Today in OS

Concurrency (locks) => Atomicity
(OS support: new) + hardware support

Ordering (when code runs)

what makes a good lock?

- correctness
- overhead
- fairness
  (extreme: starvation) => never get lock

Problems: focus on overhead (waste)

T1:调度 acquire spin lock (xchg-based)
    \[ (xchg()) == 1 \]
    while spin
    yield; // spin

OS support:

- yield() system call

same as ready

when thread calls yield => moves calling thread from running

to runnable (and push to back)
CPU: \( T_1, T_2, \ldots, T_{100} \) of scheduling queue.

**Critical Section**

- Timer Interrupt
  - Run Time Slice
    - \( T_2 \) (spins) [10ms]
    - \( T_3 \) (spins) [10ms]
    - \( T_4 \)

**Worst Case Analysis**:
- Num of threads \( \times \) length of time slice [waste]

**Worst Case**
- (yield) << (spin)
- Much faster

**Admin**

\[ C + \text{Concurrency} = \text{SEGFAULT} \]
→ midterm scores released
→ solutions coming ASAP
  (nice version)
→ P2b : due Monday
  (grace period : 2 days)
→ discussion: office hour (not recorded)

Atomicity

Locks: e.g.

h/w support: exchange inst.

OS support: e.g. yield, park/cmpark

Problem #2: Ordering + control races
  (condition variables)