CS 5600
Computer Systems

Project 1: Threads in Pintos

- Getting Started With Pintos
- What does Pintos Include?
- Threads in Pintos
- Project 1

What is Pintos?

- Pintos is a teaching operating system from Stanford
  - Written in C
  - Implements enough functionality to boot…
    - … perform basic device I/O…
    - … and has a small standard library
- Your goal will be to expand it's functionality

Pintos Documentation

- All of the Pintos docs are available on the course webpage

http://www.ccs.neu.edu/home/skotthe/classes/cs5600/fall/2016/pintos/doc/pintos.html

- You will need to copy the Pintos source to your home directory
  - See Lab2

Pintos Projects

- Each project in this class corresponds to a particular directory

  Project 1: *pintos/src/threads/*

  Project 2: *pintos/src/userprog/*

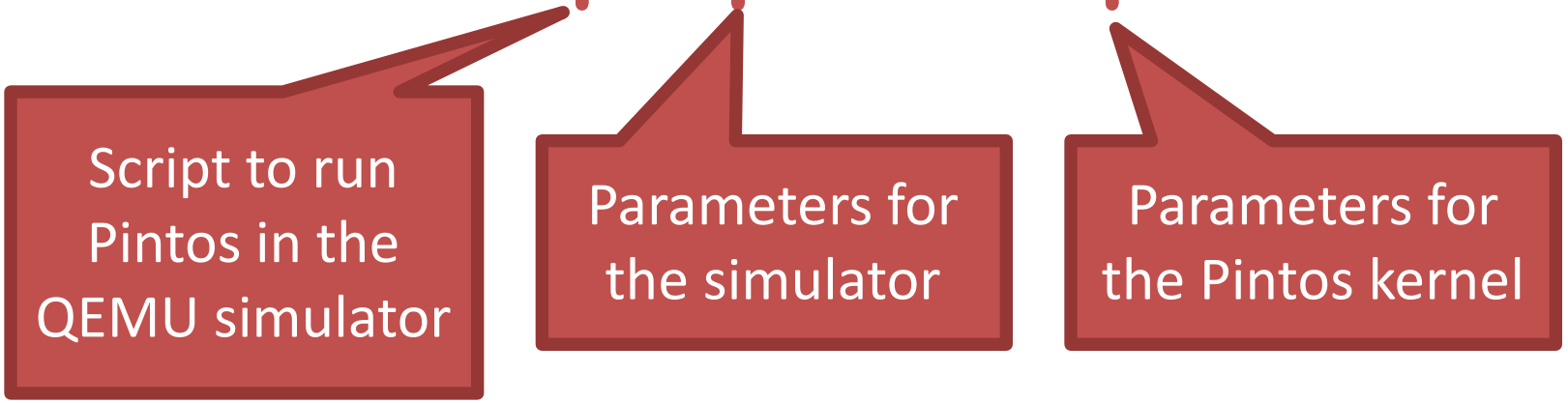
  Project 3: *pintos/src/vm/*

  Project 4: *pintos/src/filesys/*

- Each directory includes a Makefile, and all necessary files to build Pintos

Building and Running Pintos

```
$ cd ~/pintos/src/threads
$ make
$ cd build/
$ pintos  -v  --  -q run alarm-single
```



Making Pintos

- When you run *make*, you compile two things
  - build/loader.bin
    - The Pintos bootloader (512 byte MBR image)
    - Locates the kernel in the filesystem, loads it into memory, and executes it
  - build/kernel.bin
    - The Pintos kernel
- The *pintos* script automatically creates a file system image that includes the MBR and kernel

QEMU

- Pintos could be run on an actual machine
  - But that would require installing it, dual booting with another OS
  - Debugging would be hard
- Instead, we will run Pintos inside QEMU
  - QEMU is a machine emulator
    - In our case, a 32-bit x86 CPU with basic devices
  - Executes a BIOS, just like a real machine
    - Loads bootloader from MBR of emulated disk drive

- Getting Started With Pintos
- What does Pintos Include?
- Threads in Pintos
- Project 1

Pintos Features

- Pintos is already a basic, bootable OS
  - Switches from real to protected mode
  - Handles interrupts
  - Has a timer-interrupt for process preemption
  - Does basic memory management
  - Supports a trivial file system

