Will Suitor
CSCI Multicore - HW 4

1.
A. Because of how the problem is described, I would say the prime number program should only spawn 1 or 2 threads. If the person is playing a “fast-paced game”, then you would assume those 2 threads would be very computation heavy. The user also probably wouldn’t want their game slowed down while those two threads compete for processors with the prime number program, especially if they’re running it in the background.

B. Here, I would use average throughput as a metric for performance. Execution time would not be an effective metric because if I could engage all four cores with just the prime number program, then the execution time would be much lower. Also, as it is a multithreaded program, one test of throughput would not be enough. The amount of instructions can vary run to run with multithreaded programs, so the better metric would average multiple runs.
Both programs that are running (video game and prime number program) have to include very complex instructions. Therefore, I think it makes more sense to try to minimize the amount of instructions (instead of the simplicity of them). So, to optimize performance here, I would just try to cut down the quantity of instructions as much as I could.

2. The ABA problem is caused by the reuse of memory locations. If a dequeue operation is put on hold while other threads dequeue and enqueue, the original dequeue operation will still complete if memory location it’s removing has been reused. The removal of the node isn’t the problem but it will then point the head of the list to the old, next value which could be anything.

To solve this problem, you have to track the pointers you use and extend them to be 128-bit pointers. You would use half the pointer for the actual memory address and half for the counter associated with the address. Then, you can add an additional compare and swap that will check the pointer’s counters for the next node.

3.
A. Using C++ mutices (not reader_writer locks), I would use a lock for the writer pointer and for the reader pointer.
So, for push(), I would first lock both, see if the writer pointer is one behind the read (meaning buffer is full). Then, I would unlock the read pointer and do the write before unlocking the write lock.
For pop(), I would follow the same pattern. First, lock both, check if there is something to read, unlock the write lock, read and increment the read pointer, unlock the read pointer lock.
For size(), I would lock both, compare the difference in the pointers, unlock and return.

B. Yes, this would be more performant with two reader_writer locks for each pointer. For size, all you need to do is read both pointers. Similarly, when you are checking to see if the buffer is full/empty for push() and pop(), you only need a read lock for the read pointer in push() and a read lock for the write pointer in pop(). This way you reduce the size of the “true” critical section for each function.

C. I don’t believe you could make this more performant with CAS then with reader_writer locks. For example, when you’re using size(), you do not need to update either pointer. However, you could read one of the pointers as it is being written to unless if you surround it with a pointless CAS. So, at the absolute best, you could get the same performance.

4.
A. This breaks an atomicity assumption. Essentially, thread 1 is assuming that if statement it can get past that if statement, then the pointer will be valid for the inner part of the statement. However, thread 2 can come in after thread 1 has gotten past the if statement (but not to fputs()) and set that pointer to be null. Then, you’ll get a segfault.

B. To fix this, you could do a couple of things. The first thing I would do is simply surround that with a mutex. As it is a shared variable, you could guarantee that thread 1 and thread 2 at least do not segfault when run concurrently if they both need to get the same lock. Another solution would be to create a temporary variable in thread one that is equal to thr_glob->proc_info. Then, put that temporary variable in place of thr_glob->proc_info in both places. This solution is fine for this example because all you’re doing with that info is printing it to the stream. However, you were actually trying to access that address, you could not use this solution.