CS 2510: Computer Operating Systems - Assignment 1

Name: Haneef Ahamed Mohammad(ham149)

Date of Submission: 2/17/2022

Introduction:

In this Assignment, we will implement a simple client-server "group chat" system with n clients/configuration. With Alice, Bob and Chad (the three client processes).

Requirements:

1. The clients should be able to send/receive messages from the server.
2. The server keeps track of the clients that has received the messages and only sends the unread messages.

Test Cases:

1. Alice sends a single message. Chad and Bob come online after a 5 second delay, and receives all messages from Alice. (Log the message in the console)
2. Alice, Bob, and Chad are online. Bob sends a message to all; Chad and Alice receive the message (The sender Bob doesn't receive the message from the server). Alice sends a message to all; Bob and Chad receive it (but not Alice). Doug, not part of the group, joins the server but receives no message.

Implementation:

For the implementation of the asynchronous RPC, we will be using and the gRPC package in python. In gRPC, a client application can directly call a method on a server application on a different machine as if it were a local object, making it easier for you to create distributed applications and services. As in many RPC systems, gRPC is based around the idea of defining a service, specifying the methods that can be called remotely with their parameters and return types. On the server side, the server implements this interface and runs a gRPC server to handle client calls. On the client side, the client has a stub (referred to as just a client in some languages) that provides the same methods as the server.

Based on the gRPC definition we need a .proto file, server and a client to create our group chat.

Proto file: Proto is an ordinary text file with a .proto extension, we can define our service, message type and much more.

In our chat.proto file, we defined the message type with respective fields and service as bi-directional as it best suits our requirement.

```
message Note {
    string name = 1;
    string message = 2;
}
```

Fig: Message Type

```
service ChatServer {
    // This bi-directional stream makes it possible to send and receive
Notes between 2 persons
    rpc ChatStream (Empty) returns (stream Note);
    rpc SendNote (Note) returns (Empty);
}
```

Fig: Service type.

Server: My server program can be broken down into two components.

1. **ChatStream():** This function is used to This is a response-stream type call. This means the server can keep sending messages. Every client opens this connection and waits for server to send new messages
2. **SendNote():** This function is used to This method is called when a clients sends a Note to the server.

These functions help us achieve some of the important features of server.

1. Running a gRPC server to listen for requests from clients and transmit responses.
2. The server keeps track of the clients that has received the messages and only sends the unread messages.
3. Create a chat group with specified client ids.

```
def ChatStream(self, request_iterator, context):
    lastindex = 0
# For every client a infinite loop starts (in gRPC's own managed thread)
while True:
    # Check if there are any new messages
    while len(self.chats) > lastindex:
        n = self.chats[lastindex]
        lastindex += 1
        if request_iterator.name != n.name and request_iterator.name in
self.groups and n.name in self.groups:
            yield n
```

Fig: ChatStream()

```
def SendNote(self, request: chat.Note, context):
    # this is only for the server console
    self.chats.append(request)
       return chat.Confirmation(value=True)
```

Fig: SendNote()

Client: My client program's primary feature or task is

1. Client should be able to send/receive messages from the server.

This is achieved with help of two functions.

1. **listen_messages ():** This method will be ran in a separate thread as the main/ui thread, because the for-in call is blocking when waiting for new messages.
2. **send_message():** This method is called when user enters something into the textbox

```
def __listen_for_messages(self)
    for note in self.conn.ChatStream(chat.User(name=self.username)):
```

```
        print("Recieved from Server {}: [{}] {}".format(self.username,
note.name, note.message))
```

Fig: listen_for_message()

```
def send_message(self, event):
    while True:
    message = input()
    if message != '':
        yield self.make_messages(message)
```

Fig: send_message()

Dockerfile:

```
FROM python:3.8-slim-buster

WORKDIR C:/Users/hanee/Downloads/docker/cs2510
COPY requirements.txt requirements.txt

RUN pip3 install -r requirements.txt

COPY . .

RUN chmod a+x run_test.sh
CMD ["./run_test.sh"]
EXPOSE 3000
```

To use this docker image use the following commands.

docker build --tag <name> .

docker run <name>

Output:

References:

1. https://grpc.io/docs/languages/python/basics/