Issues

Understanding which MVar to use to communicate between the threads and to the main thread proved to be one of the biggest challenges I encountered while working on this project. Initially, I used a single MVar for all threads, which resulted in only one message being sent because the main thread would immediately take from the MVar after it, killing all other threads. As a result, I decided to introduce another MVar to facilitate thread communication.

This second MVar presented me with problems as well. When attempting to update this list for each message sent, I ran into issues with the idea I had for storing the list of sent messages in this MVar. I rejected this notion and opted to store the messages in a database instead.

Finding the ideal moment to delay each thread for their unpredictable waiting period was my last problem. I discovered that my initial range of 0.5 to 2 seconds wasn't working. I significantly shortened the range to 500 to 2000 microseconds, which dramatically increased the program's effectiveness.

Design Decisions

Modelling Types

I modelled the User datatype with a `userID`, `firstName` and `lastName`. The first and last name make the username and the ID is used to uniquely identify each user.

Similarly, I modelled the Message datatype with a `msgID`, `msgContent`, `userFrom` and `userTo`. The msgID uniquely identified each message, the msgContent stores the content of the message and the final two store the sending and receiving user IDs.

Choosing MVars

I decided to include 2 MVars in my program. Once 100 messages have been sent, the first variable,'msgsSent,' which was initially created as an empty MVar, is filled by one of the 10 user threads. It just includes the integer 100. The main thread stops all other threads and stops sending new messages by waiting until this MVar is full before taking from it.

The second MVar, "msgBox," is utilised for inter-thread communication. It has a placeholder message when it is first defined so that the first thread won't have to draw from an empty MVar. It makes sure that only one thread sends messages at a time, allowing each thread to verify if 100 messages have been sent and receive an accurate response without other threads sending messages in the interim. To signify the completion of its process, each thread places the message it just sent back into this MVar.

Extra Work

I built a database connection to my programme to support the extra functionality. I made a single table to house all of the messages being sent, and each thread accesses it to see if there are 100 messages there. The procedure was greatly facilitated by this feature, which assisted in message tracking. Additionally, it helped with the need to display the messages that each user has received.

I included database queries as a second extra functionality. Some, like the one I just stated, were necessary, and I also added more queries to show how many messages each user sent and how often each message was sent.