OS Tutorial 3 – Concurrency and Locks

Week 7

Question 1.

The following program consists of two concurrent threads that shared the variable x.

```
int x = 1;
Thread T1:  Do_something; x = x*2; print x
Thread T2:  Read 4 values; x = x+4;
```

Give all possible final values of the variable x after the two threads run.

Question 2.

The program was extended to three concurrent threads and two locks were added.

```
int x = 1;
lock_t   lc1, lc2;
Thread T1: Do_something; lock(lc1);lock(lc2); x = x*2;
           unlock(lc2); unlock(lc1);
Thread T2:  lock(lc2); x = x+4; unlock(lc2);
Thread T3:  lock(lc1); x = x*x; unlock(lc1);
```

Remember that any thread may attempt to run *in any order*.

a) Describe one order that works well and another possible order that will result in a race condition.
b) Will the use of a multicore processor increase or decrease the chance to get the race condition
c) Can you fix the use of the locks to achieve the desired outcome.

Question 3.

Below you can read a solution of the Producer-Consumer Problem that uses a lock M in combination with two condition variables to make the producer sleep when the buffer is full, and to make the consumer sleep when the buffer is empty.

Note the shared buffer **b** is a circular buffer of capacity for n items. The variable *in* points to the next gap, and the variable *out* points to the item to be consumed. When both point to the same item, it means the buffer if empty. When *in* points to the item before *out*, it means the buffer is full.

Producer

```
while (true) {
  /*produce item v */
  pt_lock (&M);
  while ((in + 1) % n == out)
     pt_cond_wait(&Out_CV, &M);
  b [in] = v;
  in = (in + 1) % n;
  pt_cond_signal (&In_CV);
  pt_unlock (&M);
}
```

Consumer

```
while (true) {
  pt_lock (&M);
  while (in == out)
    pt_cond_wait(&In_CV, &M);
  w = b [out];
  out = (out + 1) % n;
  pt_unlock (&M);
  pt_cond_signal (&Out_CV);
  /*consume item w */
}
```

The semantics for the condition variables operations are explained below:

```
pt_cond_wait (&CV, &Mutex);
```

1. unlock the mutex
2. sleep for a while (may wake up at any time)
3. relock the mutex

```
pt_cond_signal (&CV);
```

wake up at least one of the threads (if any) that is sleeping on the CV

The analogy to a person sleeping in a side room fits the *pthread_cond_signal* operation. A thread that is waiting on a call to *pthread_cond_wait* may wake up at any time, just as a person who goes to bed may wake up before the alarm clock goes off.

When some other thread calls *pthread_cond_signal* for the given CV, the effect is similar to someone ringing a loud alarm bell in the side "room" that corresponds to the CV. If there is a thread waiting on a CV, it is guaranteed to wake up when that CV is signaled. If there are several threads waiting on the same CV, *at least one* is guaranteed to wake up. (Whether more than one wakes up depends on the implementation.) If there is no thread waiting on the CV, the *pthread_cond_signal* call has no effect.

Explain what happens when:

a) The buffer is full, and the producer wants to put one item.

b) The buffer contains 3 items, and the consumer reads one item.

c) The buffer contains 1 item, and the consumer reads that item.