**• What changes would you suggest in a code review? Would you check in your code like this? Do you think the code is clean?**

Please see ConcurrentQueue\_codeReview.cpp

**• Do you see room for improving the performance without breaking thread safety?**

Performance could be improved by using a lock-free concurrent queue. This would prevent threads to wait that a lock is released and save expensive context-switching. Such data structures are implemented in Boost library. If the hardware allows it, we could pin key threads (the consumer...) to dedicated CPU cores in order to avoid any preemption.

**• What steps would you take to find the bug?**

If we can reproduce the bug, I would run the program in a debugger. Once it crashes, the debugger will directly show the incriminated line of code and in which thread it occurred. Valgrind can help as well. (I changed the size from 4096 to 2 to make the bug reproducible fast)

If we cannot, I would:

- Stress the queue as much as possible by adding producers who would push different kind of tasks at a high rate in the queue and then try to reproduce the bug quickly.

- Add traces in order to print key information like thread id, methods names, memory address, variables/attributes. However, we have to keep in mind that, even with a low priority dedicated thread who manage the logs, the global behavior of the program will change. I would make all the members of ConcurrentQueue public and log (queue.mWritePtr & queue.mRingModMask) and (queue.mReadPtr& queue.mRingModMask) in the task functor.

- Activate minidump for post-mortem investigation.

- Run the modified program as described above for a long period. Typically, I would launch it on the evening and hope it crashes overnight.

**• If you can find the bug, what fix would you suggest? Write a proper test case to prove that the bug is solved (may be in pseudo code), preferably in a deterministic matter?**

As described in the code review, pop() should return T and not T&.

Concerning the test:

ConcurrentQueue<Functor\*,2> queue;

queue.push(new Functor(…));

auto task = queue.pop();

Functor\* taskCopy = task;

queue.push(new Functor(…));

queue.push(new Functor(…));

EXPECT\_EQ(task, taskCopy);

**• If you cannot find the bug, how would you go about testing the thread safety of the class?** **Give an example of such a test (may be in pseudo code).**

I would go as described two questions above. I would add a lot of producers who would post tasks at a high frequency. I would run this program for a long period and check that it has not crashed.

I would also test that the data sent by the producers are the same than the consumer get. Producers could send data from large arrays and I would check that the consumer got all the elements from those arrays. I would run this test in loop for a long period.