Datarace

Operating systems and multicore programming (1DT089)

Project proposal for group 17

Falk Nilsson, Max (910316-2518)
Jaksic, Marina (910411-0920)
Reeves, Max (860107-0033)
Sandberg, Joel (870326-1456)
Toghiani-Rizi, Babak (891109-6371)

Version 2 May 7, 2014

1 Introduction

We want to create an app that allows two (or more) users to challenge each other to run a given distance on time, the fastest one will be crowned the winner. The users can challenge each other without actually being in the same place and time, and still be able to compare the time and the winner will be announced. We thought it would be a fun project to do, since it allows us to develop something in a different way compared to what we've done before.

Hopefully, this project will teach us about: setting up a server, communication between the server and the app and developing an app.

The main challenge will probably be to do all the calculations in the server and then connect it with the app.

The concept of concurrency in this project is to let two competitors challenge each other at the same time, and the server does all the calculations and send the data to all competitors in real time. In that way the competitors will be able to see who is in the lead.

2 System architecture

The client will communicate with the server and the server will issue requests to the database which will save the users' data. The server will hold the logic for how to interact with the database while the client will act as a user interface where no real calculations will be done.

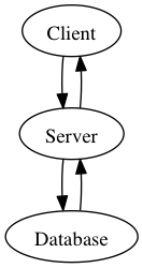

Figure 1: Highlevel model of how the different systems interact

The figure below shows how the different components in the system will interact with each other and work together to achieve different tasks required by the user. The UI will act as a dumb client and send requests to the server. When a new client connects to the server it will enter at the new node and there it will get it's own process in the form of a client manager, which will be the the center for all the communication for that client. The client manager will have the most relevant data in memory (Internal state node). The client manager will then use different sub procedures to complete different tasks. GPS Calc will handle the GPS coordinates received from the client and calculate distances and other GPS related tasks. DB Logic will be an interface for the server to the database. Match Logic will hold logic for how the ongoing challenge against the competitor goes.

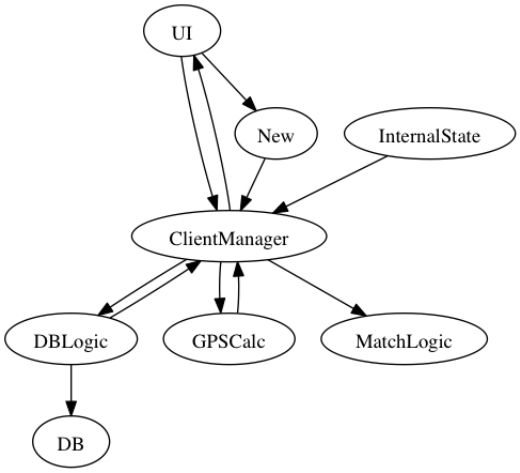

Figure 2: Model of how the different subsystems interact

The figure below describes the process architecture of the server. Here the MasterSupervisor process is in charge of monitoring the behaviour of the two supervisor processes NewSupervisor and ClientMasterSupervisor. ClientMasterSupervisor acts as the main supervisor for the ClientServer processes, with the intermediary ClientSupervisors controlling the supervision for each individual ClientServer process.

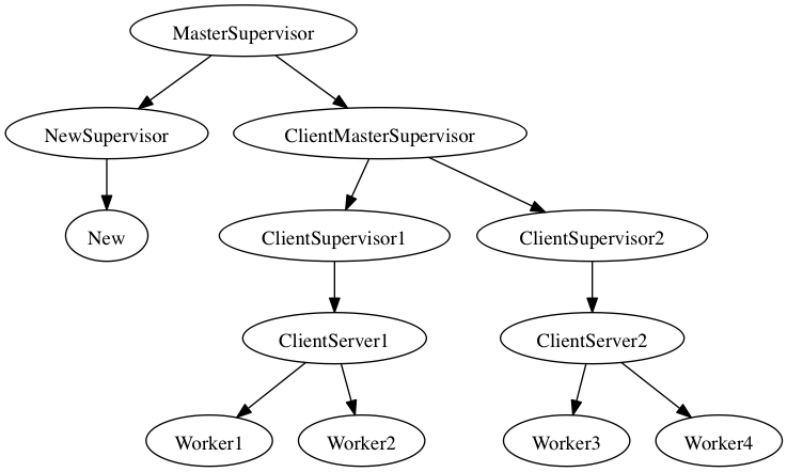

Figure 3: A picture of server side architecture

3 Concurrency models

In this project we will use Erlang as the back-end server language and thus we will be using the Actor model as concurrency model. We chose Erlang because of its simplicity in developing concurrent software, which comes from the inherently concurrent nature of the language and great tools available for creating and supervising processes.

We also want to increase our knowledge about the Erlang OTP design principles and framework, in order to get a better understanding of what tools are used and how Erlang applications are developed professionally.

For the app we will use Objective-C, the native language in iOS. The app in itself is not concurrent since it is just the front-end, but it will use the servers concurrency, and that makes it a graphic user interface.

4 Development tools

For Erlang we will be using Emacs. It's a great text editor, since everyone has been using Emacs a lot we don't have to learn to use it. The app will be developed using Xcode, which is Apple's tool for developing apps in iOS. All our code will be stored and distributed amongst all members of the group in a repository on GitHub. We will use Make and eMake to compile our Erlang code and Xcode for compiling the app. The testing part is done with eUnit for the Erlang server, because it is the simplest way to test Erlang code. Xcode has it's own unit testing system that we will be using to make sure our Objective-C code works.

By using Edoc and, in Xcode, Doxygen, we will document all our source code. Edoc is great because it generates a html page of the documentation automaticly from the comments in the source code. Xcode has built in support for Doxygen as the documentation method. Doxygen generates, in the same way as Edoc, a html page that contains the code's comments as documentation.

5 Process evaluation

We started discussing ideas surrounding the project in the beginning of the course. When it was time to decide for a project, we already had a clue. We had an idea of creating an app that communicates with a server, so we sat down and brainstormed until we had a pretty clear vision on how we wanted it to look, act and work. We got the idea of a running challenge app and everyone in the group thought it seemed fun, as it was something we could use in our daily lives. The actual brainstorming process went very well, since we had an idea that evolved along with the course and through input from every member of the group.