

Server Structure:

The main decision I made for the server was how to handle both sending the audio information whilst still tending to the changing values of lambda. To adapt to this, I used select, but with only one file descriptor. I was either ready to read from, or not every time select was ran. The important change from the last lab pertaining to select is that I had to make the time out 0 time, otherwise it would arbitrarily wait until the client sent it a new value, and it would never send bytes of the audio file. So if a fd was ready, I knew there needed to be an update in the lambda value, but if there was no files ready when polling with select, just move on and keep sending the blocks of the file.

Client Structure:

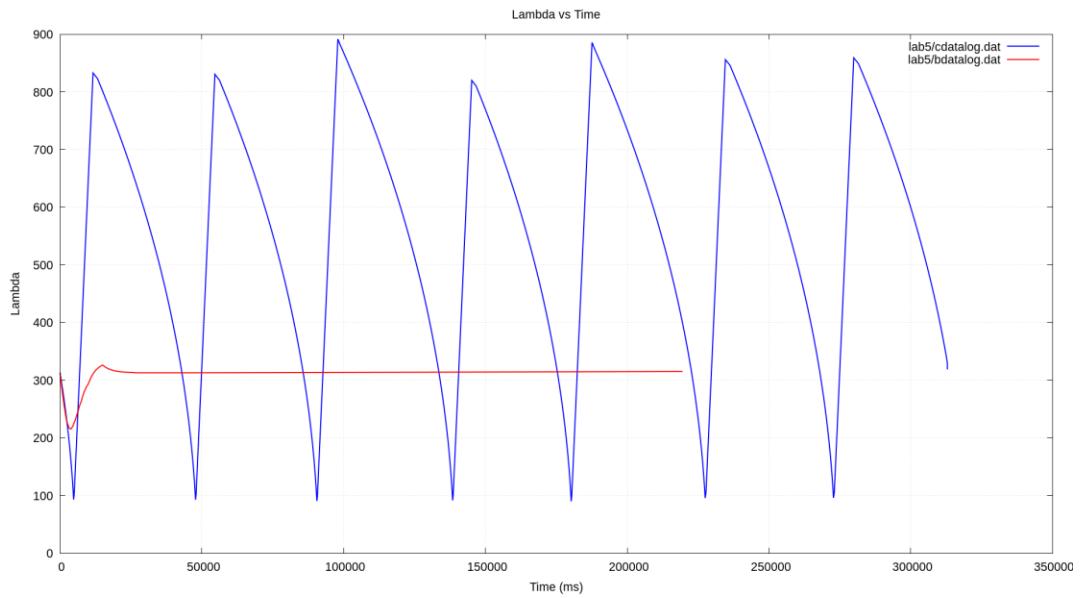
I made two major decisions for the client. To run the play and adjacent functions in the muaudio.c file, I ran that as a separate thread, so it could continue on playing the bytes while the main thread was handing the receiving from server, calculating lambda, and sending values back to the server.

With that, another two decisions I made were about communication between the two threads. The child thread was given two pieces of shared memory, one that was the buffer, where bytes of the audio file were to be written to and read from. However, there is no way to query the buffer about how many bytes were remaining in the buffer, so the second piece of shared memory was a simple int in a mutex, that the two threads would lock, adjust, and unlock as needed. This was necessary for the parent's thread to find $Q(t)$.

In many ways, I made these decision due to comfort, since I am more familiar with them than other methods, say signal handling, but I also found them to be more graceful at handling shared memory.

Bonus

To do this, I just accepted bytes without acking until it was adequately full, and then never update invlambda. Performance is great on these computers where there is almost no loss



Clearly, method D is preferable. Although it took a lot of adjusting parameters until the convergence was realistic, At least this method did converge, which cannot be said for method b. Even after adjusting the parameters, I could barely find something that didn't go negative.