COMP28112 Exercise 1 Report

Oliver Clements

March 2023

Contents

1 Exercise 1 - Messaging Protocol

1.1 Description of Protocol

The code I wrote implements a simple messaging protocol between two users and uses a client-server architecture. The server is used to set up a shared data store that the clients can use to exchange messages, this uses the dictionary data type. Each client key has a value which stores if it's in the send or receive state, there's a key which stores the most recent message as its value and a key which stores which clients turn it is as its value. When a client connects to the server it first checks to see if the server is currently storing a previously sent message in the shared data store. If there is no previous message stored, then that client sets its status to 'send', showing that it is ready to send a message when the user gives it one. If there is already a previous message in the shared data store, the client sets its status to 'receive' and waits for the other client to send a message before attempting to send a message itself. When a client sends a message it updates the server's shared data store with the message it just sent and changes that client's status to 'receive' showing it is ready for the other client to send it a message. The client also sets the value of the 'Turn' key in the dictionary to the other client's username, this is done to indicate that it is the other client's turn to send a message next. When the other client sends a message, it updates the server's shared data store with its response and changes its status to 'send', it also sets the value of the 'Turn' key to the other client's username indicating it is their turn to send a message next. The conversation continues back and forth, following this protocol, until either of the clients enters the 'exit' keyword which triggers code that clears the server's data and terminates the program. The program uses a combination of time.sleep(1) and recursive function calls to make sure that the server is being constantly polled by each client to see

if it's its turn to send a message now. A client is unable to send a message until it knows it's its turn to send, this prevents system deadlocking as both clients will never attempt to send a message at the same time. This protocol is relatively simple and relies heavily on the two clients following the expected behaviour to keep the conversation going. It also doesn't have any security features or authentication measures at this stage unfortunately, however due to the simplicity of the program the only data that could be taken at any point is the most recent message as that is all that is stored in the server.

1.2 Description of How to Run the Code

To run the code the python file needs to be run in two different terminals separately, this can be done using the following command: python3 imclient-e11226oc.py. For the program to work properly the program needs to be run straight after each other before any inputting is done, then the username of both clients needs to be input and only then can the first message can be sent. Once this happens the terminals are interacted with by following the prompts, the message from one client is printed to the terminal of the other client so that they can actually see the message sent and respond appropriately. This exchange continues back and forth until either client chooses to type 'exit' into the terminal, which causes the server data to be cleared and the program to terminate.