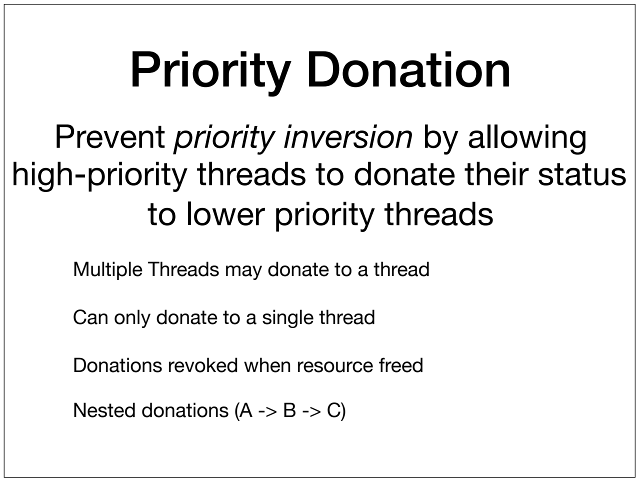
Threads might get a higher priority when..

A new thread is created

A thread is unblocked
(or woken up)

NOTE:
Sleeping threads
don't wake early!

A thread dies



17

Priority Donation

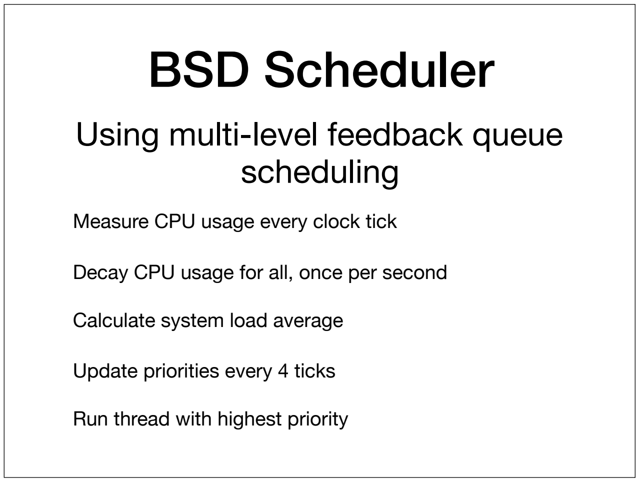
Prevent *priority inversion* by allowing high-priority threads to donate their status to lower priority threads

Multiple Threads may donate to a thread

Can only donate to a single thread

Donations revoked when resource freed

Nested donations (A -> B -> C)



18

BSD Scheduler

Using multi-level feedback queue scheduling

Measure CPU usage every clock tick

Decay CPU usage for all, once per second

Calculate system load average

Update priorities every 4 ticks

Run thread with highest priority

Implementation Order

19

Alarm Clock

Initial set/get priority()

Prioritized thread blocking and unblocking

Priority Scheduler

Priority Donation

Fixed-point math and BSD Scheduler

Tips from the after-code

20

Read, think, design, then code

Keep it simple, use lists

Avoid duplicating code

Verify errors and watch warnings

Keep context of code execution in mind
(interrupt, scheduler, other)

Don’t forget the **DESIGNDOC**

21

It’s **<u>50%</u>** of your grade

A few good hours here is worth more than the last 5% on the test suite!

End

Pintos Project 1