CSSE 332 -- OPERATING SYSTEMS

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:

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

Parent (main) thread:

Solution:

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

Child thread:

Solution:

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

Question 4. (5 points) Consider the following sequence of events, we have three threads, $\mathbf{T}_1$, $\mathbf{T}_2$, and $\mathbf{T}_3$. Also, assume that $t_1 < t_2 < t_3$.

Time	Thread	Event
...	...	...
t_1	$\mathbf{T}_1$	`pthread_cond_wait(&c, &m);`
...	...	...
...	...	...
t_2	$\mathbf{T}_2$	`pthread_cond_wait(&c, &m);`
...	...	...
...	...	...
t_3	$\mathbf{T}_3$	`pthread_cond_signal(&c);`

Some time after t_3, which one of the waiting threads ($\mathbf{T}_1$ and $\mathbf{T}_2$) would wake up and start executing?

A. $\mathbf{T}_1$.
B. $\mathbf{T}_2$.
C. Neither $\mathbf{T}_1$ nor $\mathbf{T}_2$.
D. Other: **Cannot determine**

Question 5. (15 points) The following pieces of code contains errors, find and fix these errors.

```c
pthread_cond_t c = PTHREAD_COND_INITIALIZER;
pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;

void *thread1(void *unused) {
  // some code here...

  // need to wait on a condition variable
  while(!ready) {
    pthread_cond_wait(&c, &m);
  }
}

void *thread2(void *unused) {
  // some code here

  ready = 1;
  pthread_cond_signal(&c);
}
```

Thread 1 is accessing the `ready` variable without having the lock `m`.

```c
pthread_cond_t c = PTHREAD_COND_INITIALIZER;
pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;

void *thread1(void *unused) {
  // some code here...

  // need to wait on a condition variable
  pthread_cond_wait(&c, &m);
}

void *thread2(void *unused) {
  // some code here

  pthread_mutex_lock(&lock);
  ready = 1;
  pthread_cond_signal(&c);
  pthread_mutex_unlock(&lock);
}
```

Thread 1 does not check any conditions on the `ready` state variable.

```c
pthread_cond_t c = PTHREAD_COND_INITIALIZER;
pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;

void *thread1(void *unused) {
  // some code here...

  // need to wait on a condition variable
  pthread_mutex_lock(&lock);
  if(!ready) {
    pthread_cond_wait(&c, &m);
  }
  pthread_mutex_unlock(&lock);
}

void *thread2(void *unused) {
  // some code here

  pthread_mutex_lock(&lock);
  ready = 1;
  pthread_cond_signal(&c);
  pthread_mutex_unlock(&lock);
}
```

Thread 1 should **always** use a `while` loop before waiting on a condition variable.