**Concurrent Algorithms** October 20, 2015

# Exercise 5

In the `ca15_locks.zip` file you will find the skeleton code for a very simple benchmark that uses a number of threads (configured with `-n`) that simply `lock` and then `unlock` one lock object.

**Problem 1.** Your task is to implement the following lock algorithms:

1. Test-and-set (TAS)
2. Test-and-test-and-set (TTAS)
3. Ticket lock

**Problem 2.** For TAS and TTAS, devise solutions where `lock` is implemented with CAS, atomic swap (SWAP), and atomic fetch-and-increment (FAI).

For instance:

```
tas_lock(lock_t* lock)
{
  while(FAI(&lock->val) != 0);
}
```

implements TAS `lock` with FAI.

**Problem 3.** Although x86 (e.g., Intel, AMD) processors offer FAI and SWAP operations, you can still use compare-and-swap (CAS) to implement FAI, or SWAP. Implement FAI and SWAP with CAS and compare the performance of the locks against the native atomic operations.

What do you observe? Why?

**Instructions:**
You can compile all the variations of the test that we need by invoking the following shell script in a Linux terminal:

```
./make_all.sh
```

To run all the variations invoke:

```
./run_all.sh [params]
```

As an example, we have already implemented TAS with CAS, so the output that you will initially get from `run_all.sh` will be:

```
---> run with TICKET
## Start the threads
## Stop the threads
Throughput: 1061.31 Mop/s
# global counter: 556865908  vs.
# of increments : 1112860791
 :-( :-( incorrect, non-atomic counter!
 (your lock / unlock / init implemntation is broken)
---> run with TAS
------> and USE_CAS
## Start the threads
## Stop the threads
Throughput: 13.74 Mop/s
# global counter: 14409995   vs.
# of increments : 14409995
 :-) :-) correct, atomic counter!
 ------------------------------------------------------------- TEST PASSED
...
... output for other lock algorithms/atomic operations
```

with different values for throughput and statistics! Notice that ticket lock that is not yet implemented reports an error, while TAS with CAS works correctly!