CSC 481/581 Fall 22 Homework 2: Time and Networking Foundations

Due: 09/28/22 by the start of class

Overview

Your task for this assignment is to explore how to represent time and the basics of constructing a multi-threaded network server. This is an ***individual assignment***, you are to work alone. As always, you are expected to abide by the University's Academic Integrity Policy (`http://policies.ncsu.edu/policy/pol-11-35-01`), which includes providing appropriate attribution for all external sources of information consulted while working on this assignment.

There are 125 points available on this assignment. Students enrolled in 481 are required to complete the first 100 points (Sections 1–4), but may also choose to complete Section 5. Students enrolled in 481 earning scores above 100 will receive a 100 on the assignment (*i.e.*, extra points will not be given as extra credit). Students enrolled in 581 are required to complete all 125 points, and will receive a grade as the percentage of the 125 points they earn. All students are required to submit a writeup addressing the sections of the assignment they have completed—simply submitting code without a writeup discussing that code or including a section in the writeup to cover a part of the assignment for which there is no code submitted will result in zero points for that section of the assignment.

Section 1: Multithreaded Loop Architecture (20 points)

The first part of the assignment is to build upon the `ThreadExample` provided on Moodle to make your main game loop multithreaded. You must have at least two threads handling distinct subsets of the regular update tasks. For example, one thread may handle all of the moving platform object movement and the other thread may handle all of the keyboard input and resulting character changes. This is just a suggestion, you can divide up the work however you see fit and it need not be the same division of work every iteration. Take care to be thread safe (*i.e.*, use getters and setters with mutexes for data access, *etc.*) where necessary, as not doing so can cause strange and unpredictable errors.

Section 2: Measuring and Representing Time (20 points)

The next part of the assignment is to implement an explicit representation of time. You are to have a `Timeline` class, that is flexible enough to represent both real time and local time, can represent time at different scales (*i.e.*, real time vs. game time vs. loop iterations vs. *etc.*). Your class must support an adjustable tic size and enable anchoring your timeline to another, arbitrary timeline (or to some measure of real time). Further, your timelines must support pausing and un-pausing.[1]

To demonstrate your representation of time, you must incorporate the following functionality in your SFML window:

1. Object movement based on elapsed time: You need to maintain a timeline object that covers "game time", and every moving object should update it's position based on the elapsed time of the timeline object. In other words, the moving platforms should have a "velocity."

2. Pause and un-pause: At the press of a key on the keyboard, the timeline object should enter a paused state, and all objects that depend on it for movement should remain stationary. A subsequent press

[1] Hint: It's probably wise to maintain a count of the number of tics that have ellapsed while paused to use when reporting the actual time using the timeline.

of the same button should un-pause the timeline, and objects should begin moving again from the position they were at during the paused period.

3. Changing speed: At the press of one or more additional buttons, the timeline should change it's scale from 0.5 time, 1.0 time (real time), and 2.0 time. The effect should be realized without making any changes to the moving objects themselves, only by changing the timeline object upon which they depend for movement.

Section 3: Networking Basics (20 points)

For this section of the homework, your task is to familiarize yourself with some basic basic networking capabilities. While C++ does provide a socket API, we will instead use the third party 0MQ (ZeroMQ) library. 0MQ is a competing protocol to Google's "Protocol Buffers," and provides a feature-rich set of primatives and an API that facilitates efficient and clean network communications or IPC. You can find 0MQ here: `https://zeromq.org/`. There is an extensive guide available on the 0MQ website, which provides code samples in a variety of languages (including C++). I suggest narrowing your investigation to the `PUB-SUB` model or the `REQ-REP` model. You may choose to investigate others, but those two are sufficient for this assignment.

Once you have familiarized yourself with 0MQ, your task is to implement a simple client-server setup. The client and server are not required to be integrated with SFML for this part of the assignment. The required functionality is:

1. Start a server process that will listen for incoming network connections (0MQ sockets)

2. Start a client process that will connect to the server and receive regular update messages (strings are sufficient, but you may choose to use another type of message without penalty).

3. Each client should receive messages from the server of the form: "Client X: Iteration Y" where X is the `int` representing the order in which the client connected to the server, and Y is the `int` representing the loop iteration. So, for example, if client 1 connects and the server sends it 10 messages, then client 2 connects, both client 1 and client 2 would receive the following two strings: "Client 1: Iteration 11" and "Client 2: Iteration 1".

4. Your program must handle at least 3 simultaneous clients, each of which should run in their own separate process (*i.e.*, not have shared memory), and the clients should be able to be started at any time (*e.g.*, we should be able to start client 2 after 1 second or 30 minutes if we so choose).

Section 4: Putting it all Together (40 points)

For this part of the assignment, your task will be to integrate your SFML window with your 0MQ messaging to enable a networked multiplayer environment. Much like Part 3 of the assignment, the server will be responsible for coordinating clients; however, rather than sending messages, the server should ensure that any movement of the character object on one client gets reflected on the other clients. The server does not need to be a SFML window, but the clients do. Your code from Homework 1 and Parts 1 & 2 of this assignment are probably a good starting point for this effort, since all clients must display the same environment (background if you have it, static rectangles, and moving rectangles) and have the same characters (one per client).

Like Section 3, your program must handle at least 3 simultaneous clients, each of which should run in their own separate process (*i.e.*, not have shared memory), and the clients should be able to be started at any time (*e.g.*, we should be able to start client 2 after 1 second or 30 minutes if we so choose).

Section 5: Asynchronicity! (25 points, 581 required/481 optional)

Your task for this part of the homework is to make your server design fully asynchronous, to support an arbitrary number of clients (up to five) running at different speeds (*i.e.*, we should be able to leverage the functionality from Part 2 of the assignment to increase or decrease the speed of a client's main game loop (thereby halving or doubling the rate of 0MQ messages) and the right thing should still happen. In other words, slowing down one client should not make all other clients slow down as well. Similarly, increasing the speed of a client should not make other clients run faster as well. 0MQ does provide semantics that will give you this for free (*i.e.*, the `ROUTER` or `DEALER`), but ***you may not use them for this assignment***. You must re-create these semantics using multiple threads with synchronous communications from server thread to client process.

To accomplish this, you'll need to pay attention to two things:

1. Since moving platforms should be governed by the server and therefore be in the same place on all clients, you will probably want to tie your character object to a timeline rather than the moving platform.

2. You may want to have threads on the server dedicated to reading/writing to clients running at slower rates.

Writeup

Now that you have designed and implemented a number of engine components, write a 1–2 page paper summarizing your design. That is a minimum of 1 FULL single-spaced page. Think creatively about what you have done. Why did you make the design decisions you made? What did they enable for this assignment? What will they enable in the future? What didn't work out the way you had hoped, and why? The most successful writeups will contain evidence that you have thought deeply about these decisions and implementations and what they can produce and have gone beyond what is written in the assignment.

As an appendix to your paper, please include all relevant screenshots to support your discussion. The appendix does not count toward your 1–2 page requirement. Your points for the writeup will represent 1/2 of the points allocation for each of the above sections (*i.e.*, 20 of the 40 points for the "Putting it all Together" Section will come from your writeup).

What to Submit

By the start of class on 09/28/22, please upload to Moodle a .zip file containing your code (in multiple subdirectories if you have different versions for different Sections of the assignment), a README file with compilation and execution instructions detailing which files or directories pertain to which portions of the assignment, and a pdf of your writeup.