

Lab 1 – Chat Application

Rafat Khan <rafatk@kth.se>

March 1, 2022

1 Introduction

The distributed system with basic functionalities provided for this assignment consists of a client and a chat server. Clients and servers are both Jini applications, so to find each other, we have to use a Jini infrastructure. To complete this assignment, several tasks need to be performed. There is a choice of alternative challenges which need to be implemented within the existing system.

2 The Assignment

2.1 Selected Tasks

Within the given list of tasks, following was selected:

Task 1. Enable all clients connected to the service to see whenever a client joins or leaves the chat. Add a user command to the client that allows the user to list users currently joined to the service. If the client uses the .name command, this should be seen by all connected clients (2).

Task 2. Make the user information useful to the maintainer of the chat service, so that the chat server console prints when someone connects and disconnects. In the case of a disconnect, also print session time (time between connect and disconnect), and how many messages or Kbytes that user has sent (2a).

Task 3. Add an AFK (Away From Keyboard) detector that automatically notifies the other clients when someone has been inactive for a certain period of time. The message should be automatically generated when the timeout period expires. Add a command in the client for turning on and off the display of such automated messages (6).

2.2 Implementation

2.2.1 Task 1

On the server end, data structures will be used to handle information about which user logged into the chat client, as well as a list of active users. On receiving the prompt, the server will store the list of users and return the active ones.

```
– On Client
public void removeUserName(ChatServerInterface server)
    throws RemoteException {
    if (server != null) {
        String removeData = myName;
        heartBeat(awayTime);
        server.removeDummyUser(removeData);
    }
}
```

```

- On Server
    public void removeDummyUser(String name) {
        InName = name + " is offline now";
        usernames.remove(name);
        System.out.println(name+" disconnected from the chat");
        displayActive();
    }

```

2.2.2 Task 2

User messages are inserted into the HashMap as they are sent in key-value pairs. The number of bits stored and the size of the total number of messages sent can be determined through this. From the key values, we return how many bits are stored for each user.

```

protected void removeClient(RemoteEventListener rel) {
    Calendar savedTime;
    synchronized (clients) {
        savedTime = userLoggedInfo.get(rel);
        clients.remove(rel);
        userLoggedInfo.remove(rel);

        System.out.println("Removed client : " +
            rel.toString());
        System.out.println("Session Info ——LoggedIn time:
"+savedTime.getTime()+" Loggedout Time——
"+Calendar.getInstance().getTime());
        System.out.println("Session lasted "+
            TimeUnit.MILLISECONDS.toMinutes
            ((savedTime.getTimeInMillis()-Calendar.getInstance()
            .getTimeInMillis()))+" minutes");
        DecimalFormat _numberFormat= new DecimalFormat("#0.0000");
        float kbData = ((float)userBytes.get(rel)/1024);
        System.out.println("Total chat Kbytes/Bytes sent
by the user "+ _numberFormat.format(kbData)
        +"/"+userBytes.get(rel));
    }
}

```

2.2.3 Task 3

Keep track of which users are active/available on the chat client using the Heartbeat signal. The storage and sending of data takes place on the client side, and there is no data store on the server side.

A timer will be defined and when the timer expires and the user is idle, the thread execution starts and the keyboard detection displays are not displayed.

```

public void heartBeat(long number) {
    ScheduledExecutorService scheduler=
        Executors.newSingleThreadScheduledExecutor();
    Executors.newSingleThreadScheduledExecutor();
    Runnable task = new Runnable(){
        public void run() {
            try {
                myServer.AFK(myName);
                check.clear();
            }

```

```

                                } catch (RemoteException e) {
                                    e.printStackTrace();
                                }
                            }
    };
};

```

2.3 Interface

```

public void addActiveUsers( String name)
    throws java.rmi.RemoteException;

public void removeDummyUser(String name)
    throws java.rmi.RemoteException;

public List<String> sendActiveUsers()
    throws java.rmi.RemoteException;

public void swapUserName(String oldName, String newName)
    throws java.rmi.RemoteException;

public void AFKRequiredUsers(String name, final boolean
    status, final RemoteEventListener currentListner)
    throws java.rmi.RemoteException;

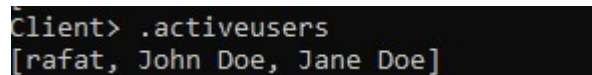
public void AFK(String name)
    throws java.rmi.RemoteException;

