Ram Vakada

Prof. Alexey Nikolaev – Functional Programming in OCaml

Project Report

May 9th, 2019

Program Name: OCamlAlert

OCamlAlert is a TCP based alert system. It's an alert system like CUNY's alert system. CUNY's alert system works in such a way that all registered users are sent a text message informing them about emergencies like the school being closed down due to severe weather of if there is an active shooter and even fires. In the case of OCamlAlert, any user that is connected to the server will receive an alert that an administrator sends out. It uses Lwt, plain TCP sockets from the Unix library and consists of the following three parts:

1. The Server
2. The User Client
3. The Admin Client

The User Client & Admin Client are the simpler parts of the program and implement sockets from the Unix library. The User Client connects to the server on port 8484 and identifies itself as an alert receiver by sending the handshake message "iAmClient." Then, it keeps the connection alive by pinging a "KEEP-ALIVE" message every five seconds and listening for alerts sent by the server. If it receives an alert, it simply prints it out onto the console. The Admin Client is even simpler in the sense that it has a runtime of almost 0 seconds. It takes the administrator password as the first command line argument and the rest of the command-line

arguments are concatenated into the alert string. It connects to the server and first sends “iAmAdmin” to identify itself as an admin client, then the passwords, and finally the alert. The server will then send a response which lets the admin client know whether the password was correct and if the alert was sent out. The execution stops there and it needs to be re-run every time a new alert needs to be sent out.

The server is the meat of the entire program and implements Lwt, an OCaml concurrent programming library that allows us to use the “future-promise” model to handle requests. The server on initialization will open a server socket on port using Lwt’s Unix wrapper on 8484. Upon any connection request, the connection is immediately passed onto the handshakeHandler and Lwt continues to accept requests on the main thread. The handshakeHandler then goes through the handshake process with the client and routes the connection to the clientHandler or the adminHandler appropriately.

I chose this project because I wanted to do something that was related to networks and was under the impression that it wouldn’t be too difficult to implement. I was wrong, Lwt is dense and although it has good documentation, there is a lack of examples on the internet. I learnt a lot, especially about functional programming and it’s uses but at the same time, this was one of the most frustrating software I’ve built. I’ve spent hours on end trying to fix errors that I would have never even encountered have I used an Object-Oriented imperative language. Even apart from that, I’ve run into roadblocks and obstacles that took more time than they should have taken to solve/overcome, mainly because OCaml isn’t really the language to be used to develop this sort of software and there aren’t any resources

available online that can really help you. What made it even harder was that Lwt's Infix operators like >>=, >|>, >> make it impossible to search code on google.

To sum it up:

1. Difficulties Encountered: Lwt.
2. Things Learned: Don't use Lwt. Don't build such apps on OCaml.
3. Things I would have done differently: Choose a less imperative-logic demanding project.