Lab 2: Critical region

This lab should be done in a team with each student assigned individual item. It is preferable to use the same implementation language by all team members. Supposed implementation languages are Java and C++. Implementation of some items, e.g. 6, in C requires modeling interfaces with structures consisting of function pointers or extensive usage of macros.

1. Demonstrate race condition on two threads manipulating counter. Use Dekker's algorithm to prevent data corruption. Develop DekkerLock type that provides FixnumLock interface or concept in the case of C++ (see 6). Provide framework that checks if a given locking primitive satisfies mutual exclusion condition.
2. Experiment with threads synchronously accessing counter using atomic, spinlock, monitor primitives etc. Utilize java.util.concurrent.Lock interface or Lockable concept to implement benchmarking framework. Implement missing primitives (such as spinlock in Java) and use primitives implemented in 1, 3, 5. Compare performances of different primitives. Try to estimate thread contention.
3. Realize Lamport algorithm (bakery algorithm). Develop BakeryLock type that provides FixnumLock interface or concept in the case of C++ (see 6). Check how counter value is drifting when some threads are increasing and some decreasing counter with average 0.
4. Simulate deadlock and item loss in producer-consumer problem solution with sleep-wakeup primitives.
5. Using atomic variable improve Lamport mutual exclusion algorithm (bakery algorithm) so that every thread would receive unique ticket. Consider the problem of bounding the value of this variable.Develop ImrovedBakeryLock type that provides java.util.concurrent.Lock interface or Lockable concept in the case of C++ (see 6)
6. Develop new interface FixnumLock (or concept FixnumLockable) that allows mutual exclusion between limited (fixed) number of threads. It should extend interface java.util.concurrent.Lock (refine concept Lockable) and additionally include methods getId/register/unregister for obtaining id by a thread. Provide logic for these methods in abstract class that can be used to derive classes for mutual exclusion primitives with limited number of threads. Consider additional complexity necessary to implement reset method.