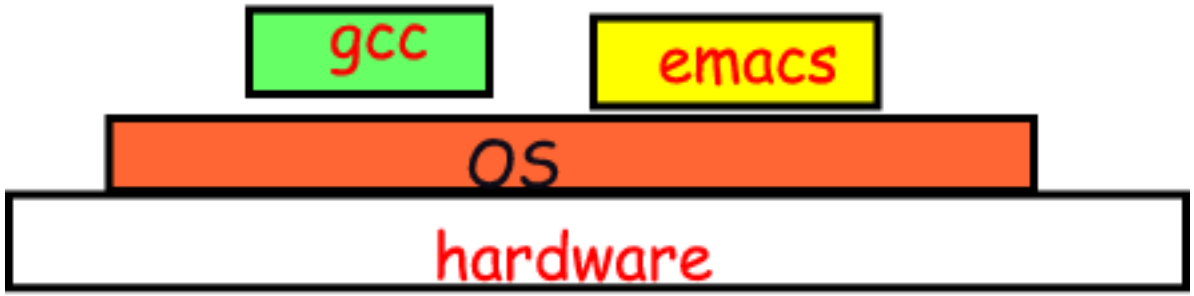
CS 140 Project 1

Threads

Overview

- Background
  - OS
  - Threads
  - Scheduling
  - Synchronization
- Project
  - Pintos
  - Alarm Clock
  - Priority Scheduling
  - Advanced Scheduler

Background: Operating System



- Interface between user programs and system resources
- Provides:
  - Resource Sharing
  - Protection
- User programs gain access to protected resources via system calls

