CS 408 - Computer Networks - Fall 2021

Project – Switter, a Social Networking Application

This project is made of three steps; each has different deadlines specified below;

Project Step 1 Deadline: November 24, 2021, Wednesday, 22:00
Project Step 1 Demo: to be announced

Project Step 2 Deadline: December 10, 2021, Friday, 22:00
Project Step 2 Demo: to be announced

Project Step 3 Deadline: December 29, 2021, Wednesday, 22:00
Project Step 3 Demo: to be announced

Introduction

In this project, you are going to develop a social networking application called *Switter* (SU version of Twitter) by implementing client and server modules. (i) The *Server* module manages storage of messages (called *sweets*; SU version of tweets), sweet feeds, and follower relationships among the users, and (ii) the *Client* module acts as a user who posts sweets, follows and blocks other users, and view sweets of other users.

The server listens on a predefined port and accepts incoming client connections. There might be one or more clients connected to the server at the same time. Each client knows the IP address and the listening port of the server (to be entered through the Graphical User Interface (GUI)). Clients connect to the server on a corresponding port and identify themselves with their usernames. Server needs to keep the usernames of currently connected clients in order to avoid the same name to be connected more than once at a given time to the server.

On the server side, there is a predefined database of users which are presumed to be registered to the social network so that you do not need to implement any registration process between a client and the server. That user database has actually been provided you together with this document as a text file. You should parse this file in order to load the usernames in your server program; we can use a different file in the demos, so please do not hardcode the usernames in your code. A client, whose username should be in the user database, will be able to connect by providing his/her username only (i.e. no password or other type of security). Once connected, he/she will act as mentioned in the rest of this document.

The abovementioned introduction is for the entire project, which is to be completed in three steps. Each step is built on the previous step and each has specific deadlines and demos.

Project Step 1 (Deadline: November 24, 2021, Wednesday, 22:00):

After server starts listening, clients start to connect to the server. A connected client can post messages (sweets) and server should store them and make them ready to show in other users' feeds.

Each user can get all sweets of all other users (i.e. *sweet feed*) from the server upon request by pressing on a button on client GUI. When such request is issued, the server will send all

sweets belonging to all users in the platform except those of the requesting user. These sweets must be shown in the requesting client's GUI as they are received. The feed request can be issued repeatedly.

Each sweet has three attributes: (i) the username of the sweet owner, (ii) a unique sweet ID assigned by the server, and (iii) the timestamp (date and time) that the sweet is posted. These attributes should also be shown, in addition to the sweet itself, at the client side.

Note that a sweet owner can be offline when a particular user requests all sweets. Still, the requesting client must be able to see all sweets even if one or some of the sweet owners are offline. Therefore, server needs to keep the sweets and the related attributes in a text file. The format of this text file and the details of parsing are up to you.

Users perform all of the operations through a GUI; such as connecting to the server, entering their username, posting sweet, requesting sweet feed, etc. Additionally, upon requesting, sweets of others must be shown with their attributes on the client GUI.

This step is the basis of the project. Although the project has some additional operations (such as following, blocking, sweet deletion, etc.), you do not have to implement those in this step of the project. You will implement them in the coming steps.

For programming rules and submission specifications, please read the corresponding sections at the end of this document.

Server Specifications:

- There is only one server running on this system.
- The port number on which the server listens is not to be hardcoded; it should be taken from the Server GUI.
- The server will start listening on the specified port. It has to handle multiple clients simultaneously. To do so, whenever a client is connected to the listening port, the corresponding socket should be added to a list and the server should continuously accept other client sockets while listening.
- Only users which reside in the user database can connect to the server as a client. In other words, no user with a username that does not exist in the user database can connect.
- All activities of the server should be reported using a rich text box on the Server GUI including the usernames of the connected clients, follow operations, error messages as well as all the sweet posting and sweet feed request details. We cannot grade your project if we cannot follow what is going on; so the details contained in this text box is very important.
- Server must handle multiple connections. At the same time, one or more clients can post sweets and request their feeds to/from the server.
- Connected clients' usernames must be unique; therefore, the server must keep track of connected clients' names. If a new client comes with an existing username, server must not accept this new client.