Devices

- *pintos/src/devices/* includes drivers and APIs for basic hardware devices
  - System timer: *timer.h*
  - Video: *vga.h* (use *lib/kernel/stdio.h* to print text)
  - Serial port: *serial.h*
  - File storage: *ide.h*, *partition.h*, *block.h*
  - Keyboard input: *kbd.h*, *input.h*
  - Interrupt controller: *intq.h*, *pit.h*

Standard Library

- The typical C standard library is not available to you (C lib doesn't exist in Pintos)
- Pintos reimplements a subset of C lib in *pintos/src/lib/*
  - Variable types: *ctypes.h*, *stdbool.h*, *stdint.h*
  - Variable argument functions: *stdarg.h*
  - String functions: *string.h*
  - Utility functions: *stdlib.h*
  - Random number generation: *random.h*
  - Asserts and macros for debugging: *debug.h*

Data Structures

- *pintos/src/lib/kernel/* includes kernel data structures that you may use
  - Bitmap: *kernel/bitmap.h*
  - Doubly linked list: *kernel/list.h*
  - Hash table: *kernel/hash.h*
  - Console printf(): *kernel/stdio.h*
- Include using #include <kernel/whatever.h>

Tests

- Each Pintos project comes with a set of tests
  - Useful for debugging
  - Also what we will use to grade your code
- Out-of-the-box, Pintos cannot run user programs
  - Thus, tests are compiled into the kernel
  - You tell the kernel which test to execute on the command line

```
$ pintos  -v  --  run alarm-single
```

- Use `$ make check` to run the tests

```
[>] pintos -v -- -q run alarm-single
qemu -hda /tmp/8HDMnPzQrE.dsk -m 4 -net none -nographic -monitor null
PiLo hda1
Loading.....
Kernel command line: run alarm-single
Pintos booting with 4,088 kB RAM...
382 pages available in kernel pool.
382 pages available in user pool.
Calibrating timer...  523,468,800 loops/s.
Boot complete.
Executing 'alarm-single':
(alarm-single) begin
(alarm-single) Creating 5 threads to sleep 1 times each.
(alarm-single) Thread 0 sleeps 10 ticks each time,
…
(alarm-single) end
Execution of 'alarm-single' complete. Execution of 'alarm-single' complete.
Timer: 276 ticks
Thread: 0 idle ticks, 276 kernel ticks, 0 user ticks
Console: 986 characters output
Keyboard: 0 keys pressed
Powering off...
```

Pintos Bootup Sequence

- pintos/src/threads/init.c → main()

```
bss_init (); /* Clear the BSS */

argv = read_command_line ();
argv = parse_options (argv);

thread_init ();
console_init ();

printf ("Pintos booting with…");

/* Initialize memory system. */
palloc_init (user_page_limit);
malloc_init ();
paging_init ();

/* Segmentation. */
tss_init ();
gdt_init ();
```

```
/* Enable Interrupts */
intr_init ();

/* Timer Interrupt */
timer_init ();

/* Keyboard */
kbd_init ();
input_init ();
exception_init ();

/* Enable syscalls */
syscall_init ();

/* Initialize threading */
thread_start ();
serial_init_queue ();
timer_calibrate ();
```

```
/* Initialize the hard
   drive and fs */
ide_init ();
locate_block_devices ();
filesys_init (format_filesys);

printf ("Boot complete.\n");

/* Run actions specified
   on kernel command line. */
run_actions (argv);

shutdown ();
thread_exit ();
```

- Getting Started With Pintos
- What does Pintos Include?
- Threads in Pintos
- Project 1

Threads in Pintos

- Pintos already implements a simple threading system
  - Thread creation and completion
  - Simple scheduler based on timer preemption
  - Synchronization primitives (semaphore, lock, condition variable)
- But this system has problems:
  - Wait is based on a spinlock (i.e. it just wastes CPU)
  - The thread priority system is not implemented

Threading System

- thread_create() starts new threads
  - Added to *all_list* and *ready_list*
- Periodically, the timer interrupt fires
  - Current thread stops running
  - Timer interrupt calls `schedule()`

```
static void schedule (void) {
  struct thread *cur = running_thread ();
  struct thread *next = next_thread_to_run ();
  struct thread *prev = NULL;

  if (cur != next) prev = switch_threads (cur, next);
  thread_schedule_tail (prev);
}
```