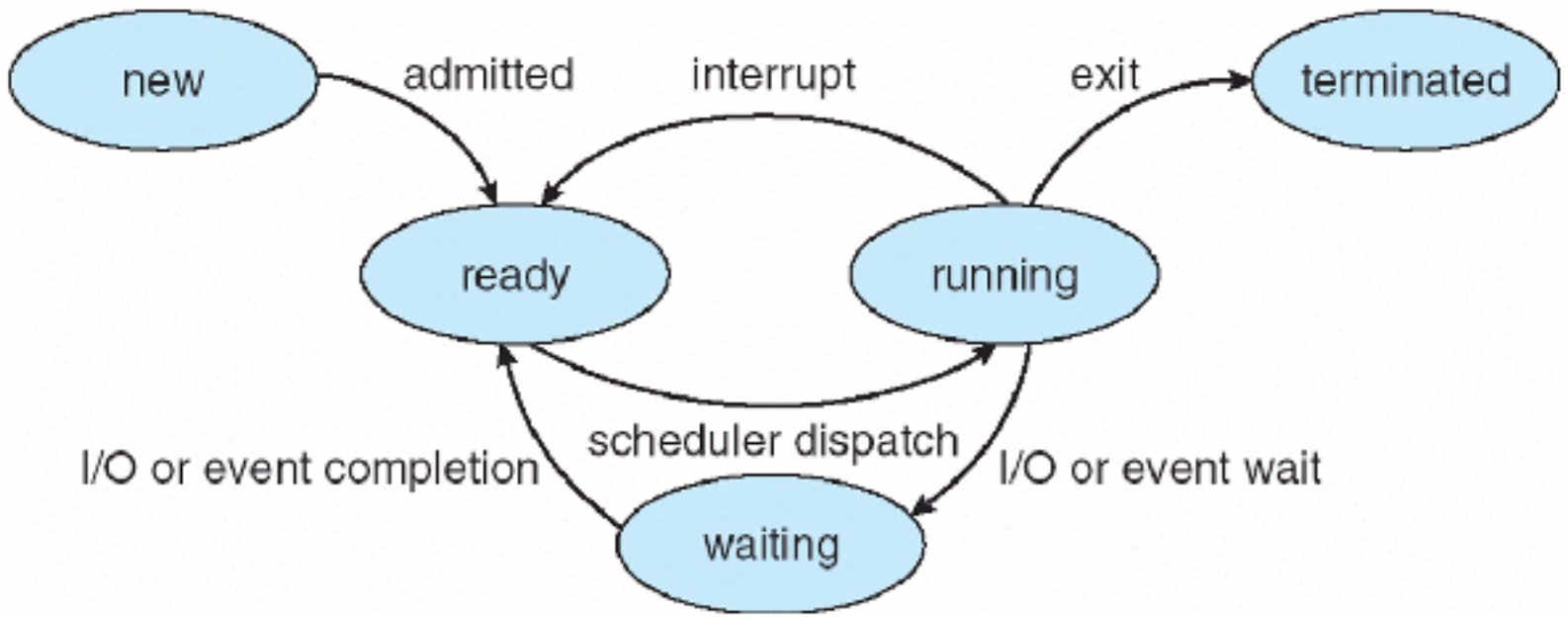
Background: Threads

- Execution Stream
- Execution context consists of:
  - Current location in program
  - Values of variables
- This is maintained on the thread's stack

Pintos:

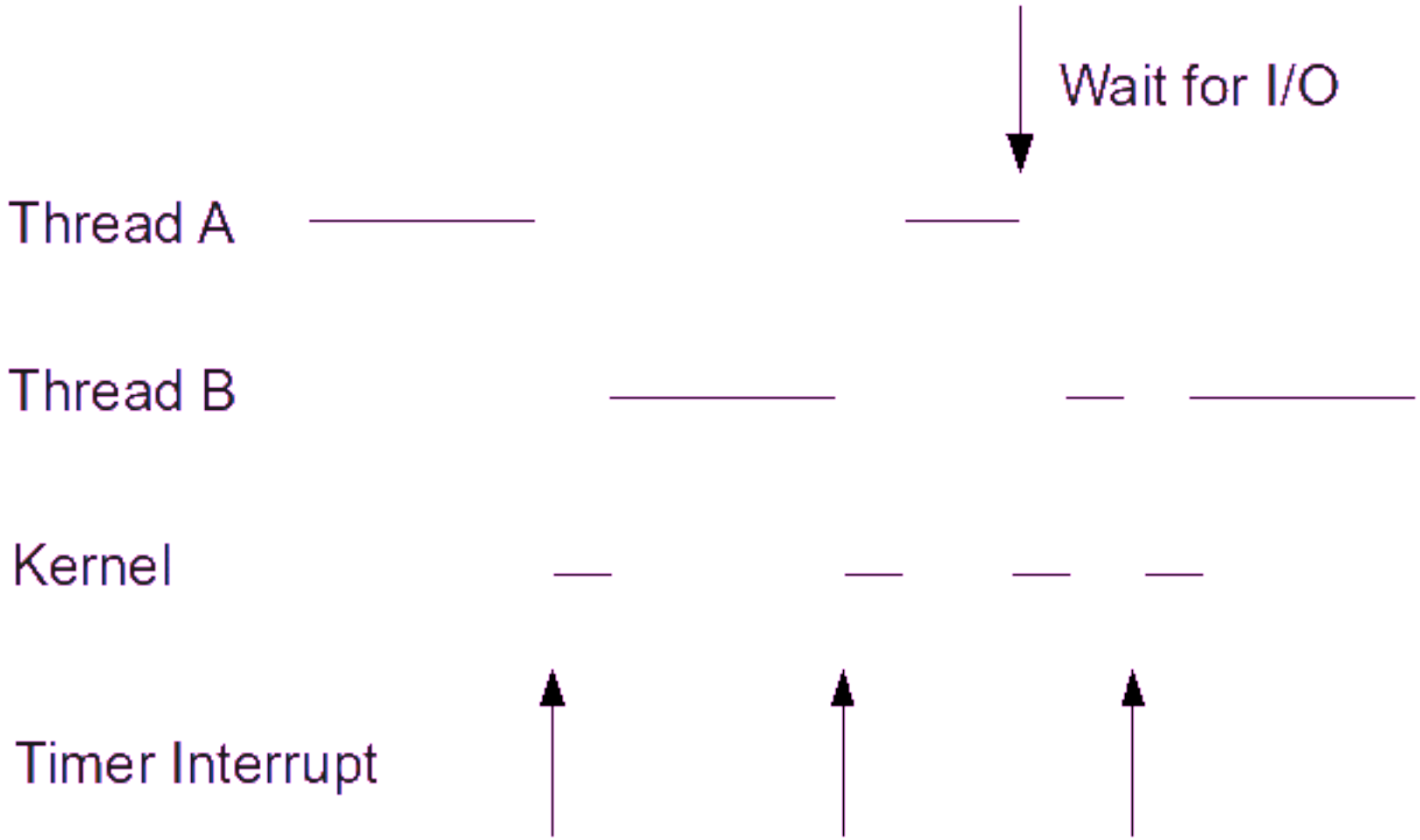
```
struct thread {
    name
    ID
    stack pointer
    page directory
    etc.
}
```

