CS140 Project Session 1

THREADS

(Based on slides from previous quarters)

Syed Akbar Mehdi

Overview

- Getting Started
- Overview
- Project 1 Tasks
  - Alarm Clock
  - Priority Scheduling
  - Priority Donation
  - Advanced Scheduler (MLFQS)
- Misc.

Getting Started

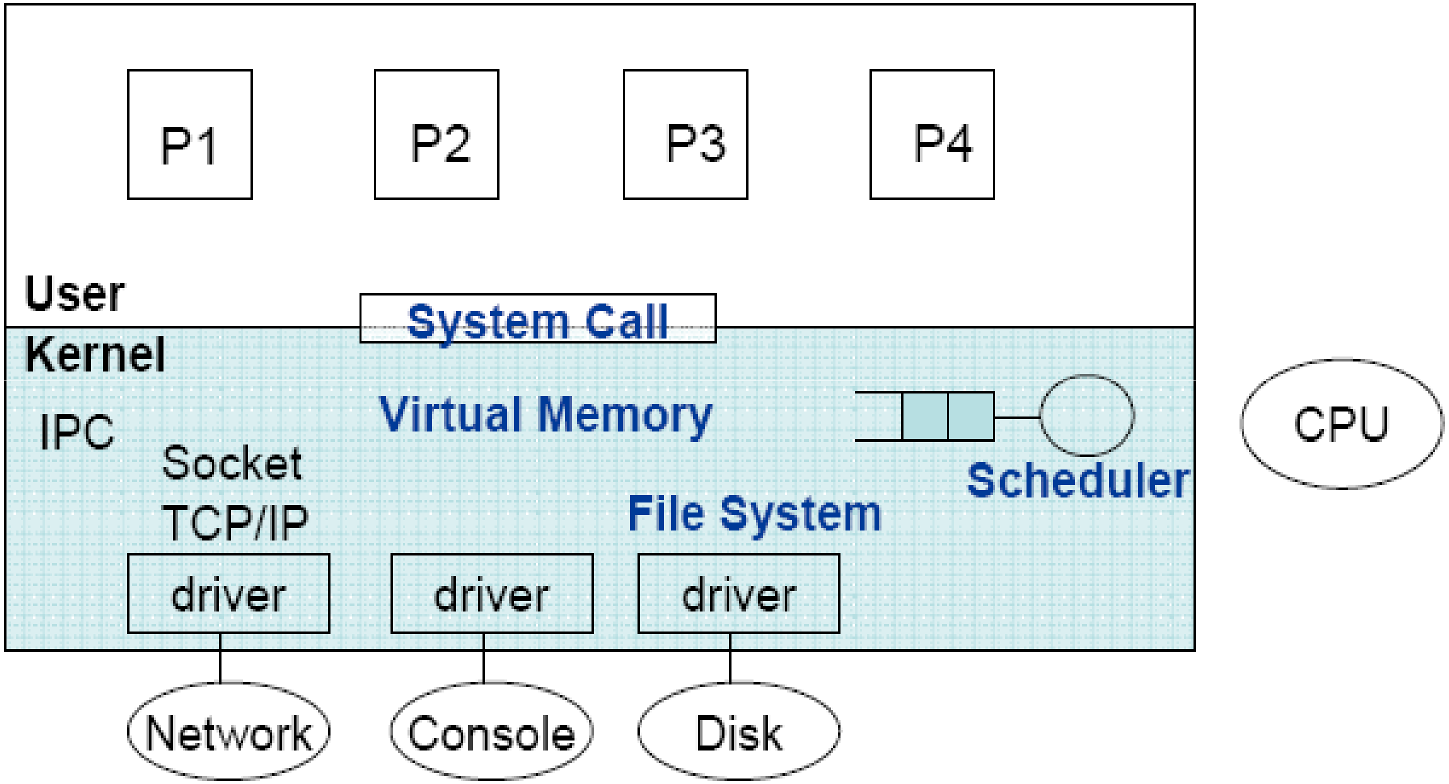
- Stanford UNIX Computing Environments
  - http://www.stanford.edu/services/unixcomputing/environments.html

- We will be testing your projects on the myth machines primarily.

