**Part 6d. Producer/consumer implementation with locks and semaphores [20 points]**

Assume you have the following code for accessing a shared-buffer that contains `max` elements (for some very large value of max). Assume multiple producer and multiple consumer threads access these routines concurrently. Assume the initial state is that the mutex is not held and that all buffers are empty. Assume the semaphore `empty` is initialized to 0 and `fill` is initialized to `max`. *Assume numfull is initialized to 0.*

```
void *producer(void *arg) {
            Mutex_lock(&m);                          // p1
            if (numfull == max)                      // p2
                  sema_wait(&empty);                 // p3
            do_fill(i);  //updates numfull           // p4
            sema_post(&fill);                        // p5
            Mutex_unlock(&m);                        // p6
}
void *consumer(void *arg) {
            Mutex_lock(&m);                          // c1
            if (numfull == 0)                        // c2
                  sema_wait(&fill);                  // c3
            int tmp = do_get(); // updates numfull   // c4
            sema_post(&empty);                       // c5
            Mutex_unlock(&m);                        // c6
 }
```

79) After the instruction stream "PPPPP" (i.e., after the scheduler runs 5 lines producer() for a producer thread P), which line of acquire will be executed for P when P is scheduled again?

a) P1
b) P3
c) P5
d) Some line after P6
e) None of the above

*Answer: d. Mutex is not locked, numfull != max, so just runs through all lines: p1, p2, p4, p5, p6. Next time it is scheduled, it will run a line after P6.*

80) Assume the scheduler continues on with CCCCC (i.e., the scheduler runs 5 lines of consumer() for a consumer thread C and the full instruction stream is PPPPPCCCCC). Which line will C execute when it is scheduled again?

a) c1
b) c3
c) c5
d) Some line after c6
e) None of the above

*Answer: d. Mutex is not locked, numfull is 1, so just runs through lines: c1, c2, c4, c5, c6. Next instruction will be after c6.*

81) Assume the scheduler starts another consumer with OOO (i.e., the full instruction stream is PPPPPCCCCCOOO). Which line will thread O execute when it is scheduled again?

a) c1
b) c3
c) c4
d) Some line after c6
e) None of the above

*Answer: c4 -> c. Mutex is not locked, but numfull is 0, so executes c1, c2, c3. Now, we see that the consumer is incorrectly waiting on a semaphore fill that was initialized to numfull. As a result, the consumer does NOT end up waiting here and the next time it runs it will execute c4.*

82) Assume the scheduler starts another producer with RRR (i.e., the full instruction stream is PPPPPCCCCCOOORRR). Which line will thread R execute when it is scheduled again?

a) p1
b) p3

c) p5
d) Some line after p6
e) None of the above

*Answer: p1->a. Since the mutex is currently held by O, R will be stuck waiting to acquire the lock.*

83) If a problem exists in the above code, what would be the easiest solution to fix it?

a) There is no problem with this code
b) Remove the calls to mutex_lock() and mutex_unlock()
c) Correct how the two semaphores were initialized
d) Change the semaphores to condition variables
e) None of the above

*Answer: d or e (points given after initial grading). A common answer was to say that the code would work if we corrected how the semaphores were initialized – but this is actually not enough! Semaphores do NOT release the mutex when sema_wait() is called; therefore, the system will be in a deadlock if a process calls sema_wait() because it will hold the mutex and the process that needs to call sema_post() will never be able to grab the lock and run. The only correct way to fix the code is to use condition variables (which do not require initialization) AND change the if() loop to a while() loop.*

**End of Exam! Congratulations!**