Switching Threads

- Remember the switch() function we talked about earlier?
- Pintos has one in threads/switch.S
  - Saves the state of the CUR thread
  - Saves ESP of the CUR thread
  - Loads the state of the NEXT thread
  - Loads ESP of the NEXT thread
  - Returns to NEXT thread

Idle Thread

- There is always one thread in the system
- Known as the idle thread
  - Executes when there are no other threads to run

```
for (;;) {
  intr_disable (); /* Disable interrupts */
  thread_block (); /* Let another thread run */

  /* Re-enable interrupts and wait for the next one.
     The `sti' instruction disables interrupts until the
     completion of the next instruction, so these two
     instructions are executed atomically. */
  asm volatile ("sti; hlt" : : : "memory");
}
```

- Getting Started With Pintos
- What does Pintos Include?
- Threads in Pintos
- Project 1

Pintos Projects

- All four Pintos projects will involve two things
  1. Modifying the Pintos OS
  2. Producing a DESIGNDOC that explains your modifications
- We will use automated tests to gauge the correctness of your modified code
- The TA/graders will evaluate the quality of your DESIGNDOC
  - Templates for DESIGNDOCs are provided by us

Project 1 Goals

1. Fix the timer_sleep() function to use proper synchronization
   - No busy waiting
2. Implement the thread priority system
   - High priority threads execute before low priority
   - Watch out for priority inversion!

Goal 1: Fixing timer_sleep()

- Sometimes, a thread may want to wait for some time to pass, a.k.a. sleep
- Problem: Pintos' implementation of sleep is very wasteful
- devices/timer.c

```
void timer_sleep (int64_t ticks) {
    int64_t start = timer_ticks ();
    while (timer_elapsed (start) < ticks)
        thread_yield ();
}
```

Modifying timer_sleep()

```
void timer_sleep (int64_t ticks) {
   //int64_t start = timer_ticks ();
   // while (timer_elapsed (start) < ticks)
   //       thread_yield ();
   thread_sleep(ticks); // New function!
}
```

Modifying struct thread

- threads/thread.h

```
enum thread_status {
   THREAD_RUNNING,     /* Running thread. */
   THREAD_READY,       /* Not running but ready to run. */
   THREAD_SLEEPING,    /* New state for sleeping threads */
   THREAD_BLOCKED,     /* Waiting for an event to trigger. */
   THREAD_DYING        /* About to be destroyed. */
 };

struct thread {
   …
   int64_t wake_time;
}
```

thread_sleep()

- threads/thread.c

```
static struct list sleeping_list;

void thread_sleep (int64_t ticks) {
   struct thread *cur = thread_current();
   enum intr_level old_level;

   old_level = intr_disable ();
   if (cur != idle_thread) {
      list_push_back (&sleeping_list, &cur->elem);
      cur->status = THREAD_SLEEPING;
      cur->wake_time = timer_ticks() + ticks;
      schedule ();
   }
   intr_set_level (old_level);
}
```

Modifying schedule ()

- threads/thread.c

```
struct list_elem *temp, *e = list_begin (&sleeping_list);
int64_t cur_ticks = timer_ticks();

while (e != list_end (&sleeping_list)) {
   struct thread *t = list_entry (e, struct thread, allelem);

   if (cur_ticks >= t->wake_time) {
      list_push_back (&ready_list, &t->elem); /* Wake this thread up! */
      t->status = THREAD_READY;
      temp = e;
      e = list_next (e);
      list_remove(temp); /* Remove this thread from sleeping_list */
   }
   else e = list_next (e);
}
```