Background: Thread Scheduling



- Scheduling Policy - Chooses a "ready" thread and executes it
  - Round Robin
  - Priority
- Preemption - Remove the "running" thread and schedule a new one
  - Timer Interrupt
  - Device Interrupt

Background: Thread Scheduling



Background: Synchronization

- Covered in depth in lecture next week
- Concurrency issues - prevent race conditions
- Interrupts
  - Can disable interrupts to prevent preemption (cons?)
- Use synchronization primitives instead
  - Locks
  - Semaphores
  - Condition Variables

Project: Pintos

- Simple OS implementation
- Can run on hardware, but we will use x86 simulators to run it
  - Bochs
  - QEMU
- Follow website instructions to install and set path
- To build, run "make" from pintos/src/threads directory
- To run test suite, run "make check"
- "pintos" script provided to simplify use of the simulators
  - "pintos -v -- -q run alarm-single"

Project: Alarm Clock

Function: void **timer_sleep** (int64_t ticks)

- Current implementation busy waits: it loops until enough time has elapsed
- Need to instead put the thread to sleep and then wake it up when enough time has elapsed

Project: Priority Scheduling

- Each thread is assigned one of 64 priorities

- Ensure that the highest-priority "ready" thread is always running
- Several cases:
  - Scheduler chooses the next thread to execute - choose the highest-priority thread
  - Several threads are waiting on a lock/semaphore - wake up the highest-priority thread
  - A thread lowers it priority - should immediately yield if now a higher-priority thread is "ready"
  - A higher-priority thread wakes up - it should preempt the running thread
  - etc.

Project: Priority Donation

- Problem: Priority Inversion
  - L acquires a lock
  - H becomes ready, preempts L and starts running
  - M becomes ready
  - H waits for the lock held by L
  - M starts running
- M runs, then L, then H

- Solution: Priority Donation
  - When H tries to acquire the lock, it donates its priority to L
  - When L releases the lock, it returns priority to H
- Nested Donations: H->M->L
- Multiple Donations: H->L<-M

Project: Advanced Scheduler

- BSD Scheduler
  - Attempts to reduce the average response time of jobs
  - Calculates statistics during thread execution and sets priority based on these
  - Does not do priority donation

Design Document

- Template provided on project website
- Start the document early
- Thinking hard about design before coding will save a lot of time

Submission

- Save design document as DESIGNDOC in threads directory
- Run "make grade"
- Copy build/grade to GRADE in threads directory
- Run "make clean"
- Run "/usr/class/cs140/bin/submit 1"

Final Advice

- Start early
- Read the code - it is very well documented
- Think about design a lot before coding
- Meet as a group often to integrate
- Tools
  - CVS, git, svn
  - GDB