Tutorial 4
Synchronization tools & examples – Part I & II & III
Operating Systems Comp Sci 3SH3 Term 2, Winter 2022
Prof. Neerja Mhaskar

Tutorials are not mandatory. They are simply a tool for you to understand the course concepts better.
Tutorial Format: The questions will be posted a day before or on the day of the tutorial on the course website. You can choose to solve these problems before hand and come in with your solutions. I or one of the TAs helping me will check your solutions. If you have all of the questions correct you can choose to leave. If you have any of them incorrect, it is recommended that you stay and understand the solutions.

Solutions to the tutorial will not be posted online.

1. Write pseudocode that Illustrates how a binary semaphore can be used to implement mutual exclusion among *n* processes.

2. What is the difference between a semaphore and conditional variable?

3. In the solution for dining philosophers problem using monitors, provide a scenario in which a philosopher may starve to death.

4. Design an algorithm for a bounded-buffer monitor in which the buffers (portions) are embedded within the monitor itself.

5. Explain what will happen if a process A locks the associated mutex before calling `pthread cond wait( )`, but another process B does not lock the associated mutex before call `pthread_cond_signal( )`.

6. Implement a mutex lock using the `test_and_set()` atomic hardware instruction. Assume that the following structure defining the mutex lock is available:

   ```
   typedef struct {
        bool held;
   } lock;
   ```

   Also, `available == 0` indicates that the lock is available, and `available == 1` indicates that the `lock` is unavailable. Using the `struct lock`, illustrate how the following functions can be implemented using the `test_and_set()` instructions:
   - void acquire(lock *mutex)
   - void release(lock *mutex)

   Be sure to include any initialization that may be necessary.

7. Explain why interrupts are not appropriate for implementing synchronization primitives in multiprocessor systems.