Haskell Independent Project Report

Koustubh Saxena, 230842410

Summary

The goal is to simulate a small social network. Where people user post stuff and can read each other posts. We do this by creating 10 users, each having two `MVars`. One tracking the number of messages they received (`messagesReceived`) , while the other acting as the inbox(`inbox`). We then create 10 users of this type. Each user is given A random username by randomly selecting three alphabets. After which, we create 10 threads, One for each user.

Inside their threads, each user first sends a message to a randomly selected user. We error check it. If successful. The thread switches to receive mode and waits for someone to send it a message. This happens 10 times per thread.

Steps

User Representation:

Each user is represented by the User data structure, which includes the username, the number of messages received (messagesReceived), and the inbox (inbox). The username is a random combination of three alphabets.

User Creation::

The `createUser` function is responsible for creating a user. It initializes the `MVar` for tracking message count and an empty `MVar` for the inbox. A thread is then forked for each user to run asynchronously.

User Threads:

In the thread, a user attempts to send a message to a randomly selected recipient (another user). If successful, the user switches to receive mode and waits for someone to send them a message. This process is repeated 10 times for each user

Messaging and Receiving

The `sendMessage` function attempts to send a message from one user to another. If successful, the recipient's inbox is notified.

The `receiveMode` function is invoked when a user is unable to send any messages. In this mode, the user waits for messages from others.

Timed Receive Mode:

To avoid indefinite waiting during the receiving mode, a timeout mechanism is introduced using the timedReceive function. The timeout function limits the waiting time for a message to 3 seconds. If the user receives a message within this time, a success message is printed; otherwise, a timeout message is displayed

Issue Faced

Thread Sleeping

Haskell automatically sleeps threads that try to write into an occupied MVars, it sleeps the thread until further activity. This makes it difficult to execute the program further as some threads are stuck at sending messages to some user who is also stuck at sending message. Deadlocking those threads, while most of the other are stuck at receiving messages even when there is no one to send messages too.

To address this issue, a timeout mechanism was introduced in the receiving mode. Threads waiting for messages for an extended period exit the receiving mode and move on to the next replicate step. This prevents threads from being indefinitely stuck in the receiving mode and allows the program to make progress, avoiding deadlocks during the receiving phase.

Additionally, to avoid deadlocking threads during message sending, the `tryMVar` function was employed instead of `putMVar`. `tryMVar` attempts to put a value into an `MVar` without blocking the thread. If the `MVar` is occupied, it returns immediately without waiting. While this approach may result in skipping some messages, it ensures that threads can continue execution and prevents the program from coming to a halt due to occupied MVars.