- Make sure Pintos is up and running
  - **set path = (/usr/class/cs140/`uname -m`/bin $path)**
  - **tar xzf /usr/class/cs140/pintos/pintos.tar.gz**
  - **cd pintos/src/threads/**
  - **make**
  - **cd build/**
  - **pintos –v -- run alarm-multiple**

Overview

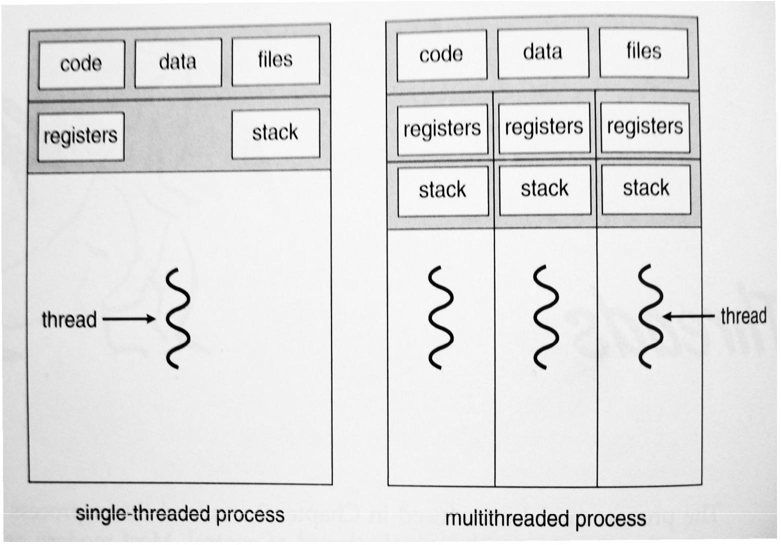
Typical OS structure



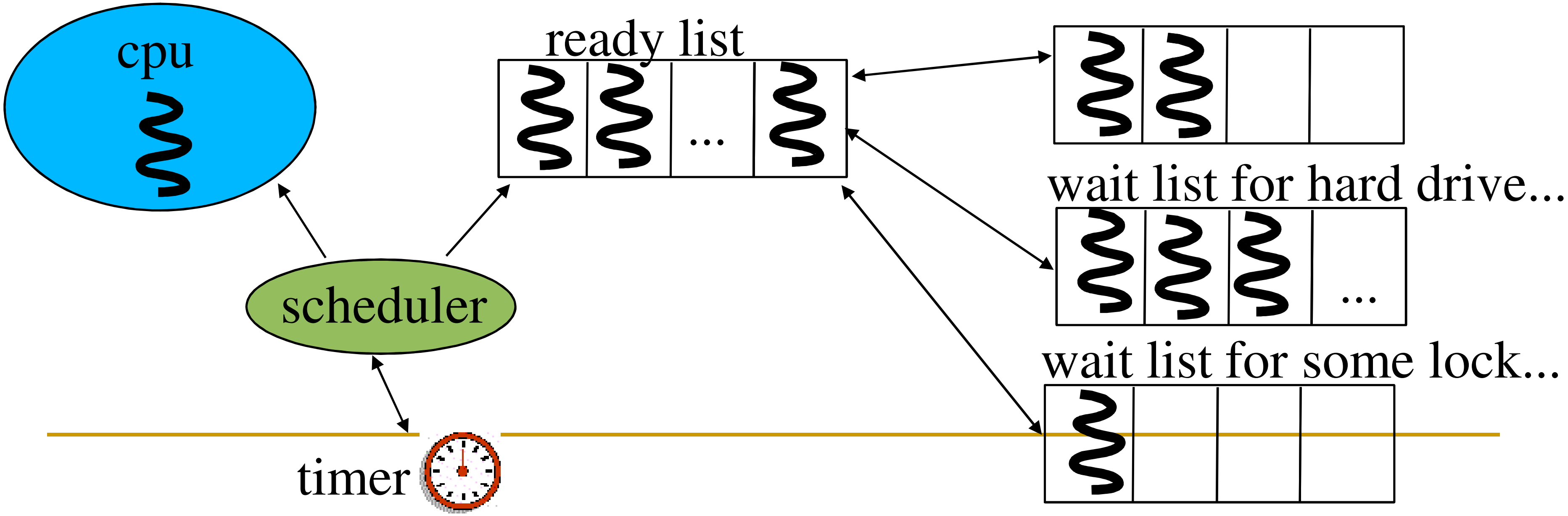
Based on Chia-Hui Tai’s slides Autumn ‘07

