Individual Haskell Project

Soham Trivikram Naik Khawte - 230965856

Summary:

The project is a simulation of a social network system using Haskell. It models users and messages in a chat system, where users can send messages to each other. The goal of this project is to implement a Haskell stack app that uses threads and concurrent computation and simulates a social network. The main program represents ten "user" threads, and each of these threads model a user in the social network. The users then (at random intervals) choose one of the other users (at random) and send a random message to that user.

Design Decisions

User and Message Types
The User type is modelled with a username and a list of receivedMessages.
the Message type includes fromUser and content. This design allows each user to have a unique identity and keep track of their received messages. The Messagetype includes the sender's username and the content of the message, providing all the necessary information for a basic chat message.

Concurrency
The forkIO function is used to create a new thread for each user's activity. This allows all users to send messages concurrently.

File Handling
All the chat history is being saved in a text file at the end of the execution.

Randomness
The randomRIO function is used to introduce randomness into the program, such as the delay between messages and the selection of the message recipient and content.

MVars
Due to the synchronising nature of MVars which is very helpful in concurrent threads following MVars were created:

globalCounter: It is used to keep track of the total number of messages sent across all users. It ensures that the count is updated safely even when multiple threads are trying to update it concurrently.

receivedMessages: It keeps track of all the messages that a user has received.

Issues Faced

Cyclic dependency

While working on the project initially 2 modules are created namely User and Message. User.hs was importing Message.hs, and Message.hs was also importing User.hs, creating a direct cyclic dependency. Thus to deal with this another module was created Acitivity.hs and both User.hs and Message.hs were imported into this new module, breaking the cycle and resolving the issue.

Complexity

The ‘userActivity’ function sends a message from one user to another with a delay to simulate the time it takes for a user to read and respond to a message.

At the end of execution, a text file named ‘chats.txt’ is generated which saves the chat history of all the users.

Extension

BangPatterns

By using the BangPatterns extension, we can specify that an argument should be evaluated immediately when the function is called, rather than when the argument is used. This is done by prefixing the argument with a bang (!).

In the provided code, BangPatterns is used in the sendMessage and userActivity functions are evaluated immediately, which could potentially avoid miscount of messages and make the program’s behaviour more predictable.