- When the server application is closed (even abruptly), nothing should crash! Also, the process in the operating system regarding the server should be terminated properly.

Client specifications:

- The server's IP address and the port number must not be hardcoded and must be entered via the client GUI.
- There could be any number of clients in the system.
- If the server or the client application closes, the other party should understand disconnection and act accordingly. Your program must not crash!
- All activities of the client should be reported using a rich text box on the client GUI including posted sweets, error messages, and sweet feed request details (in second and third steps, more will come). We cannot grade your project if we cannot follow what is going on, so the details contained in this text box is very important.
- Each client must have a unique username. This username must be entered using the client GUI. This username is also sent to the server. The server identifies the clients using their usernames.
- Each client can post sweets and request sweet feeds at any time. If a client posts a sweet, this sweet is stored on the server side.
- Each client can disconnect from the system at any time. Disconnection can be done by pressing a disconnect button on client GUI or by just closing the client window.
- If the client application is closed (even abruptly), nothing should crash! Also, the process in the operating system regarding the client should be terminated properly.
- Both connection and sweet transfer operations will be performed using TCP sockets.

---------- End of Step 1 ----------

Project Step 2 (Deadline: December 10, 2021, Friday, 22:00):

Second step of the project is built on top of the first step. In this step, you will modify previous client and server modules to add more functionalities.

In this step, in addition to sweet posting and requesting sweet feed features of Step 1, the following features are to be added to the application.

1. Requesting and getting the list users;
2. Following other users;
3. Requesting sweet feed posted by the followed users only.

A user can request the usernames of all users in the platform from the server via the Client GUI. When requested, server must send the usernames of all users in the user database and the client GUI should display this list.

Each user can *follow* another user. The concept of *following* is the same as any social networking application; it is basically a kind of subscription to get messages posted by the users being followed. In order to follow a user, the follower should make a request to the

server via GUI. If a user aimed to be followed does not exist, server should respond with an error message, and this should be shown in both client and server GUIs. Note that follow operation does not require a permission and anyone can follow any other user. And, of course, you cannot follow yourselves!

Another mechanism that should be implemented in this step is the ability of clients to request the feed only from the followed users. In the previous step, a client was able to request sweet feed consisting of the sweets posted by all clients. You should keep this feature, but in addition to that, in this step, a client can also request the sweets posted by only the users followed. In this case, the sweet feed will contain the sweets posted by the users that the requesting client follows. Again, the feed requests can be issued repeatedly.

As in the step 1, all of the operations must be clearly shown on the client and server GUIs.

For programming rules and submission specifications, please read the corresponding sections at the end of this document.

---------- End of Step 2 ----------

Project Step 3 (Deadline: December 29, 2021, Wednesday, 22:00):

Third step of the project is built on top of the first and second steps. In this step of the project, remaining sweet and user operations are to be implemented in the application, which are (i) blocking (banning) a user, (ii) requesting the list of followers and the users followed, and (iii) deleting a sweet.

A user can block any other user in the platform so that the blocked user cannot follow her/him. If the blocked user was following the blocking client before the block operation, she/he must be removed from the follower list. After that, the server informs the blocking user about the outcome of the operation (consider success and all possible error cases here) and all details are shown on the GUIs.

When the blocked clients try to follow a user who blocked her/him, the operation is not allowed and the server must send an appropriate banning message, and this must be shown on both client and server GUI.

If a user *X* blocks user *Y*, then the sweets of *X* should never be received by *Y*.

The server should keep track of the follower relationships among the users in the system. There must be a button at the client GUI to ask for current followers of that client from the server. In response, the server should send the list of current followers of that client to be displayed on the client GUI.

