

Lecture 30

04 October 2018 11:37

Quiz #3. 5th Oct. 2018.
8.30 am CC 101, 103, 105

Tomorrow!

Chapters
28, 29,
30, 31
26

1. Synchronization so far.

(a) spin locks ——— kernel-execution entities | disable interrupts
atomicity.

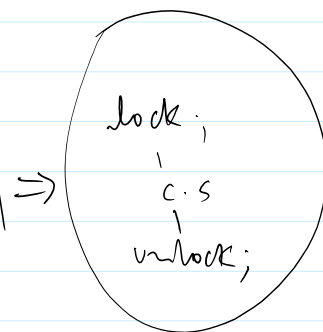
(b) non-spinning locks — sync. w/ "schedulable" entities | process context
process context + yield

(c) condition variables — process sync. on conditions
sleep + wakeup
spin lock (simplified)

(d) mutex < 0/1 condition

(e) Semaphore

spinlocks
mutexes



locking
notification
signaling

② # Readers-Writers problem.

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

Reader Lock(s) ← spinlock
lock(s);
while (nwriters > 0)
sleep(readers, s);

Writer Lock(s)
lock(s);
while (nwriters > 0 or
nreaders > 0)
...

```

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

```

~~Reader~~ Reader unlock()

```

lock(S);
nreaders--;
if (nreaders == 0)
    wakeup(writerv);
unlock(S);

```

```

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

~~down~~ 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

```

② producer/consumer w/ semaphores. — bounded buffer problem.

```

semaphore ps(max);
semaphore cs(0);

```

MAX.
 curbuffsize = 0;

producer

```

down(ps);
lock;
curbuffsize++;
unlock;
up(cs);

```

consumer

```

down(cs);
lock;
curbuffsize--;
unlock;
up(ps);

```