Assignment 4. FaceIn

10.05.14

Radu Ionita, Nicolae Mariuta

We worked again on the assignemnt after we understood Erlang a lot better and managed to finish it.

We have done the following:

- created function *start(N)* that starts process for person with name N. We used tuples to return message as requested in the assignment text. The function starts the function *loop* having parameters: Name of the user, lists of type *dict* with friends and received messages which are initially empty.
- for communication we did not create the function *rcp* as taught at the course and we followed the advice of Olex to create 2 functions: *asyn* for asynchronous communication which just sends the message without waiting a response and function *sync* which sends message and waits for an answer
- the required functions(*add_friend*, *friends*, *broadcast*, *received_messages*) which send requests to the server (using *sync* function).
- the *loop* function for handling the requests
- a separate file for creating the graph in the documentation and testing the results

Inside the *loop* we handled the following requests:

- *add_friend* which sends an *sync* message to the process ID for the friend we want to add. It also processes the results, in case the person returns message with the name , it will be added to list of friends, otherwise show error
- *nameRequest* to handle a friend request; is returning a message containing the name of the person using a process ID
- *friendsList* that use the *dict* functions to display the list of friends as pairs [pid,name]
- *broadcast* for starting making a broadcast and *broadcastMessage* for sending a message further. The algorithm is like this: the person adds the message at list of messages together with name of sending person in case the, then checks if contor R and if it is not 0, the message is sent to all friends in list of friends of that person. Maybe it is not the most optimal way to do

it and perhaps we could have only one request handling to do the broadcasting but this is how we found to make it work easier.

- *received_messages* to display the list of received messages for the person, it is done almost the same way as the *friendsList*

As for the testing we created a file with 2 test functions; in each of them it is created the graph with connections from the assignment and then:

- for *test1* it is displayed the list of friends for persons in graph
- for *test2* broadcast messages are sent and then it displays the received messages for different persons
- for each test we created several *friends* and *received_messages* and only the last one is displayed in the console when running the function. To view a specific one, we just move it to the end with list of requests.
- we tried to use some unitesting using the Erlang libraries but we did not manage to understand how it works.

As a conclusion, our code works properly and answers most of the requirements in the assignment. Maybe we could improve the testing and take care of situation in which deadlock may appear, but it was hard for us to figure when deadlock may appear using only one PC.