|  |  |
| --- | --- |
| Program: | GDP1 |
| Course: | INFO6023 – Game Algorithms and Gems |
| Professors: | Daniel Maclam |
| Project # 2: | Threading |
| Weight: | Divided equally over all the projects |
| Due Date: | Friday, March 27th at 11:59 PM |

*Note: This project can be done alone or in pairs*

Description and Purpose

Using either the Win32, c-run-time threading libraries or C11 threads, clearly demonstrate a “significant” speed up in your code, by moving processing onto another thread (create thread) along with dealing with memory contentions (blocking, locking, race conditions, etc.), specifically using:

* Critical Sections
* Buffering data
* Interlocking Exchange functions (in windows or x86/x64 assembler)
* Lockless/Lock Free methods (which usually involve multiple buffers)

NOTE: You are restricted to C++11 threads and C run-time threading calls (\_beginthread, CreateThread, etc.) and the “win32” operating system calls. You may ***not*** use any other threading or asynchronous library without prior approval.

Details

**Using a new, or building on an existing, project, demonstrate (which MUST be a 3D “game” or technical demo, *not* a console application like the 1st project) a “before” and “after” comparison of the performance of your application. You must significantly speed up one key area of the application, that is *visibly* noticeable, so likely at least 3x faster at least**. (i.e. if you showed some random person, who knew nothing about programming, they could easily notice that the “threaded” version was “much” faster)

Keep in mind that “speeding up” can also involve loading other assets in the background; think about it: say you have some giant level that takes only 10 seconds to load at the start, but the user is waiting for all the assets to load. Instead, you (almost instantly) load only a small corridor/room/“landing bay” that the player can navigate around, while the rest of the level loads, then that’s a *massive* speed up. The “trick” is that you *have* to load those other assets on another thread, while the main thread plays the game.

Some suggestions (i.e. things that are slow on modern computers):

* Loading from a hard drive (particularly a mechanical one). Note that spawning a bunch of threads to load files from *the same* drive will actually make it *slower* (as all the threads are “fighting” to read from the same, slow, resource).
* Passing data over the network
* Heavy processing (physics, filtering, etc.), like the physics calculation portion, but note a couple things about “threading” this:
  + It’s hell-a difficult (i.e. it’s certainly not “easy” or “simple”), because of the synchronization
  + There has to be a *large* increase of things simulated to even notice (because modern computers have vast processing *over* capacity)
* Copying from one part of memory to another (again, this is difficult to “thread” and show)

“Threading” isn’t just spawning threads, but involves buffering and locking of some kind, so **must** include:

* The use of **Critical Sections** or other locks
* Where applicable, the **use of buffers and/or shared variables to isolate access to shared data**
* Unless you are specifically needing separate *processes* (and you’ll have to justify why you are doing this), you *not* to use specific *built in* “process” synchronization items like semaphores, mutexes, etc. Note: I’m talking about the *explicit built in* items of these names, not the logical equivalent you can make using threads. Essentially, if the parts of your applications are *sharing the same memory*, then it’s “threading” situation, and if they *aren’t* sharing the same memory, it’s a “process”.

Note: it *is* possible to not have any “locking” with a threaded application, but that’s very, very tricky to do; it’s almost inevitable that there will be some “race conditions” and/or contention for shared data – so telling me that “oh, I don’t use locks because that *never* happens”, isn’t going to fly… unless you can prove this is the case, of course (saying “well, it never crashed so far!” isn’t “proof”, though ☺)

Some notes, restrictions, and clarifications:

* *Be sure to demonstrate the speed up using the* **Release** version of your application, *not* the Debug. Simply switching from Debug to Release can speed an application 100x, and the Debug does a number of “time consuming” things that are masked by adding threads (i.e. the threads might seem to be speeding things up, then switching to release proves that they were pointless, in terms of performance).
* The STL and most of the standard library are \****not\**** “thread safe”, at all. This is one of the main “things you need to learn/deal with” in this project, and also why students saying:

*“Oh. My. Gawd! If we just used boost or whatever, it would be, like, so, easier!”*

…are mis-informed: it’s not the *library* that’s going to “save you”, it’s thinking about how to handle things, programmatically. Also, what if the code you eventually work at (on your job) *doesn’t* use the C++11/boost/whatever libraries? (How likely is that? Answer: essentially a certainty) What are you going to do then?

* Also, OpenGL is \*not\* thread-safe. Like, not even close.
* You may only **use** the STL (Standard Template Library), or your own containers (that were used in Project 1), to answer all questions, unless you are explicitly told otherwise (there is at least one question where you are allowed to use the STL but not certain part of it, for example).
* You may **NOT** use any threading libraries like boost, (TBB) Threading Building Blocks, etc.

Grading Scheme

1. Normally a grade of zero will be assigned to any assignment that is submitted late. However, certain rare exceptions apply according to the *Infotech* *Policy on Missed Evaluations and Evaluation Deadlines*.
2. If you code does not even compile, I will not mark it. Period.
3. If you code does not build (i.e. linker error) and run (i.e. no crazy run-time crash that is unexpected), I may investigate this further, but only if there is some simple problem and/or slight configuration error.

70/12-year old “squinty eye” plagiarism test:

* While you may freely “borrow” mine (or anyone other) code ***but*** your code should be “sufficiently” different from mine.
* You should also give credit for where you found the code, if you’ve used a significant portion of it.
* In other words, you *cannot* simply use an existing game engine (or part of a game engine) to complete this assignment; it should be either completely new of significantly modified.
* How will I determine this?

If I showed your application and/or your source code to either my *very* pragmatic 70-year-old mother, or a typical 12-year-old, and they looked at it, tilted their heads, squinted their eyes, and said, “you know, they look the same,” then they ***are*** the same.

In other words, I’m not going to be drawn into a giant debate over how “different” your code is from mine or anyone else’s, if a complete stranger, say a randomly stopped in the hallways, would conclude that the code and/or output is pretty much the same thing, then it’s “the same.”

Project Corrections

If any corrections or changes are necessary they will be posted to the course web site and you will be notified of any changes in class. It is your responsibility to check the site periodically for changes to the project. Additional resources relating to the project may also be posted.