4/15/24

More Monitors!

The Morning Coffee Problem

```
Lock lk;    int coffee = 0;    Condvar coffee_cv;
```

```
get_coffee() {
  lk.acquire();
  while (coffee <= 0) {
    coffee_cv.wait(lk);
  }
  coffee--;
  lk.release();
}
```

```
make_coffee() {
  lk.acquire();
  coffee++;
  coffee_cv.signal();
  lk.release();
}
```

Implement A Sleeplock

lock lk; bool free = False; Condvar lock_cv;

```
lock_acquire() {
  lk.acquire();
  while (!free) {
    lock_cv.wait(lk);
  }
  free = False;
  lk.release();
}
```

→ Safety? ✓

→ Progress? ✓

→ Bounded Waiting? X

```
lock_release() {
  lk.acquire(); // assert(!free);
  free = True;
  lock_cv.signal();
  lk.release();
}
```

Implement A Fair Sleeplock w/ Bounded Waiting

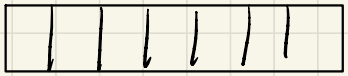
- → Lock must be acquire in FIFO order
- → new thread can't acquire the lock till all prior waiters have done so.

```
lock lk;
bool free = False;
Condvar lock_cv;
wait_queue = [];
Condvar wait_cv;

lock_acquire() {
    lk.acquire();
    wait_queue.append(self);
    while (wait_queue.head() != self) {
        wait_cv.wait(lk);
    }
    while (!free) {
        lock_cv.wait(lk);
    }
    free = False;
    wait_queue.remove_head();
    wait_cv.signal();
    lk.release();
}

lock_release() {
    lk.acquire();
    free = True;
    lock_cv.signal();
    lk.release();
}
```

Producer Consumer / Bounded Buffer Problem



fixed size buffer of N entry

Producer = Put one item into the next empty slot, blocks until buffer has room to put

Consumer = Removes an item from the next non empty slot, blocks until buffer is not empty.

```
Lock lk;
Item buffer[N];
int consume_ofs = 0;   // read offset
int produce_ofs = 0;   // write offset
int total_items = 0;   // available items in the buffer

produce(item) { }      consume() { }
```

Lock lk; Item buffer[N]; Condvar not_full_cv; Condvar not_empty_cv;

int consume_ofs = 0; int produce_ofs = 0; int total_items = 0;

produce(Item) {

lk.acquire();

lk.release();

}

consume() {

lk.acquire();

lk.release();

}