```

2.4 Result

2.4.1 Task1

The “. active” command accesses the collection that stores the usernames and it returns the list of active users from this list onto the client end.



```

Client> .activeusers
[rafat, John Doe, Jane Doe]

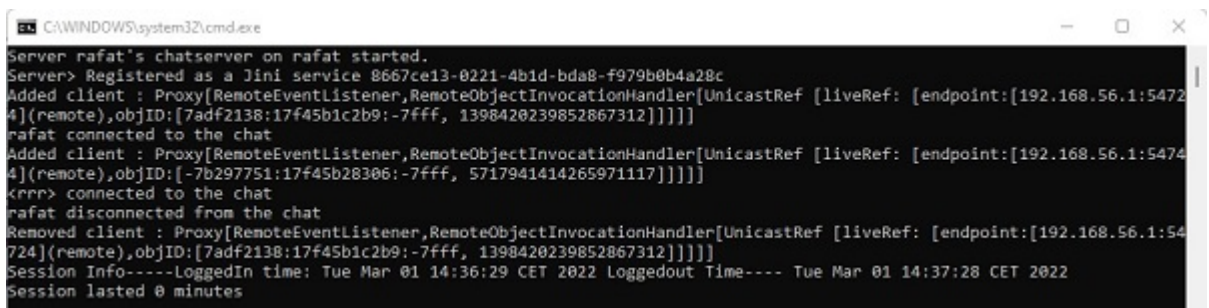
```

Figure 1: List of Active Users.

2.4.2 Task2

The session information is stored in a HashMap, which consists of the key value players. This can be displayed on the server end and the values stored comprise the login information.

The session information has also been configured to consist of the following – kilobytes of messages sent, logged in time, logged out time. The server is made more informative with this event log type of storage.



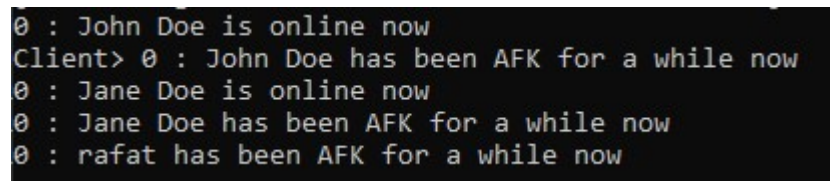
```
Server rafat's chatserver on rafat started.
Server> Registered as a Jini service 8667ce13-0221-4b1d-bda8-f979b0b4a28c
Added client : Proxy[RemoteEventListener,RemoteObjectInvocationHandler[UnicastRef [liveRef: [endpoint:[192.168.56.1:5472
4](remote),objID:[7adf2138:17f45b1c2b9:-7fff, 1398420239852867312]]]]]
rafat connected to the chat
Added client : Proxy[RemoteEventListener,RemoteObjectInvocationHandler[UnicastRef [liveRef: [endpoint:[192.168.56.1:5474
4](remote),objID:[-7b297751:17f45b28306:-7fff, 5717941414265971117]]]]]
krrr> connected to the chat
rafat disconnected from the chat
Removed client : Proxy[RemoteEventListener,RemoteObjectInvocationHandler[UnicastRef [liveRef: [endpoint:[192.168.56.1:54
724](remote),objID:[7adf2138:17f45b1c2b9:-7fff, 1398420239852867312]]]]]
Session Info-----LoggedIn time: Tue Mar 01 14:36:29 CET 2022 Loggedout Time---- Tue Mar 01 14:37:28 CET 2022
Session lasted 0 minutes
```

Figure 2: Server Log.

2.4.3 Task3

Heartbeat Signal- I have enabled the heartbeat signal on the client end in order to implement it. Each time the client sends a message, a counter is initiated on a separate thread, and when the timer expires, the method is invoked by the scheduler. The method notifies all clients that the user is AFK.

Away from Keyboard- By using the enable and disable AFK commands, I have implemented a toggle feature for the away from keyboard messages to enable and disable them based on the user's needs. The names of AFK users are taken from the list of names present on the server and we will use this to determine if the user called by enable or disable AFK is a match. AFK disabled users will not receive the message.



```
0 : John Doe is online now
Client> 0 : John Doe has been AFK for a while now
0 : Jane Doe is online now
0 : Jane Doe has been AFK for a while now
0 : rafat has been AFK for a while now
```

Figure 3: AFK notification on client end.

2.5 Conclusion

Overall, this assignment was challenging for me. I had fun learning how to use the codebase server and had a deeper understanding of distributed systems while solving the tasks.