Today in OS

Concurrency (locks)

Last time:

what makes a good lock?

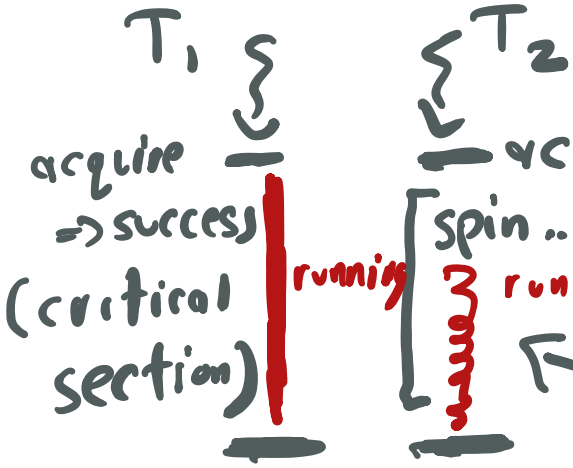
- correctness
- overhead
- fairness

(extreme: starvation) → never get lock

Problems: focus on overhead (waste)

data → => Atomicity ((OS) support: new + hardware support)

control (when code runs) → => Ordering (new primitive: condition variable)



spin lock (xchg-based)

```
while (xchg() == 1)
    yield(); // spin
```

how to reduce waste of spinning

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)

T_1 acquire crit. section

timer interrupt { run time slice

T_2 (spins) } 10 ms

T_3 spins } 10 ms

T_4 ...

T_{100}] 10 ms

worst case analysis:

[num of threads x length of time slice] [waste]

now, with yield:

T_2 yield → T_3 yield → T_4 ... → T_{100}

worst case:

[num of threads x context switch cost]

worst case (yield) << worst case (spin)

[much faster]

Admin

[C + Concurrency = SEG FAULT

→ midterm scores released
→ solutions coming ASAP
(nice version)
→ p2b : due Monday
(grace period : 2 days)
→ discussion : office hour (not recorded)

Atomicity

Locks :

- h/w support : e.g. exchange inst.
- OS support : e.g. yield, park/unpark

Problem #2 :

Ordering + control races

[condition variables]