Lecture 19

synchronization & threads primitives

Key words:

race condition, critical section, atomic operations, mutual exclusion,

instructions: xchg, test-and-set, cmpswp,

locks, spinlock, mutex, synchronization

↳ sleep lock, sleep/wakeup, wait/signal

Lab quiz 3 - week of 16th Oct

Quiz 2 - week of 23rd Oct

Lab quiz 4 - week of 6th Nov

Endsem - ??

---

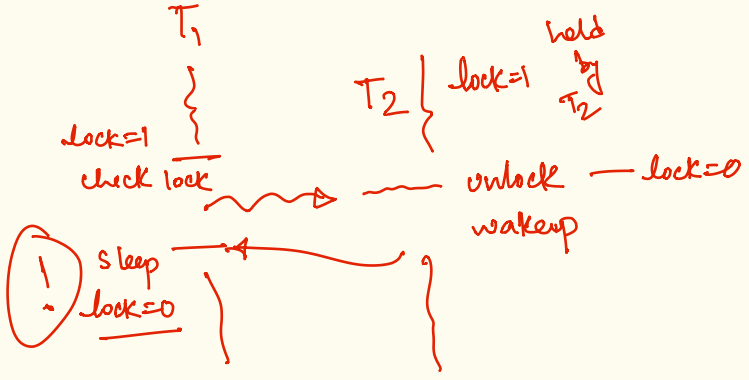
mutex : non-blocking lock/sync. primitive.

lock is a variable

memory object

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

mutex_lock (mutex *m) {

    while (xchg(lock, 1) != 0)   ⊛

        sleep (m->id);   ⊛

    m->lock = 1;
}
```

```
mutex_unlock (mutex *m)
{
    m->lock = 0;
    wakeup (m->id);
}
```



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

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

```
* wakeup (m->id) {
    for all processes p {
        if p->chan == m->id {
            p->state = RUNNABLE;
        }
        p->chan = 0;
    }
}
```

```
* sleep (m->id, m->s) {
    p->state = SLEEPING;
    p->chan = m->id;
    // set process p to sleep;
    // records condition for sleep;
    spinunlock(m->s);
    sched(); // invoke scheduler
    spinlock(m->s);
    return;
}
```

```
spinlock l1;
spinlock l2;
```

Bad condition

```
lock(l1);
count++;
unlock(l1);
```

```
lock(l2);
count++;
unlock(l2);
```

```
mutex_unlock (m) {
    spinlock(m->s);
    m->lock = 0;
    wakeup(m->id);
    spinunlock(m->s);
}
```

Synchronization in the Kernel

- locks
+
- disable local interrupts

on the same cpu

- disables local interrupt
- xchg

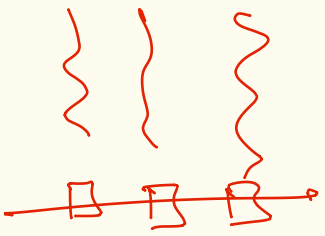
```
fork() {
    spinlock(ptable.lock);
    p = newproc();
    spinunlock(ptable.lock);
}
```

Semaphore

```
semaphore s(5);
```

down() — if (s > 0) s--; else sleep

up() — s++; wakeup(s);



barrier

semaphore s1 (5);

semaphore s2 (0);