Lecture 28

27 September 2018 10:51

Let's get started!

Purru has a frog in his throat!

Unais to handle proceedings.

Mutex / Non-spinning lock.

```
mutex_lock :
        mov   R0, 1
        Test-and-set  lock, R0
        JZ    Done
        CALL  yield
        JMP   mutex_lock
Done :  RET
```

Problem : Inefficient : Scheduler selects the same process again and again.

Design: 1 get the process to blocked state and not schedule until lock is available

- wait
- signal

```
struct  mutex
{
        int   locked
        int   id
}       spinlock  s
```

```
mutex_lock (mutex *m)
{
        if (m->locked)
                wait (m->id)
        m->locked = 1
}
```

```
mutex_unlock (mutex *m)
{
        m->locked = 0
        signal (m->id)
}
```

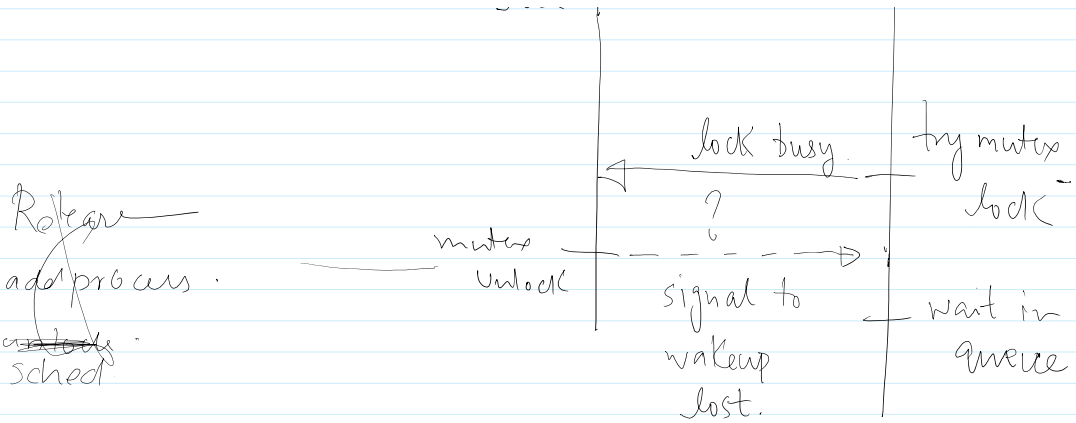
```
mutex_lock (m)
{
        spin_lock(m->s)
while   ( m->locked )
        {
        release (m->s)

        wait (m->id)
        sched()
        spin_lock(m->s)

        }
        m->locked = 1
        release (m->s)
}
```

```
mutex_unlock (m)
{
        spin_lock(m->s)
        m->locked = 0
        signal (m->id)
        release (m->s)
}
```

P1

P2

mutex lock



await (m->id, m->s)
{
add process (m->id)
unlock(m->s)
sched()
spinlock(m->s)
}

this is a tricky f^n call. you will wakeup in another process!

Condition variables based sync. (mutex is one example & using CVs for a non-spi lock

```
cond_lock(m)
{
    spinlock(m->s)
    while(m->l)
        wait(m->id, m->s);
    m->l = 1;
    spinunlock(m->s);
}
```

Condition

variable on which processes blocked

```
cond_unlock(m)
    spinlock(m->s);
    m->l = 0;
    wakeup(m->id);
    spinunlock(m->s);
```

wakeup processes sleeping on this variable.

"Condition" can be generic

① m->l : 0/1 is a binary (lock-like) condition.

② have all processes executed a f^n 10 times

nning

k).

```
struct mutex
{
    int l;
    int id;
    spinlock s;
} m;
```

(3) wait till all processes reach an execution pt.

etc. etc.