STACK4 and the D Programming Language (Part2)

This is the second installment in a series of articles. This series describes the server backend that I wrote for the mobile game app STACK4[1].

In the first post I roughly elaborated the different options of languages+frameworks that were evaluated before deciding to use D.

This post describes in more detail what kind of requirements the server infrastructure imposed and what kind of auxiliary libraries were open sourced in the process of developing the STACK4 server backend.

The STACK4 Client uses Unity3D[2] as a cross-platform game engine. It makes it easy to develop for all the major mobile systems like Android, iOS and Windows-Phone with a (almost) unified code base.

Long Polling

Unity Basic (Free license) restricts you from using raw sockets. So you can choose to either implement your own native extension for all platforms or to resort to using the web requests that they allow. I chose to use this approach since it is the most proxy/firewall friendly approach and since I do not depend on low latency it is sufficient for me. I knew libraries like socket.io and sockjs from my javascript/typescript adventures so I decided to implement the sockjs protocol on top of the web requests function (WWW class[3]) in Unity3D. These libraries give me the advantage of using an interface that acts like regular persistent connection socket but can fall back to a long polling method if needed. In my case only this method is needed. This work is available on github: https://github.com/Extrawurst/sockjs-unity3d-xhr

Asynchronous I/O, Fibers and SockJs

I used nodejs a lot the last 2 years, even at work. Using Socket.IO I implemented a realtime multiplayer game in a toy project called beachwars-online (hiding the whole browser inconsistency hell from me). Socket.IO supports long polling as a fallback too, but unfortunately the Socket.IO implementation for D[4] did not support that fallback to long polling yet. Since I already decided to use a similar but less feature rich lib called SockJs for the client side I implemented SockJs for the server-side in D too: https://github.com/Extrawurst/sockjs-d.

Through my work with nodejs I was already sold to the concept of asynchronous I/O and the performance of it. As much as I hated coding in JS all the servers I wrote in node where exceptionally fast. Good thing Sönke Ludwig wrote vibe.d[5] which offers exactly that for D. It is based on libevent and offers a lot of functionality similar to nodejs making web development and network programming in general very convenient. That is why the sockjs server lib is based on vibe.d.

The major difference in using vibe.d compared to nodejs (other than the fact that it is D code) is the fact that vibe.d chose to use an approach different to cascading callbacks inside of callbacks to implement async events. vibe.d uses a co-routine-like approach (in d-land called Fibers[6]) which basically leaves the user code synchronous and linearly. As a bonus or more a natural consequence of that is that exceptions and debugging in general still have intact callstacks (sounds small but believe me debugging nodejs in comparison sucks big time).

Dependency Libraries

Here is a simple list of dub[7] libs that I used in the project. They all integrate nicely and use the asynchronous approach if needed. The ones that I developed during this project are all published on github[8] and are going to be subject of the following posts:

- sockjs-d
  SockJs server implementation based on vibe.d (see above)
- sockjs-unity3d-xhr
  The unity3d compatible implementation of the SockJs Protocol (see above)
- mysql-native
  The mysql connector used (vibe.d/async compatible)
- gcm-d
  Google Cloud Messaging connector (async)
- apn-d
  Apple Push Notification Service connector (work in progress)
- xtea-d
  eXtended Tiny Encryption Algorithm implemented in native D
- elo-rating-d
  Simple lib implementing the elo rating system in native D
- forever-d
  Tool based on node's forever to keep a process running continuously

Since I was asked to show some actual code, I now did in Part 3…

1. https://play.google.com/store/apps/details?id=com.Extrawurst.FIR [↩]
2. http://unity3d.com/ [↩]
3. http://docs.unity3d.com/Documentation/ScriptReference/WWW.html [↩]
4. http://code.dlang.org/packages/socket.io [↩]
5. http://vibed.org/features#performance [↩]
6. http://vibed.org/features#fibers [↩]
7. http://code.dlang.org/ [↩]
8. https://github.com/Extrawurst [↩]