Similarly, a client can request the list of users who are followed by him/her from the server. After server sends this list, it must be shown on the client GUI.

Finally, a client can delete a sweet of her/his own. For this purpose, a unique sweet identifier is needed and we use sweet ID attribute for this purpose. In order to delete a sweet, the user should send the sweet ID to the server. After that, the server deletes it and informs the user. If there is no such sweet or if there is an ownership problem, then an appropriate error message must be returned. As always, all the details must be displayed in both client and server GUIs.

As in the step 1 and 2, all of the operations must be clearly shown on the client and server GUIs.

For programming rules and submission specifications, please read the corresponding sections at the end of this document.

---------- End of Step 3 ----------

Group Work

- You can work in groups of minimum **four** and maximum **five** people for all steps (not less than four except really exceptional cases). No group changes are allowed after the first step. Any group changes must be performed before the submission of the first step. However, we do not recommend to change groups once you start the project since moving codes might lead to plagiarism.
- Equal distribution of the work among the group members is essential. All members of the group should submit all the codes for both client and server. All members should be present during the demos. In case of any dispute within the group, please do not allow the problematic group members to submit the code, so that he/she will not be graded. However, if a group member submits the same code, then the other members automatically accepts his/her contribution. In such a case, the same group continues with the second and the third steps. In other words, submitting step 1 together and separating for the other steps is not possible.
- If a particular student does not submit the first step of the project (due to being groupless or being excluded from a group due to low contribution/effort), he/she can join another group for the second and the third steps.
- Similarly, if a particular group member does not work enough in the second step, please do not let him/her submit. In such a case, the people who do not submit second step cannot join another group for the third step.
- Your TA Mustafa Aydın (mustafaaydin@sabanciuniv.edu) is responsible for keeping trach of the groupings. He will send an announcement to the class about how the group information will be collected, how underpopulated groups will be merged and how people without groups will be grouped.
- You have a chance to form your own groups. However, if you cannot find enough people to form a group, then you have to accept to work with people that we assign. If you do not like to work people that you do not know, then form your own groups!
- Please notice that neither Mustafa and other TAs nor myself is responsible for dispute resolution among the group members.

Programming Rules

- Preferred languages are C#, Java and Python, but C# is recommended.
- Your application should have a graphical user interface (GUI). **It is not a console application!**
- You must use pure TCP sockets as mentioned in the socket lab sessions. You are not allowed to use any other socket classes.
- In your application when a window is closed, **all threads related to this window should be terminated.**
- Client and server programs should be working when they are run on at least two separate computers. So please test your programs accordingly.
- Your code should be clearly commented. This affects up to 5% of your grade.
- Your program should be portable. It should not require any dependencies specific to your computer. We will download, compile and run it. If it does not run, it means that your program is not running. So do test your program before submission.

Submission

- Submit your work to SUCourse+. Each step will be submitted and graded separately.
- Delete the content of debug folders in your project directory before submission.
- Create a folder named **Server** and put your server related codes here.
- Create a folder named **Client** and put your client related codes here.
- Create a folder named **XXXX_Lastname_OtherNames_StepY**, where XXXX is your SUNet ID and Y is the project step (1, 2 or 3) (e.g. fkerem_Ors_FaikKerem_Step1). Put your Server and Client folders into this folder.
    - Compress your XXXX_Lastname_OtherNames_StepY folder **using ZIP or RAR**.
- You will be invited for a demonstration of your work. Date and other details about the demo will be announced later.
- 24 hours late submission is possible with 10 points penalty (out of 100).

We encourage the use of our WhatsApp group for the general questions related to the project. For personal questions and support, you can send email to course TAs.
Good luck!

Faik Kerem ÖRS - fkerem@sabanciuniv.edu
Mustafa AYDIN - mustafaaydin@sabanciuniv.edu
Simge Demir - simgedemir@sabanciuniv.edu
Albert LEVİ