Lecture 20 — synchronization beyond locks

mutual exclusion

Imp. dates
Lab quiz 3: 19th (Thursday) Oct.
Quiz 2: 27th (Friday) Oct.

~ mutual exclusion / serializability.

(i)

```
lock();
{   } critical section.
unlock();
```

spinlock
mutex

~ one critical section, needs to mutally exclusive

~ entity that holds lock, also releases the lock.

(ii) condition variables

- generalized mutex primitives
- condition
  - non-binary
  - could depend on several state variables
  - values could be runtime.
    - ~ 10% of NCHILD processes
- eg: wait for child process state to change.
  - ZOMBIE
  - SLEEPING
- producer - consumer problem.

(iii) semaphore

semaphore S;

integer object

S.init — initializes value of sem

S.down — decrements in S != 0

S.up — increments the value of S.

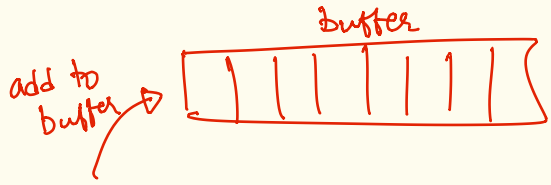
(iv) reader writer lock

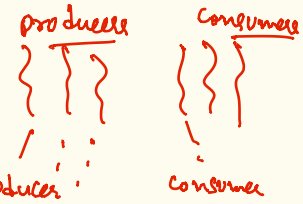
(*) implement a semaphore using a condition variable

```
down(S) {
spinlock(S->L);
    while(S<=0)
        sleep(S, S->L);
    S--;
Spinunlock(S->L);
}
```

```
up(S) {
spinlock(S->L);
    S++;
    wakeup(S);
spinunlock(S->L);
}
```

(3) producer & consumer





- many producers & many consumers.
- bound: buffer has MAX capacity

→ semaphore sp(MAX) sc(0)

producer()

```
down(sp);
lock()
    add to buffer;
unlock()
up(sc);
```

consumer()

```
down(sc);
lock()
    remove from buffer;
unlock()
up(sp);
```

→ condition variables. np → # elements currently in buffer. MAX ~ bound

producer() {

```
spin_lock(L);
    while(np >= MAX)
        sleep(bp, L);
    add to buffer;
    np++;
    wakeup(bc);
spinunlock(L);
```

consumer() {

```
spin_lock(L);
    while(np <= 0)
        sleep(bc, L);
    remove from buffer;
    np--;
    wakeup(bp);
spinunlock(L);
```

(4) reader-writes problem.

- readers & writers.

Synch. primitives ~

i) many readers. — okay.
ii) many writers — not okay.
iii) if reader, no writer.
iv) if writer, no readers or other writers.

Reader writer lock rw;

rw_readlock, rw_readunlock, rw_writelock, rw_writeunlock

using condition variables.

nwriters & nreaders.

writers readers

writelock() readlock()

writeunlock() readunlock()

```
rw_readlock() {
    spinlock(L);
    while (nwriters > 0)
        sleep(rc, L)
    nreaders++;
    spinunlock();
}
rw_readunlock() {

    spinlock(L);

    nreaders--;
    if (nreaders == 0)
        wakeup(wc);
    spinunlock(L)
}
```

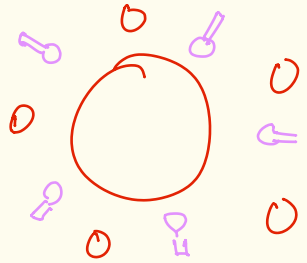
```
rw_writelock() {
    spinlock(L);
    while (nreaders > 0 OR
           nwriters > 0)
        sleep(wc, L);
    nwriters++; spinunlock(L)
}
rw_writeunlock() {

    spinLock(L);

    nwriters--;
    wakeup(wc);
    wakeup(rc);
    spinunlock(L);
}
```

Deadlock : condition for progress never possible!

T1 { ✓ lock(L1) ✗ lock(L2) }

T2 { lock(L2) ✓ lock(L1); ✗ }

Dining Philosophers Problem



```
while (1) {
    think()
    grabspoons()
    eat()
}
```

two spoons