CS 744 Lecture 15

Midsem
scheduled for 27th Sunday 2pm.

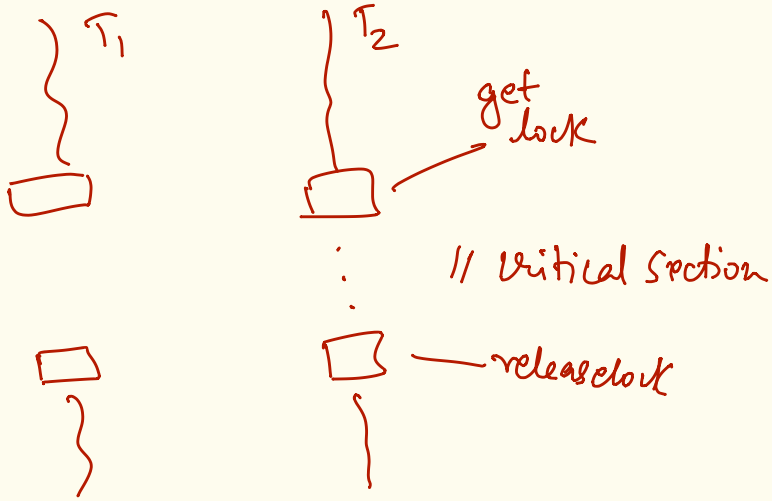
⊛ Synchronization & concurrency!

- multiple processes
- multiple threads
- multiple OS instances

↑ simultaneous / concurrent threads
↑ shared memory.

↑ threads of execution

solutions:
1. disable preemption
2. disable interrupts
3. locks



Q how to design the lock?

- what is a lock?
  - memory variable / object / value at an address!
    - lock ← 0 — available
    - ← 1 — unavailable

(#) **Spinlock** : ~ spin on CPU till LOCK is obtained.

LOCK : ←

```
getlock:  MOV R0, 1
          CMP LOCK, 0
          JNZ getlock
          MOV LOCK, R0
          :
          :
```

Test and Set

Compare & Swap / Exchange

↓

w/o atomicity is a problem!

OS relies on the ISA to provide atomic instructions for Test & Set

xchg instructions

all or nothing!

e.g: TSL ——— test & set,

```
TSL LOCK, R0
```

atomic sequence:
- oldvalue = LOCK
- LOCK = R0
- return oldvalue in R0.

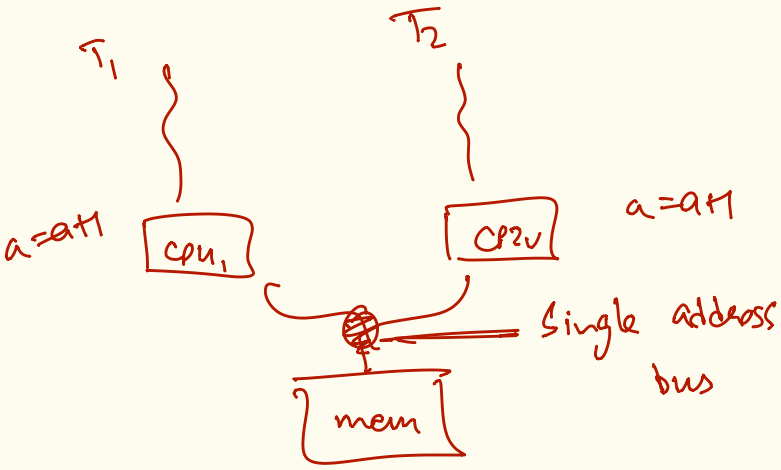
```
getlock:  MOV R0, 1
          TSL LOCK, R0
          CMP R0, 1
          J2 getlock
          :
          :
```

```
lock; inc eax
```

LISA tag locks the address bus

release lock:

```
MOV LOCK, 0
```



T1

```
getlock(L)
a=a+1;
releaselock(L);
```

T2

```
getlock(L);
a=a+1;
releaselock(L);
```

inc eax; ← single instruction increments value of eax;

fetch - decode - execute ||

fetch - decode - execute

fetch - decode - execute

lock; inc eax

↳ freeze address bus till instruction finishes.

---

(2) mutex / sleeplock

↳ if LOCK not available

wait / sleep → no spinning!

on available, try to get the lock

```
mutexlock: mov R0, 1
           TSL LOCK, R0
           CMP R0, 0
           JZ  OK      ← got lock
           CALL YIELD  → proc->state = WAITING;
                         slow out of CPU till lock not available
           jmp mutexlock
```

mutex unlock :

```
mov LOCK, 0
wakeup (LOCK);
```

```
struct mutex {  int id;
                spinlock s;
                int lock;
                }
```

