CSSE 332 -- OPERATING SYSTEMS

Introduction to Condition Variables

Name: SOLUTION KEY

Question 1. (5 points) In your own words, define what are condition variables are and what they allow you to do.

Solution: Answers vary, add your own definition based on the lecture. Use this as your way to assess if you understand condition variables.

Question 2. Write down the API call that corresponds to each of the actions below.

(a) (5 points) Create a condition variable `c`:

`pthread_cond_t c = PTHREAD_COND_INITIALIZER;`

(b) (5 points) Given a condition variable `c` and a mutex `m`, wait on the condition variable:

`pthread_cond_wait(&c, &m);`

(c) (5 points) Given a condition variable `c`, signal **exactly one** waiting thread, if any.

`pthread_cond_signal(&c);`

(d) (5 points) Given a condition variable `c`, signal **all** waiting threads, if any.

`pthread_cond_broadcast(&c);`

Question 3. Consider a thread that calls `pthread_cond_wait(&c, &m);` where `c` and `m` are a condition variable and a mutex lock, respectively.

(a) (5 points) Describe the steps performed by the thread as it is ready to wait on the condition variable.

Solution: First we assume that the thread *owns* the lock `m`. Then, in an atomic fashion, do the following:

- Release the lock `m`.
- Enter a sleep state awaiting for a signal or a broadcast.

Note that the behavior is undefined if the thread does not have the mutex `m` locked when it calls this function.

(b) (5 points) Assume now that another thread calls `pthread_cond_signal(&c)`. Describe the steps taken by the waiting thread when it gets signaled.

Solution: The thread will enter the `READY` state awaiting for the scheduler to put into active running on the CPU. When it enters the `RUNNING` state, it will attempt to **grab** the lock `m`. **Note** that entering the `READY` state does not necessarily mean that the thread enters execution, it simply means it is ready for the scheduler to pick it up next.

If `m` is unlocked and ready, the thread will lock it and then continue with its execution.

If `m` is not unlocked (i.e., locked by another thread), then the thread will enter the `SLEEP` state again waiting for the mutex.