Thread System Overview

Based on Ben Sapp's slides Winter '06



```
4 kB +---------------------------------+
     |          kernel stack           |
     |                |                |
     |                |                |
     |                V                |
     |         grows downward          |
     |                                 |
     |                                 |
     |                                 |
     |                                 |
     |                                 |
     |                                 |
     |                                 |
     +---------------------------------+
     |              magic              |
     |                :                |
     |                :                |
     |             status              |
     |              tid                |
0 kB +---------------------------------+
```



Project 1 Tasks

Part I: Alarm Clock

- Already in Pintos: devices/timer.c
- Redo timer_sleep() to avoid busy waiting:

```
/* Suspends execution for approximately TICKS timer ticks. */
void timer_sleep (int64_t ticks){
  int64_t start = timer_ticks ();
  ASSERT (intr_get_level () == INTR_ON);

  while (timer_elapsed (start) < ticks)
    thread_yield ();
}
```

- Requirement:
  - No busy waiting

- Hint: Need to take thread off the ready list

Part II-A: Priority Scheduling

- Implement Priority Scheduling
  - Thread in ready list with the highest priority is always selected to run.
  - If a thread is added to the ready list with a higher priority than the running thread, yield the cpu <u>immediately</u> to the new thread.
  - Threads waiting on a semaphore, lock, or condition variable should have the highest priority waiting thread wake up first.
- Go over source code in thread.c and synch.c.
- Hint: When does the scheduler need to take action?
  - Picking next thread to run?
  - Adding threads to ready list?
- Priority Scheduling needs to work before Part II-B and Part III.

Part II-B: Priority Donation

- *Priority Inversion*: Consider three threads H,L and M with priorities p(H) > p(M) > p(L)
    - H needs a lock held by L.
    - H must wait until L runs and releases the lock.
    - If M is on the ready list H will never get CPU.
    - Higher priority thread is getting starved waiting for lower priority threads.
- Implement Priority Donation.
- H donates its priority to L, which then runs and releases the lock.
- Important:
    - Remember to return L to previous priority once it releases the lock.
    - Be sure to handle multiple donations (max of all donations)
    - Be sure to handle nested donations, e.g., H waits on M which waits on L...
- Need to Implement only for locks

Part III: BSD Scheduler

- Multi-Level Feedback queue scheduler
  - 64 ready queues, one for each priority
  - scheduler chooses a thread from the highest-priority non-empty queue
  - priority calculated using the recent cpu time used by a thread, and it's “niceness”.
  - Appendix B4.4 for details.

- Enabled when thread_mlfqs == true.

- No need to handle priority donation with BSD scheduler.

- Fixed-Point Real Arithmetic.

Misc.

Useful Tools

- cvs/svn for maintaining code revisions.
  - Setup web based cvs/svn repository for all group members to commit and check out code.
- Debugging tools.
  - GDB and useful macros like dumplist, bthreadlist
  - Backtrace
  - Read Debugging Manual at
    - http://www.stanford.edu/class/cs140/projects/pintos/pintos_10.html#SEC142
- Run an individual test (e.g. alarm-multiple)
  - make build/tests/threads/alarm-multiple.result, OR
  - pintos -v -- run alarm-multiple
- Data structure libraries in pintos/src/lib/kernel/
  - Linked lists
  - Hash Tables (Useful from Project 2 onwards)
- Newsgroup (su.class.cs140)

Random Advice

- Read the project description and Pintos Reference Manual before starting.
- Go through current code and understand the basic structure.
- Spend lots of time on the design before starting to code.
- Integrate early.
- Synchronization
  - Keep in mind: A thread can be interrupted by another thread

Grading

- 50% automatic tests
  - no exceptions
- 50% design document
  - data structures, algorithms, synchronization, and rationale.
  - coding standards: don't forget to indent and comment!

Questions?