Language Abstractions for Concurrent and Parallel Programming (1DL540)

Final Project: P2PChat

Lucas Arnström

December 9, 2016

Contents

1 Introduction

2 Compilation & Usage

2.1 Compilation

In order to compile the program you will need the following installed on your system:

- Rust: the programming language.
- Cargo: the package manager for the Rust language.

Compile the program by issuing the following command: "`cargo build --release`". This will put a binary under the path "`target/release`" called "`p2pchat`".

2.2 Usage

Issuing the command "`p2pchat -h`" displays the following output:

```
Usage:
    p2pchat [OPTIONS] USERNAME PORT

P2P Chat system built in Rust as the final project for the LACPP-course.

positional arguments:
  username              Username to use for the chat.
  port                  Local port for incoming connections.

optional arguments:
  -h,--help             show this help message and exit
  -v,--verbose          Output lots of info.
  -r,--remote REMOTE    Define remote hosts.
  --no-client           Disables the client part of the program.
                        It will not connect to remote hosts.
```

The program supports three commands when started, namely:

1. "`connect`": connect to a remote client. Example: "`connect 127.0.0.1:1234`"
2. "`say`": broadcast a message over the network. Example "`say Hi there!`"
3. "`quit`": terminates the program.

3 Program Documentation

When the program is first executed it creates two new actors. One actor that listens for new tcp-connections called the "ConnectionListener", and one actor that keeps track of all connected clients and their associated actors. The tracking actor is called the "ActorManager". The "ActorManager" also is the one responsible for propagating messages between all active connections. When it receives a new message it broadcasts it to all other actors, which in turn send the message to their connected remote clients.

When a new connection is established the "ConnectionListener" will spawn two new actors. One listens for new data on the socket, and the other waits for data to send back over the socket. It then notifies the "ActorManager" about these two new actors and the connection they represent by sending it a message.