Information Technology University of the Punjab
SE301T Operating Systems – Spring 2025
Quiz 5 [CLO3] –May 10, 2025

Name: _____Solution_____ **Roll No.: _____**
Time allowed: Attempt before midnight **Maximum Marks:** 20

---

1. Which atomic instruction is used to implement ticket locks? [1]
   Fetch-and-add

2. Mention one solution that we have studied to overcome spinning inside lock functions. [1]
   Solution 1: Calling yield() inside the lock acquire function instead of spinning:

```
void lock() {
while (TestAndSet (&flag, 1) == 1)
      yield();
}
```

   Solution 2: Using queues
   Park, setpark and unpark

3. The atomic instructions that we studied, such as compare-and-swap, test-and-set, fetch-and-add can only be used to implement spin locks. True or false? Justify your answer. [2]
   False! They can be used to create locks that do not spin, rather yield. See code in the answer of question 2 to see how TestAndSet is used with yield(). This implementation will result in a lock that does not spin.

4. What is the wakeup/waiting race condition when queues are used? Copy-paste code from the slides to explain where and how it can occur. Also describe the solution provided by Solaris to overcome this race condition. [4]
   When "guard lock" has been released in a thread that is going to park, this guard lock can be acquired by another thread that calls unpark. So, this thread may run and call unpark (call to **wakeup** a waiting thread) before the other thread had called park (call to **sleepwait** in a queue). So, when the other thread calls park, it has no other thread to wake it up or unpark it.

```
queue_add(m->q, gettid());
m->guard = 0;
park();
}
```

   The solution is that the thread should let the system know before parking that it is going to park. So, Solaris provides another system call called setpark which is called before releasing the guard lock. This ensures that in the case of above scenario, a thread that calls park **after** some other thread has called unpark, it does not go to sleep.

```
queue_add(m->q, gettid());
setpark(); // new code
m->guard = 0;
park();
```

5. What do the pthread_cond_wait and pthread_cond_signal functions do? How are they used (provide in pseudocode the template for using these)? What is the role of the state variable? [4]
   pthread_cond_wait is called by a thread that has to wait on a condition.
   pthread_cond_signal is called by a thread that has to complete a condition and wakeup any waiting threads on that signal.
   The state variable is used as a binary check variable used in both waiting and signaling threads. The waiting thread checks if this variable has been set, symbolizing that the work it was waiting to be completed has indeed completed. While in the signaling thread, the state variable is set after the work this thread had to complete has completed.
   Template for pthread_cond_wait with state variable done and lock m:

```
pthread_mutex_lock(&m);
while (done == 0)
    pthread_cond_wait (&c, &m);
// operation that must be done after state variable has been set.
```

Template for pthread_cond_signal with state variable done and lock m:

```
// operation that should be done before signaling the waiting thread to continue.
pthread_mutex_lock(&m);
done = 1;
pthread_mutex_signal (&c);
pthread_mutex_unlock(&m);
```

6. Two threads T1 and T2 change the value of a variable x to 10 and 20 respectively. Both acquire a lock lk when making this change and release it when they have made the change. Suppose the value of x is printed after both threads have run. What would be the printed value? How can you make sure that the printed value is always the value set by T2? [3]

   The value of x printed after both threads have run can be either 10 or 20 as, lock alone does not impose any order of operation for the threads, rather it only provide mutual exclusion while accessing x in T1 and T2. To get the printed value of x equal to what is set by T2, we need to change x in T1 before changing it in T2. To ensure this order, we can use condition variable inside T2 to wait for T1 to change x and inside T1 to signal when it has changed the value of x. This will ensure that the final value of x is the one assigned by T2.

7. To what value is semaphore initialized when used as (a) mutex (b) condition variable? [2]

   (a) Mutex: 1
   (b) Condition variable: 0

8. How different is the use of semaphore as a condition variable from using the actual pthread condition variables? [3]

   In the case of implementing a **single** condition variable using semaphores, we can avoid using a state variable and lock. But **if two or more conditions are to be implemented between threads of two types, where there may be more than one threads of a type**, for example in the case of a producer-consumer problem with multiple producers, we will have to use mutex and in some cases a state variable.