CSSE 332 -- Operating Systems

Introduction to Condition Variables

Name: Solution Key

Question 1. 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 2. 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.

Question 3. (5 points) In the boxes below, write down a possible implementation of `pthread_join` using condition variables.

First, list your state of the world (or concurrency state). These will essentially be your global variables.

Solution:

```
1 int child_done = 0;
2 pthread_cond_t c = PTHREAD_COND_INITIALIZER;
3 pthread_mutex_t lk = PTHREAD_MUTEX_INITIALIZER;
```

Parent (main) thread:

Solution:

```
1 int child_done = 0;
2 pthread_cond_t c =
      PTHREAD_COND_INITIALIZER;
3 pthread_mutex_t lk =
      PTHREAD_MUTEX_INITIALIZER;
```

Child thread:

Solution:

```
1 int child_done = 0;
2 pthread_cond_t c =
      PTHREAD_COND_INITIALIZER;
3 pthread_mutex_t lk =
      PTHREAD_MUTEX_INITIALIZER;
```