Distributed Systems
Exercise 1: A Simple Messaging System for Healthcare Professionals

Horia Radu

Task 1.1:

https://web.cs.manchester.ac.uk/k55592hr/comp28112_ex1/IMserver.php

Empty in the beginning, “a:1:{s:6:"nurse1";s:5:"hello";}” after setting a key and “a:0:{}” after unsetting the key

https://web.cs.manchester.ac.uk/k55592hr/comp28112_ex1/IMserver.php?action=set&key=nurse1&value=hello

```
a:1:{s:6:"nurse1";s:5:"hello";}
```

https://web.cs.manchester.ac.uk/k55592hr/comp28112_ex1/IMserver.php?action=get&key=nurse1

```
hello
```

https://web.cs.manchester.ac.uk/k55592hr/comp28112_ex1/IMserver.php?action=unset&key=nurse1

```
a:0:{}
```

Task 1.2:

```
import im
server=im.IMServerProxy('https://web.cs.manchester.ac.uk/k55592hr/comp28112_ex1/IMserver.php')
```

```
server['pythonTest1'] = 'hello'
a:1:{s:11:"pythonTest1";s:5:"hello";}
```

```
print(server['pythonTest1'])
Output: b'hello\n'
```

```
del server['pythonTest1']
a:0:{}
```

https://web.cs.manchester.ac.uk/k55592hr/comp28112_ex1/IMserver.php?action=set&key=test&value=testhello

```
a:1:{s:4:"test";s:9:"testhello";}

print(server['test'])
Output: b'testhello\n'

del server['test']
a:0:{}
```

- Describe your proposed protocol/strategy using text, explaining your rationale behind each decision. (5 marks)

First of all, my code is made inside a while True loop, this makes sure that the client will continue on running until stopped.

Secondly, I decided on using a thread, so that at a given interval of time, with time.sleep(seconds), that client will recheck whether there is any message received or sent, and proceed accordingly.

This thread has a number of if statements that will decide what to do each time, based on a number of variables, situations and inputs, similar to states in a FSM (Finite State Machine).

To completely make this work, I have decided on including an item in the server with the key “__waiting__”. After someone sends a message the item with this key will contain the name of the sender of the message, who will now be changed to a state of waiting for a reply. When the other person responds, they will replace the other person in the “__waiting__” item, and so on. This helps with preventing deadlocking, as there can not be 2 persons at the same time in the waiting state, and also makes it possible so that when both the users connect to the client, any one of them can be the one that start the conversation, and after one of them sends a message, then they will go in a state of sending-waiting and vice-versa.

- Give one example of a limitation of this system architecture, and one example of one way in which this limitation could be overcome, towards the realisation of a more realistic messaging system. (2 marks)

A limitation of this system is that we do not have a clear database that can hold information and variables better, and more organized.

Another one is that we have to rely a lot on local variables, rather than having most of the job done in the “back-end” of the server, which would be a lot safer and faster.

Something else that could be improved in the client, is making it in such a way that it doesn’t check every few seconds to see if there was an update, but instead when a user sends a message, it also sends a trigger to the receiver that makes him refresh the messages and show him the new message, and tells him he can reply. This would minimise the hardware usage, but also make the messaging faster and efficient.

A system architecture would better work and overcome a lot of problems by being well structured and organized. By this I mean separating each task to something, having a database to store data, a server, and an messaging application that gets through to the client where the user will have access.