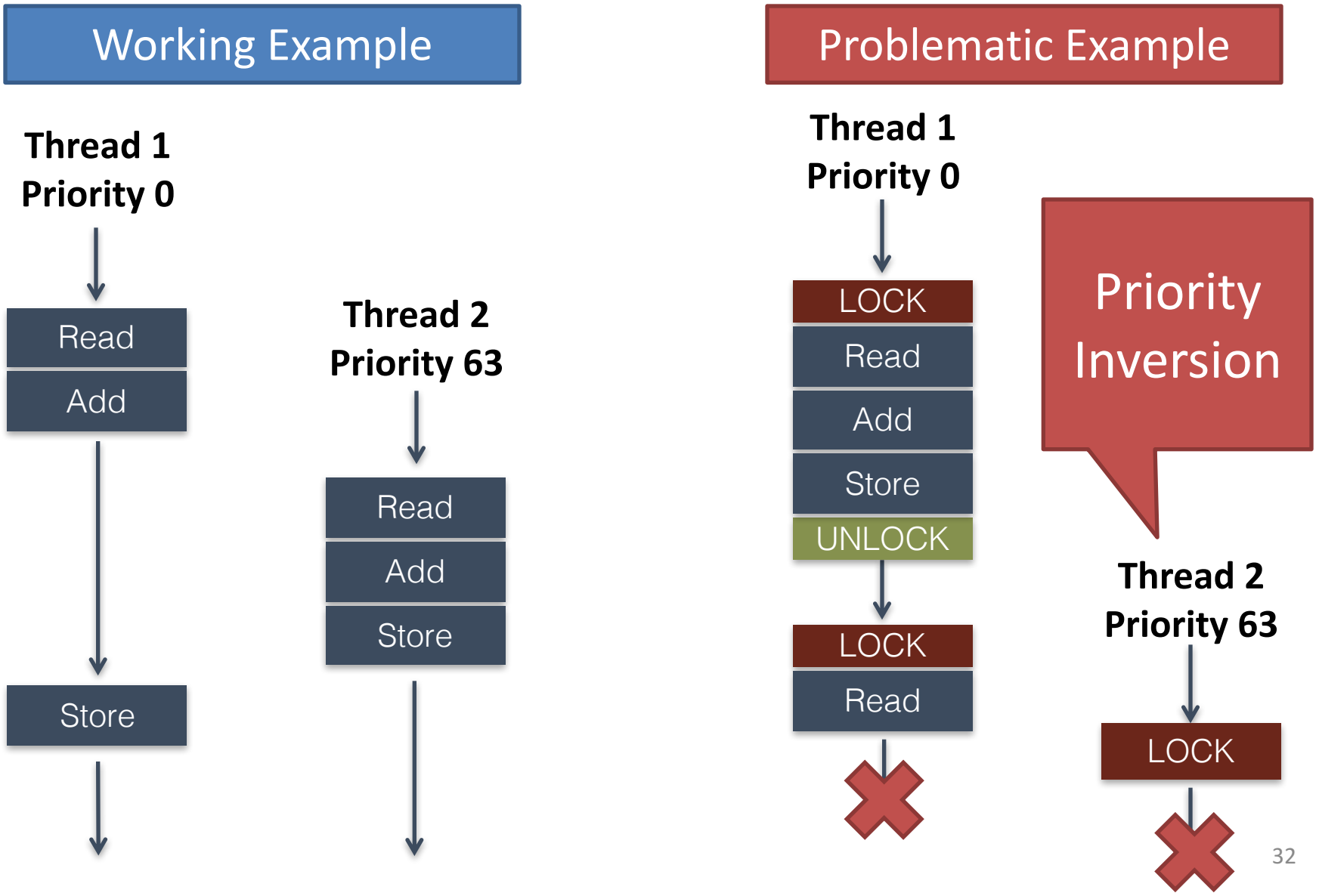
Better Implementation?

- I just (partially) solved part of Project 1 for you
  - You're welcome :)
- But, my implementation still isn't efficient enough
- How could you improve it?
- Build your own improved timer_sleep() implementation and answer 6 questions about it in your DESIGNDOC

