Lecture 30

04 October 2018 11:37

0 Quiz #3. 5th Oct. 2018. 8.30 am CC 101, 103, 103 | Tommorow! | Chapters 28, 29, 30, 31 26

1. Synchronization so far.

(a) Spin locks — Kernel-execution entities | disable interrupts — atomicity.

(b) non-spinning locks — sync. w/ "schedulable" entities | process context
- process context + yield

(c) Condition variables
- process sync. on conditions < sleep + wakeup / spin lock (simplified)

(d) mutex < 0/1 condition

(e) Semaphore

Spinlocks / mutexes ⇒

```
lock;
 |
c.s
 |
unlock;
```

Sync. — locking / notification — signaling

(2) # Reader-Writers problem.

- many readers. no sync.
- many writers. sync.
- active readers + writers. sync.
- active writers + readers. sync.

```
ReaderLock()                    ← spinlock
    lock(s);
    while (nwriters > 0)
        sleep(readerv, s);
```

```
WriterLock()
    lock(s);
    while (nwriters > 0 or
           nreaders > 0)
```

while (nwriters > 0)
sleep (readerv, S);
nreaders ++;
unlock(S);

while (nwriters > 0 || nreaders > 0)
sleep (writerv, S);
nwriters ++;
unlock(S);

~~Red~~ Reader Unlock()
lock(S)
nreaders--;
if (nreaders == 0)
wakeup (&writerv);
unlock(S);

Writer Unlock()
lock(S);
nwriters--;
wakeup (&readerv);
wakeup (writerv);
unlock(S);

(3) Semaphores ~ positive integer-valued sync. primitive for signaling coordination

- init ~ sets semaphore to a value >= 0
- down ~ decrements if value < 0 sleep
- up ~ increments, wakeup

~~Red~~ down(S)

up(S)

write up & down w/ condition variables.

Home work

④ semaphore as a lock

semaphore S(1); // init to 1.

down(S) — lock
}

```
down(s)      — lock
{
c.s
}
up(s);       — unlock
```

(30) producer/consumer w/ semaphores. — bounded buffer problem.

```
semaphore ps (max);      MAX.
semaphore cs (0);        currbufsize=0;
```

producer

```
down(ps);
lock;
currbufsize++;
unlock;
up(cs);
```

consumer

```
down(cs);
lock;
currbuffsize--;
unlock;
up(ps);
```