# C211 – Operating Systems

# Tutorial: Synchronisation

1. Explain why the following statement is false: When several threads access shared information in main memory, mutual exclusion must be enforced to prevent the production of indeterminate results.
2. Discuss the pros and cons of busy waiting.
3. One requirement in the implementation of the semaphore operations `up` and `down` is that each of these operations must be executed atomically; i.e., once started, each operation runs to completion without interruption. Give an example of a simple situation in which, if these operations are not executed atomically, mutual exclusion may not be properly enforced.
4. Can two threads in the same process synchronize using a kernel semaphore if the threads are implemented by the kernel? What if they are implemented in user space? Assume no threads in other processes have access to the semaphore. Discuss your answers.
5. Does the strict alternation solution work the same way when process scheduling is preemptive?
6. Give a sketch of how a uniprocessor operating system that can disable interrupts could implement semaphores.
7. Consider the following three threads:

```
T1:          T2:         T3:
a = 1;       b = 1;      a = 2;
b = 2;
```

   (a) Show all possible thread interleavings.

   (b) If all thread interleavings are as likely to occur, what is the probability to have a=1 and b=1 after all threads complete execution?

   (c) What about a=2 and b=2?

8. Synchronization within monitors uses condition variables and two special operations, `wait` and `signal`. A more general form of synchronization would be to have a single primitive, `waituntil`, that had an arbitrary Boolean predicate as parameter. Thus, one could say, for example,

$$\texttt{waituntil} \quad x < 0 \quad \texttt{or} \quad y + z < n$$

   The `signal` primitive would be no longer needed. This scheme is clearly more general than that of Hoare, but it is not used. Why not? (Hint: think about the implementation.)