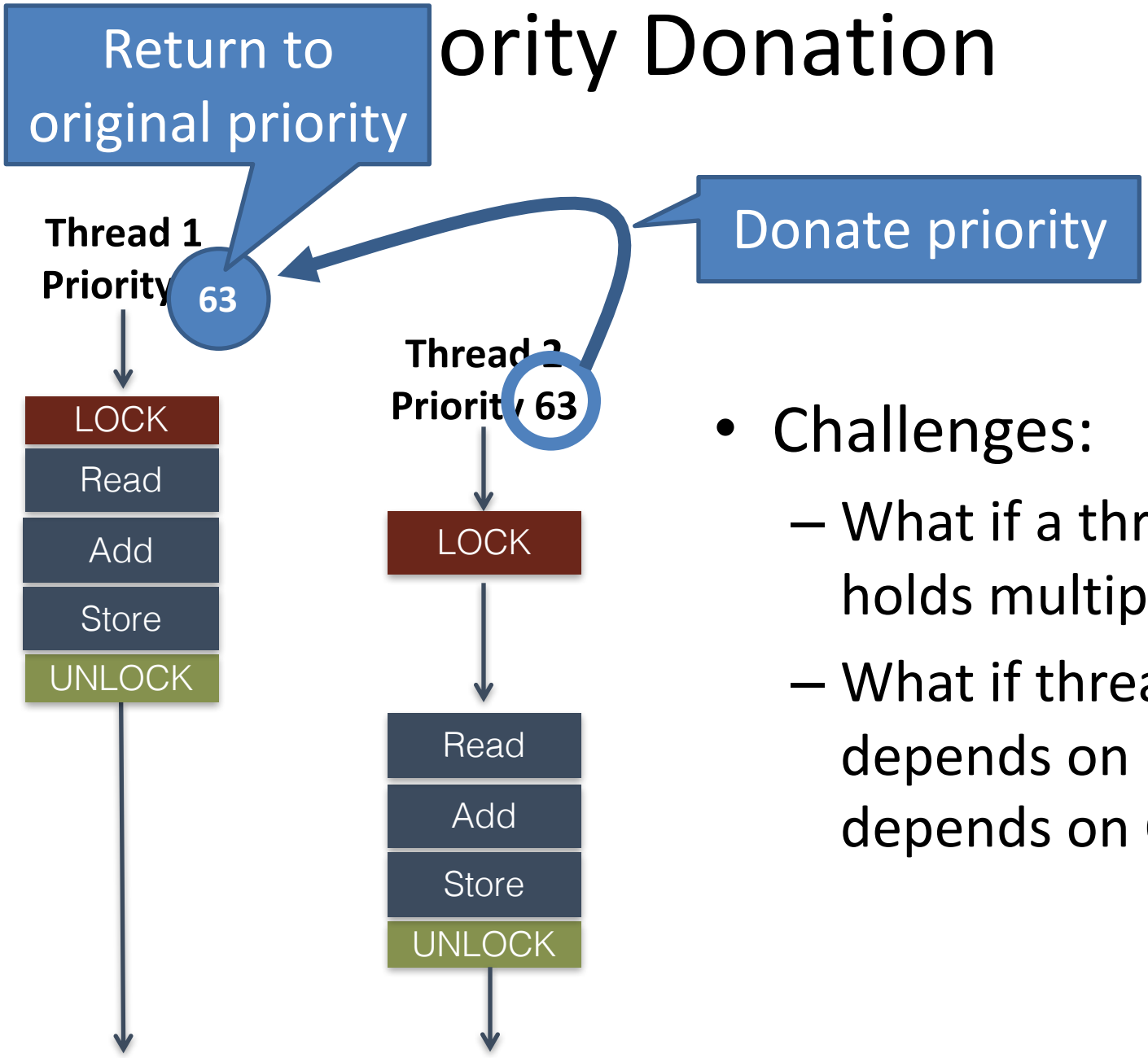
Goal 2: Thread Priority

- Modify the Pintos thread scheduler to support priorities
  - Each thread has a priority
  - High priority threads execute before low priority threads
- Why is this challenging?
  - Priority inversion
- Implement priority scheduling and answer 7 questions about it in your DESIGNDOC

Priority Scheduling Examples



ority Donation



- Challenges:
  - What if a thread holds multiple locks?
  - What if thread A depends on B, and B depends on C?

Overall File Modifications

- What files will you be modifying in project 1?
  - devices/timer.c
  - threads/synch.c ← Most edits will be here…
  - threads/thread.c ← … and here
  - threads/thread.h
  - threads/DESIGNDOC ← Text file that you will write

Advanced Scheduler? MLFQ?

- Project 1 originally included more work
  - Asked student to build an advanced scheduler that implements MLFQ
- We have removed this from the assignment
  - We will study scheduling later in the course
- If you see references in the docs to “advanced scheduler” or references in the code to “mlfq” ignore them
  - Might be a good idea to remove the mlfq tests to save time when running the full test suite.

DUE: October 3
11:59:59PM PST

QUESTIONS?