CS 219

Lecture 26

18th Mar. Wed
2pm
Lab Quiz 3
xv6

25th Quiz 2
CS219 8:30am
wed.

24th/25th

~ OS from
the ground up.

concurrency & synchronization

race condition

atomicity

mutual exclusion

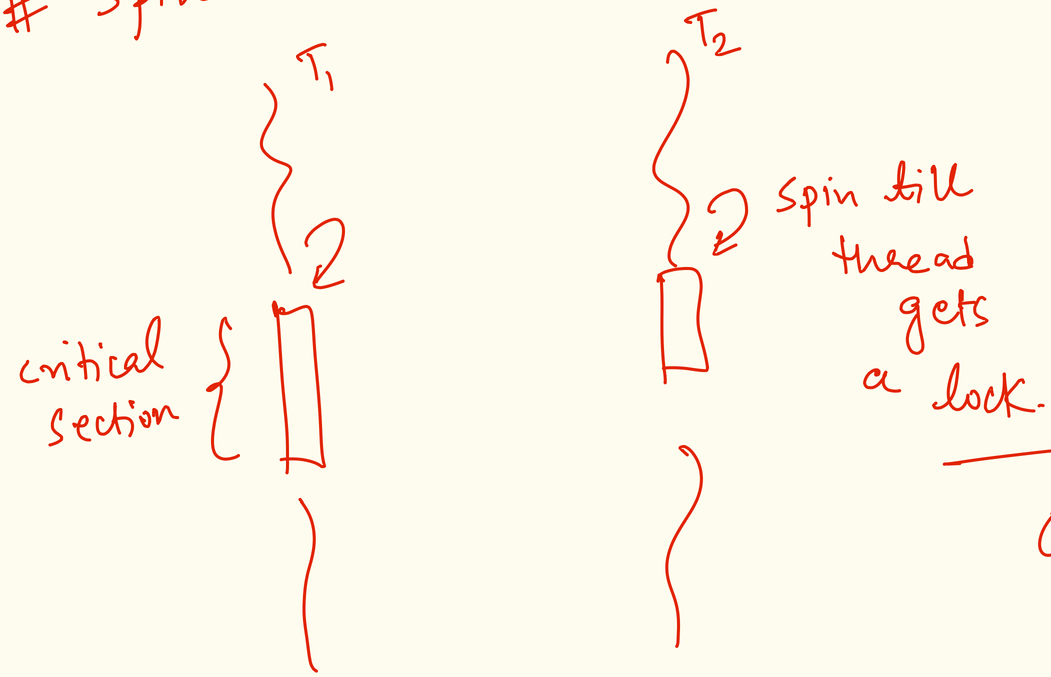
critical section

disable preemption

disable interrupts

locks

1 # spin lock



memory variable/object

0	available
1	locked ~~taken~~

```
getlock(L) {
    while (L==1);
    L=1;
    return;
}
```

read
cmp
update

multiple & non-atomic ⇒ non-deterministic!

```
unlock(L) {
    L=0;
}
```

1. multiple C statements
2. single C statement
3. single ISA instr

all non-atomic!

④ OS + ISA handshake

eg: ISA's provide a family/set of atomic operations.

lock ; inc eax;

no other inst^{r}. can access / fetch memory till this instruction is over.

TSL ~ Test and Set Lock

```
mov R0, 1
TSL R0, LOCK
```

LOCK → value at memory a address

ATOMIC swap!

```
{ temp = LOCK;
  LOCK = R0;
  R0 = temp;   //oldvalue
```

x86

xchg

```
mov eax, 1
xchg eax, LOCK
```

xchg → atomic swap

cmpxchg (eax, LOCK, 1) ← value for update

```
if(eax == LOCK)
    ZF ← 1;
    LOCK ← 1;
else
    ZF ← 0
    LOCK ← eax;
```

spinlock w/ atomic ISA instructions

LOCK $< {0 \atop 1}$

```
getlock:  MOV R0, 1
          TSL R0, LOCK   < 0
                           1
          CMP R0, 1
          jz getlock
          .
          .
          .
```

spin: jmp back to getlock since $R_0 = 1$ ⟹ LOCK was taken already

unlock:

```
TSL LOCK, 0
```

+ves: - is a ~~valid~~ / correct sync. primitive

- great for small critical sections.

-ves: ① ~ # contending threads & critical section work increases

spinlock overheads increase!

② cannot address generalized condition.

② mutex / sleeplock

- if lock not available,
  wait / sleep
  do not burn CPU.
- recheck LOCK when lock available
- (*) on unlock wakeup all threads waiting on lock.

mutexlock :

```
mutexlock :   mov R0, 1
              TSL R0, LOCK
              CMP R0, 0
              jz  OK
              call yield
              jmp mutexlock
              :
OK:                          // got lock
```

call yield ← give up cpu

- proc -> state = WAITING / BLOCKED
- proc -> chan = LOCK;

mutex unlock :

```
TSL LOCK, 0
wakeup (LOCK)
```

wakeup (LOCK) → wakes up all process waiting for LOCK.