1.a) Minimum 1 and maximum 2 threads. The machine has 4 cores, so only 4 threads can run at a time. You need two threads for running your game, so you have 2 threads left to run your prime program in the background. You need at least one thread to run the program and at max 2 because you're limited by the 4 cores.
b) I would throughput since there are more than one program running (the prime program and the game), and throughput will give me the measurement for how many jobs the machine can run, the average execution rate and the total work getting done. I would try to optimize the system throughput (STP) and this means that I would have to try to increase the time when each program is running in isolation compared to the time when the program runs with other programs.

2. Having a free lists of nodes and reusing nodes is what caused ABA problem. This is because pointer memory address is spatially unique but not temporal unique. The problem occurs when you reuse a node that already has been freed (for example, dequeued) and its pointer pointing to another freed node, your condition to check for CAS will still be satisfied even though the pointer is pointing to the wrong place. As a result, you end up messing up your data structures. You can solve it by having a counter associated with the pointer, so that whenever we re-use the freed pointer we will increment this counter so that the CAS will fail. To help with this, we can use a 128-bit pointer, where 64 bit will be the actual target address and the other 64 will be the counter. That is, when we check in CAS, every time the counter will change and we will fail CAS when we need to. This also requires us to have 128-bit CAS, supported by 64-bit x86.

3. a) Have one global mutex for everything
Push(): call the mutex lock before writing and updating the write pointer and then unlock when we're done. If queue is full then wait (we have a conditional variable associated with this)
Pop(): call the mutex lock before reading, update the reading pointer and then unlock.If queue is empty then wait (we have a conditional variable associated with this)
Size(): call the mutex lock, then calculate the distance between the two pointers and storing in a local variable before unlocking and returning the size

b)I would have a global reader-writer lock for the size() function and a per-element reader-writer lock for each position of the circular buffer. This would allow push and pop to happen at the same time on different positions of the circular buffer, unless they are working on the same position, in which case only one of the function will be able to access that position. Whenever I am calling size, then I will have the global lock the entire structure to make sure that no changes can be made to the circular buffer. I think that I will need to have a lock for each of the reading/writing pointers in case more than one thread is calling either the pop or push function. This makes sure that the pointers aren't change when one particular thread is trying to update it.

c) If it's just push and pop, then I think you can implement using CAS since you just have to have one CAS to check in push for the writer pointer and another CAS in pop for the reader pointer. However, in size, you would need both the reader and the writer pointer, and one single CAS can't check for both pointers to make sure they haven't changed.

4. a) You may pass the condition in thread 1, but then in between the fputs statement and the if condition, thread2 may have changed the global variable to null, and you will be writing nothing to the file instead of what you were supposed to write before. It can also be the case where thread2 sets the global variable to null and preventing the if condition to ever be met in thread1.

b) 1. Adding a mutex lock around the global variable. This way, we make sure only one thread can work on the variable at a time.

```
Thread 1:
Mutex_lock();
If ( thr_glob->proc_info) {
        Fputs(thr_glob->proc_info);
}
Mutex_unlock()

Thread 2:
Mutex_lock();
Thr_glob->proc_info = nullptr;
Mutex_unlock();
```

2. Using CAS in thread2 to set the thr_glob->proc_info to null ptr. IN thread1, we use a local variable to store the proc_info and fputs that local variable since we're not changing the variable, we only want to output it into a file.

```
Thread 1:
Local_proc_info = thr_glob->proc_info;
If (local_proc_info ) {
        Fputs(local_proc_info);

Thread2:
While (true) {
        Local_proc_info = thr_glob->proc_info;
        If (CAS(thr_lob->proc_info, local_proc_info, null) ) {
                Break;
        }
}
```