0 — available

1 — unavailable & sleep

```
mutex lock ( mutex * m) {
    spinlock (m -> s);
    while ( m -> lock == 1)
        sleep(m->id; m->s);

    m -> lock = 1;

    releasespinlock (m->s);
```

read lock
cmp lock, 1
jz

```
sleep ( id,
{
    proc -> state = WAITING;
    proc -> chan = id;
        condition
    releasespinlock (m->s);
    scheduler();
    spinlock (m->s);
}
```

lock(proc);
unlock(m->s);

unlock(p)
lock(m->s);

```
mutex unlock( mutex *m)
{
    spinlock (m -> s);
        m -> lock = 0;
        wakeup (m->id);

releasespinunlock (m->s);
}
```

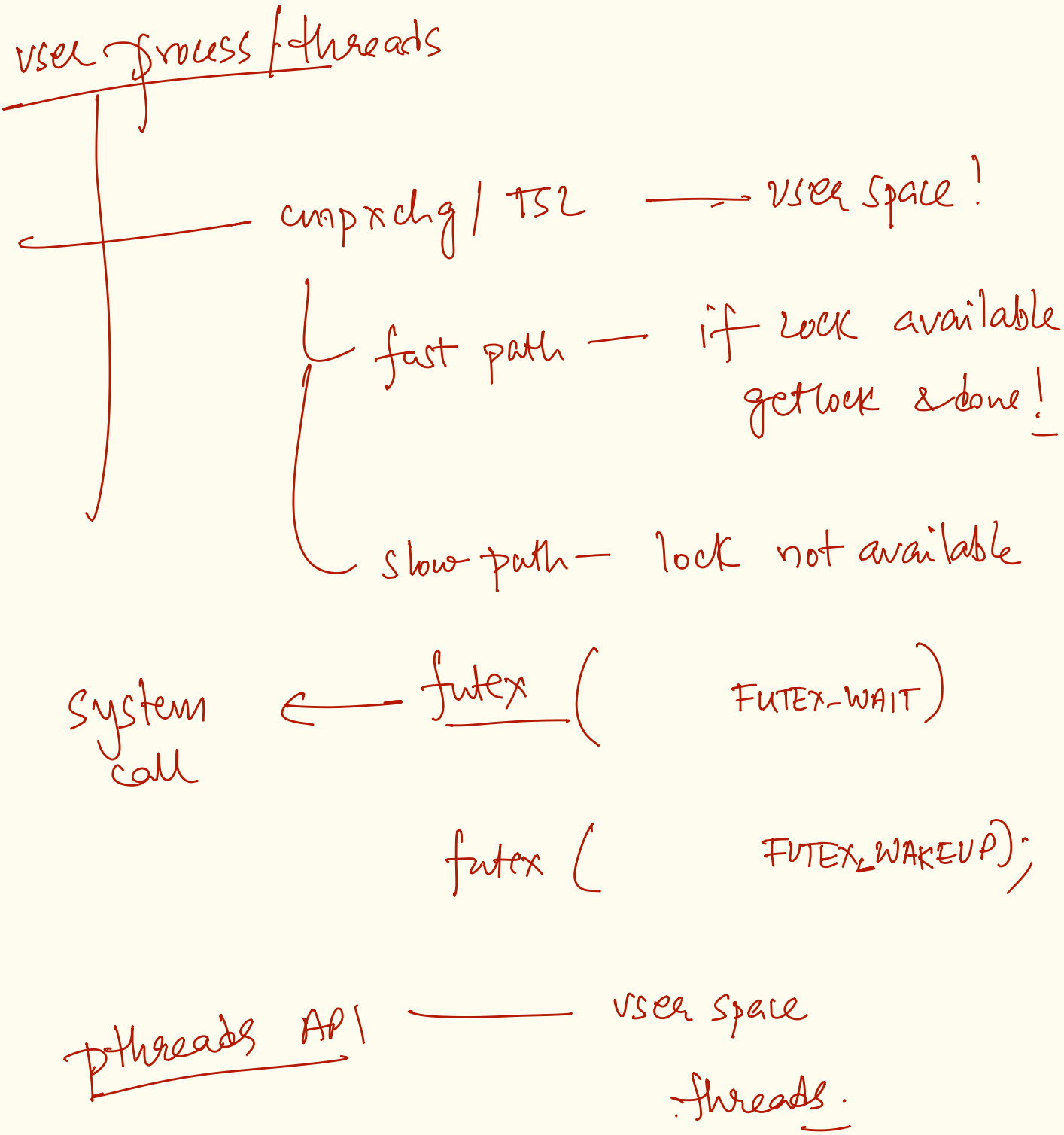
```
wakeup ( id) {
    for all processes
        if p -> chan == id
            p->state = READY;
}
```

```
mutex m;
m->id = 23;
```

T_1 T_2 T_3

```
{
    mutex lock (m);

    mutex unlock(m);
```


user process / threads
cmpxchg / TSL
user space !
fast path — if lock available
getlock & done !
slow path — lock not available
system call
futex (FUTEX_WAIT)
futex (FUTEX_WAKEUP);
pthreads API
user space
threads.