CS-E4110 Concurrent Programming
Autumn 2016 - Tutorials
Weak Memory Models

2016-11-09

Dekker's Algorithm	
boolean wantp ← false, boolean wantq ← false integer turn ← 1	
P	Q
loop forever	loop forever
p1: non-critical section	q1: non-critical section
p2: wantp ← true	q2: wantq ← true
p3: while wantq	q3: while wantp
p4: if turn = 2	q4: if turn = 1
p5: wantp ← false	q5: wantq ← false
p6: await turn = 1	q6: await turn = 2
p7: wantp ← true	q7: wantq ← true
p8: critical section	q8: critical section
p9: turn ← 2	q9: turn ← 1
p10: wantp ← false	q10: wantq ← false

Task 1

Dekker's algorithm is a solution to the critical section problem for two threads.

To familiarize yourself with the algorithm, write a Java implementation for it. Use the Java Memory Model to argue that your solution is correct.

Task 2

Use the x86-TSO abstract machine model to demonstrate why the following naive C/C++ implementation of Dekker's algorithm is incorrect and the two threads P and Q may violate mutual exclusion for the critical section.

```
//Shared variables
bool want_p = false;
bool want_q = false;
int turn = 0;
```

```
//Executed by Thread P
void p_body() {
    want_p = true;
    while(want_q) {
        if(turn == 2) {
            want_p = false;
            while(turn != 1) {
                //busy wait
            }
            want_p = true;
        }
    }
    //critical section
    turn = 2;
    want_p = false;
}
```

```
//Executed by Thread Q
void q_body() {
    want_q = true;
    while(want_p) {
        if(turn == 1) {
            want_q = false;
            while(turn != 2) {
                //busy wait
            }
            want_q = true;
        }
    }
    //critical section
    turn = 1;
    want_q = false;
}
```

Task 3

Introduce memory fences with the `MFENCE` instruction to fix the algorithm from Task 2 for the x86-TSO abstract machine model.

Task 4

This is a demonstration task. Using C++11 low level atomics is not a part of the course scope.

Use C++11 low level atomics to write a portable and correct implementation of Dekker's algorithm.

Recommended reading:

- `http://en.cppreference.com/w/cpp/atomic/atomic`
- `http://en.cppreference.com/w/cpp/atomic/memory_order`
- `http://en.cppreference.com/w/cpp/atomic/atomic_thread_fence`

Task 5

This is a demonstration task. Using C/C++11 low level atomics is not a part of the course scope.

The following C++11 code implements a trivial one-off mechanism for transferring integer values between threads. Explain why it is possible for certain architectures, that for a pair of `put()` and `wait_and_get()` calls, the assertion on line 6 does not fail, but the assertion on line 16 does fails. Assume that the compiler will not reorder any instructions.

Use low level C++11 atomics to write an improved version.

```
1  bool isReady = false;
2  int value = 0;
3
4  //Called from Thread P
5  void put(int i) {
6      assert(i > 0);
7      value = i;
8      isReady = true;
9  }
10
11 //Called from Thread Q
12 int wait_and_get() {
13     while(!isReady) {
14         //busy wait
15     }
16     assert(value > 0);
17     return value;
18 }
```