10/21

Agenda
- CV Review
- Bounded Buffer Problem.

```
acquire()
while(!condition){
  wait();
}
release();
```

```
acquire();
signal();
release();
```

Condition Variables.
- put threads to sleep until a condition might be true
- APIs: ① wait : put thread to wating/sleep, release lock
  ② signal : wakes up one thread from watting list.
  ③ broadcast: wake up everyone waiting on that condition.

in xlc:
sleep()
N/A
Wakeup()

Rules for using locks & CVs

① Consistent structure

② Use CVs & locks.
→ don't busy wait or sleep() [syscall sleep]

less wasteful than busy wait

③ Acquire lock at beginning & release at the end.

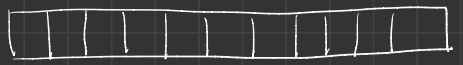
④ Hold lock while operating on CVs.

⑤ Always wait in a while loop
→ MESA vs. Hoare semantics

Implementation Considerations

When waiting upon a `Condition`, a "*spurious wakeup*" is permitted to occur, in general, as a concession to the underlying platform semantics. This has little practical impact on most application programs as a `Condition` should always be waited upon in a loop, testing the state predicate that is being waited for. An implementation is free to remove the possibility of spurious wakeups but it is recommended that applications programmers always assume that they can occur and so always wait in a loop.

Bounded Buffer Problem



fixed size buffer

Producer: produce item and put into an empty slot, blocks if no room to put item (buffer is full)

Consumer: consume item from a slot, blocks if no item to consume

Starter code

```
char buffer[100];
int read_ofs = 0; // consumer reads here
int write_ofs = 0; // producer writes here
int count = 0; // # of items in the buffer.
```

function produce() { ? } function consume() { ? }

state variables

```
char buffer[100];
int read_ofs = 0; // consumer reads here
int write_ofs = 0; // producer writes here
int count = 0; // # of items in the buffer.
```

synchronization

```
buffer_lock;
not_full_cv;
not_empty_cv;
```

```
function produce() {
    buffer_lock.acquire();
    while (count == buffer.size) {
        not_full_cv.wait();
    }
    // there is room to write now
    buffer[write_ofs] = data;
    write_ofs = (write_ofs++) % buffer.size;
    count++;
    not_empty_cv.signal();
    buffer_lock.release();
}
```

```
function consume() {
    buffer_lock.acquire();
    while (count == 0) {
        not_empty_cv.wait();
    }
    // there is data to read now
    data = buffer[read_ofs];
    read_ofs = (read_ofs++) % buffer.size;
    count--;
    not_full_cv.signal();
    buffer_lock.release();
}
```