4 Exercise #4: Mutual Exclusion and Deadlocks

In this exercise you will implement different variants of mutexes in LWT. These primitives will be able to prevent or detect deadlocks, and to be used recursively.

4.1 Fast Mutexes

Update the interface of your LWT implementation. Implement a "fast" mutex that ignores the possibility of deadlocks. This implementation is probably very similar to the mutex you have developed in previous assignments.

4.2 Recursive Mutex

Extend your mutex implementation in a configurable manner to allow for recursive use by its holder. It should be possible to wait on a condition variable while being in a deep "recursion". Recursive and fast mutexes should be able to co-exist within the same application.

4.3 Deadlock Prevention

One possibility to prevent deadlocks is to enumerate all locks and to force threads to acquire locks in the increasing order defined by their enumeration. Implement this deadlock-prevention strategy for your mutex implementation. Insert checks to the LWT library to prevent threads form locking and unlocking in the wrong order. Make this change configurable, too. This deadlock prevention strategy should also work with recursive mutexes (`ORDERED` and `RECURSIVE` flags).

4.4 Deadlock Detection

Sometimes it is not convenient to use a deadlock-prevention strategy. In this case, deadlock detection can help debugging parallel code that causes deadlocks. Keep track of the holder of a mutex and on which mutex a thread waits. When a mutex is already locked, check if a deadlock would happen when the current thread waits on the mutex. Follow the path of the mutex-holding thread and check if it also waits on a different mutex and so on. If a cycle is detected, print this cycle and the participating threads and mutexes in-order and terminate the program. To prevent false positives, detecting deadlocks must be atomic relative to other threads performing operations on mutexes. Deadlock detection should be also a configurable feature of the mutex implementation and should also work with recursive mutexes.

4.5 Tests

Write tests for your implementations and configurations. Show that deadlock prevention and detection work by detecting faulty test cases. Also write a test program that triggers the deadlock prevention even though the program cannot deadlock.

Remarks:

- Make sure the deadlock detection and prevention strategies cannot deadlock themselves.
- Deadline: 16.1.2017, 24:00