50.005 Quiz OS 4 (15 mins)

Name: _**Sample Solutions**_____________ Student ID: _________________

Note: During the quiz, you can consult written or printed materials. But you can't go online or look at anything electronic, including your laptop, smartphone, etc.

1. The two basic *kinds* of process/thread synchronisation problems (e.g., found in the producer-consumer problem) are _**mutual exclusion [1pt]**_ and _**condition synchronization [1pt]**_ .

2. In class, we discussed the following proposed solution to the critical section:

```
while (true) {
    while (turn != 0)
        ;
    // critical section
    …
    turn = 1;
    // remainder section
    …
}
```

The synchronization is between two processes 0 and 1, which share the turn variable. The code shown is for Process 0; the code for Process 1 is analogous. Argue that the proposed solution satisfies mutual exclusion.

Assume a process is in the critical section (CS) and before it exits, the other process enters. Wlog, assume 0 loads the value of turn first in the inner while loop. Case 1: the loaded value is 1. In this case, 0 can't enter the CS, i.e., contradiction. Case 2: the loaded value is 0. In this case, turn must still be 0 when 1 loads its value in the inner while loop. Hence, 1 can't enter, i.e., contradiction. [3pts]

Accept alternative answers and give partial credit where appropriate.

3. Consider the producer-consumer problem. The excerpt producer code shown on the next page uses a named reentrant lock and named condition variables in Java.

(a) When the producer calls empty.await(), what happens to the reentrant lock mutex?

The reentrant lock mutex is released by the producer. [2pts]

(b) Give a suitable actual Java statement for the **<statement X>** placeholder in the shown code excerpt.

full.signal(); [2pts]

Accept full.release() or full.notify() as alternative answer, although it isn't strictly correct.

```
Lock mutex = new ReentrantLock();
Condition empty = mutex.newCondition();  // an empty buffer slot is available
Condition full = mutex.newCondition();       // a full buffer slot is available

public void produce (Item E) {
          mutex.lock();
          try {
                  while (count == BUFFER_SIZE)
                                 empty.await();
                  // critical section – put E into shared buffer
                  …
                  count++;

                  // inform consumer of full slot produced
                  <statement X>
          } catch (InterruptedException e) { }
          finally {
                   mutex.unlock();
          }
  }
```