Exam Concurrency & Multithreading

VU University Amsterdam, 21 October 2015, 15:15-18:00

(At this exam, you may use the textbook of Herlihy and Shavit. Answers can be given in English or Dutch. Use of the slides or a laptop is not *allowed.)*

(The exercises in this exam sum up to 90 points; each student gets 10 points bonus.)

1. Consider a computer program containing a method M that cannot be parallelized, while the rest of the program is perfectly parallelizable. Let M account for 30% of the overall computation time. Suppose you run the program on a machine with $n \geq 3$ cores.

(a) Acording to Amdahl's law, what is the speedup (in terms of n) on n cores compared to a single-core machine? (3 pts)

(b) You decide to replace M by an improved version, which is k times faster. What should be the value of k (in terms of n) to at least double the speedup you computed in (a)? (10 pts)

2. Suppose that in the Bakery algorithm we replace the shared variables by regular registers. Does it still provide (1) mutual exclusion and (2) first-come-first-served fairness?

   (For both properties, either argue that this is the case by discussing linearization points, or give a counter-example.) (12 pts)

3. In the construction of an atomic MRMW register from atomic MRSW registers (in Section 4.2.6), reads pick the value with the largest timestamp in the array. Suppose that in case of multiple registers carrying the largest timestamp, reads pick the one with the *smallest* index in the array. Explain how read and write calls can be linearized. (12 pts)

4. Suppose that in the combining tree barrier, the thread that decreases the counter at the root to 0 would invert the sense field at the root before resetting the counter. Give a scenario, with $n = 4$ and $r = d = 2$, to show that then the barrier would be flawed. (10 pts)

5. Consider the unbounded lock-free stack in Section 11.2 of the textbook. Give a scenario to show that in the absence of proper garbage collection, the ABA problem could arise in the `pop()` method. Also explain how (the implementation of) the unbounded lock-free stack could be adapted to avoid this ABA problem. (12 pts)

6. Give a highly parallel multithreaded algorithm for multiplying an $n \times n$ matrix by a length-n vector, which achieves work $\Theta(n^2)$ and critical-path length $\Theta(\log^2 n)$. Analyze the work and critical-path length of your algorithm. (15 pts)

7. Consider the code below for the class `Condition`.

(a) Either argue that this implementation is correct or produce a schedule where a synchronization issue is apparent.

(The expected behavior is specified in the comments of the class. It is assumed that threads share a single `Condition` object and acquire the `Lock` lock before calling `await()` or `signalAll()`.) (12 pts)

```
public class Condition {

  private Lock lock;
  private List<Thread> waitingThreads;

  /** Create a condition variable associated with Lock lock. */
  Condition(Lock lock) {
    this.lock = lock;
    this.waitingThreads = new ArrayList<Thread>();
  }

  /** Block on this condition variable.
   *
   * Requires the current thread to hold the Lock lock. Calling await() will suspend
   * the thread: it will make no further progress unless and until it is resumed. */
  public void await() {
    waitingThreads.add(Thread.currentThread());
    lock.unlock();
    Thread.currentThread().suspend();
    lock.lock();
    waitingThreads.remove(Thread.currentThread());
  }

  /** Signal all threads awaiting this condition.
   *
   * Requires the current thread to hold the Lock lock. Calling signalAll() will
   * resume all threads that suspend due to previous calls of await(). */
  public void signalAll() {
    for (Thread thread : waitingThreads) {
      thread.resume();
    }
  }
}
```

(b) Can you name an important consideration for programming languages which support locking? To be more specific, why was `Thread.suspend()` deprecated in Java, and is `Object.wait()` now the preferred option? (4 pts)