Midterm Review

Topic List:

- **Kernel**
  - Kernel/User Separation
  - Privilege Mode
  - System Calls
  - Monolithic vs Microkernels

- **Memory Management**
  - Fragmentation
  - Segmentation
  - Virtual Memory
  - Locality
  - Paging
    - Multi-Level Page Tables
    - Page Replacement

- **Processes**
  - Process State
  - Process Transitions
    - Interrupts, Traps, Exceptions
  - PCB

- **Threads**
  - Kernel vs User Threads
  - Concurrency and Parallelism
  - TCB
Synchronization

x86 Hardware Provided Timer Interrupts:

```c
int splx(LOW)
    Turns interrupts on. Returns old interrupt value.

int splx(HIGH)
    Turns interrupts off. Returns old interrupt value.
```

x86 Hardware Provided Atomic Testing:

```c
int atomic_test_and_set(lock *l)
    Returns 1 if lock is in use.
    Returns 0 if lock is acquired.

void atomic_clear(lock *l)
    Releases the lock.
```

Mutex:

```c
lock(mutex_t lock) -
    Acquire the lock, blocking if necessary.

unlock(mutex_t lock) -
    Release the lock. Assumed that the calling thread owns the lock
```

Condition Variables:

```c
wait(cond_t cv, mutex_t lock)
    Block the calling thread until the condition has been signaled.
    Atomically does the following:
    1. Releases lock.
    2. Add thread to the waiters for cond.
    3. Sleeps thread until awoken.

signal(cond_t cv)
    Signal that the condition has been met, awakening a single waiting thread. 
    (Though not switching to the newly awoken thread immediately.)

broadcast(cond_t cv)
    Signal that the condition has been met, awakening all waiting threads.
```
Kernel Threads: Created and scheduled by the kernel. These are the threads being run on the CPU.
User Threads: Created by a threading library and scheduling is managed in user space. Much faster to context switch between these as they share address space.

User Thread States + Transitions:

- Running
- Start
- Ready
- Done
- Blocked