Birrell paper

1. **[5 points]** Why must the condition variable `wait(cv,mutex)` primitive be implemented atomically? Why can't we just say:

```c
lock( mutex );
...
if ( !condition ) {
    unlock( mutex );
    wait( cv );
    lock( mutex );
}
...
unlock( mutex );
```
2. [5 points] What problem does the following code transformation avoid on multiprocessors?

before:

lock
...
...
...
signal( cv );
unlock

after:

lock
...
...
...
unlock
signal( cv );
Lewis and Berg chapters

3. [5 points] Lewis and Berg list 11 advantages of using threads! Describe 5 advantages.
4. [10 points] Lewis and Berg have an interesting way of describing the relationship between counting semaphores and condition variables. In class I said something like “Condition variables can be viewed as generalized semaphores that have been ‘ripped open’”. Explain.
Solaris papers

5. [5 points] What is a thread stack “red zone” as described in the Solaris implementation papers?
6. [10 points] How does the Solaris user-level threads library automatically “manage” the number of light-weight processes (LWPs) allocated to a process? Be sure to mention SIGWAITING.
Psyche paper

7. [5 points] Traditional user-level threads packages intercept blocking system calls and replace them with asynchronous or non-blocking calls. Why? What clever technique does the Psyche system use to avoid this transformation?
8. [5 points] What do the Psyche designers mean when they say that user-level threads in their system are “first class”?
Bloom thesis

9. [10 points] Serializers provide the following primitives:

    wait( queue ) until ( condition )
    join( crowd ) { /* inside crowd */ }
    empty?( queue );
    empty?( crowd );

Using these primitives construct a solution to the FCFS readers/writers problem. (If there is a writer waiting, readers that arrive subsequently may not read until the writer is finished, even though there are currently readers reading.) (Hint: This isn't really that hard. You just need to queue arriving readers and writers and let them "in" at the appropriate times.)
10. [5 points] Give a path expression that specifies a solution to the readers/writers problem without the FCFS property. In other words, arriving readers are allowed to read even if a writer is waiting. (Hint: This is easy.)
Anderson paper

11. [10 points] Why does the “test-and-test-and-set” (read from cache and then try test-and-set) spinlock solution perform poorly under high contention?
12. [5 points] Complete the following code for Anderson’s queuing solution from the paper. (P is the maximum number of processes).

init:
    flags[0] = HAS_LOCK;
    flags[1..P-1] = MUST_WAIT;

lock:
    myPlace = ???
    while ( ??? )
        
        flags[ myPlace mod P ] = MUST_WAIT;

unlock:
    flags[ (myPlace + 1) mod P ] = HAS_LOCK;