10/24

Add synchronization to the following code

```
char* book;          // shared
int num_copies;      // shared
```

lock;

① one writer at a time
② no writer when there are readers reading

```
function browse_book() {
    lock.acquire(); => read_lock();
    display(book, num_copies);
    lock.release(); read_unlock();
}
```

100 : 1 , 1000 : 1

```
function checkout_book() {
    lock.acquire(); -> write_lock();
    if(num_copies>0) {
        num_copies--;
    } else {
        book = next_book();
        num_copies = get_copies(book);
    }
    lock.release(); -> write_unlock();
}
```

Reader Writer locks.

API: read_lock(), read_unlock, write_lock(), write_unlock();
(start_read) (end_read) (start_write) (end_write)

① Mutual Exclusion for writers (1 at a time, no readers can read during a write)

② Multiple readers when there is no writer

```
active_readers = 0;
active_writers = 0;   // only ever be 1 or 0
waiting_readers = 0;  // blocked by writers
waiting_writers = 0;  // blocked by readers.
```

```
lock;  // mutual exclusion.
readcv;
writecv;
```

```
function read_lock(){
  lock.acquire();
  waiting_readers++;
  while (active_writers > 0 || waiting_writers > 0){
    readcv.wait();
  }
  waiting_readers--;
  active_readers++;
  lock.release();
}
```

```
function read_unlock(){
  lock.acquire();
  active_readers--;
  if(active_readers == 0 && waiting_writers > 0){
    writecv.signal();
  }
  lock.release();
}
```

```
function write_lock(){
  lock.acquire();
  waiting_writers++;
  while(active_readers>0 || active_writers>0){
    writecv.wait();
  }
  waiting_writers--;
  active_writers++;
  lock.release();
}
```

```
function write_unlock(){
  lock.acquire();
  active_writers--;
  if(waiting_writers>0){
    writecv.signal();
  }else{
    readcv.broadcast();
  }
  lock.release();
}
```

active_readers = 0 → 1 → 0

active_writers = 0 → 1 → 0

waiting_readers = 0 → 1 → 0 → 1 ✗

waiting_writers = 0 → 1 → 0

lock = free → busy → free → busy → free → busy → free → busy → free

readcv = [ ] → [C]

writecv = [ ] → [B]

ready_list = [ ] → [B] → [ ] → [A, C] → [C, B]

```
function read_lock() {
  lock.acquire();
  waiting_readers++;
  while (active_writers > 0
  || waiting_writers > 0) {
C   readcv.wait();
  }
  waiting_readers--;
  active_readers++;
  lock.release();
}
```

```
function read_unlock() {
  lock.acquire();
  active_readers--;
  if (active_readers == 0 &&
      waiting_writers > 0) {
    writecv.signal();
  }
  lock.release();
}
```

```
function write_lock() {
  lock.acquire();
  waiting_writers++;
  while (active_readers > 0
  || active_writers > 0) {
    writecv.wait();
  }
  waiting_writers--;
  active_writers++;
  lock.release();
}
```

```
function write_unlock() {
  lock.acquire();
  active_writers--;
  if (waiting_writers > 0) {
    writecv.signal();
  } else {
B   readcv.broadcast();
  }
  lock.release();
}
```

A: reader, calls read_lock(), finishes & reads data [interrupted & switched to B], read_unlock() [switched to C]

B: writer, calls write_lock(), blocks till signaled [switched to A], finish write_lock()

C: reader, calls read_lock(), blocks till signal [switched to B]