CS 421 – Computer Networks Programming Assignment 1
NetChat

Due: March 21, 2025 at 11:59 PM

In this assignment, you will develop an instant messaging application in Java or Python. The application uses the hybrid architecture: The application uses a server for user address lookup, while messages are exchanged directly between users via a peer-to-peer (P2P) connection. The server program is provided as part of the assignment.

The goal of this assignment is to familiarize you with the internals of the HTTP and TCP protocols. Using any (third party, core or non-core) API that provides any level of abstraction specific to the HTTP protocol is not allowed. You must implement your program using either the Java Socket API of the JDK or the Socket package in the Python distribution. If you have any doubts about what to use or not, please contact **bora.tuncer@bilkent.edu.tr or furkan.guzelant@bilkent.edu.tr**.

Implementation Details

In NetChat, users can send both individual and group messages, similar to a typical messaging application. However, message history is stored only on the users' devices, meaning the server retains no chat information. When a user wants to send a message, they need the recipient's IP address and port number. If the sender does not know the recipient's address, they send an HTTP POST request to the server to retrieve it. Each client is expected to maintain a record of known addresses to avoid unnecessary requests to the server. Once a client has the recipient's address, it sends messages directly. The details for both the client and the server are provided below.

Server

The HTTP server provided with the assignment is implemented in Python. It can be run with the command:

python NetChatServer.py

Client

The client code can be implemented in Python or Java. The client code needs to be run with the command:

python NetChatClient.py <user_name>

where the <user_name> has to be given by the user.

When a client is online, the first thing it does is to update its address in the server by sending a **HTTP POST** message to the server where the body of the message should be formatted as:

UPDATE <user_name>@<IP>:<PORT>
(e.g. UPDATE bora_tuncer@192.168.1.13:1234)

When the server receives this message, it registers the user to \userlist.txt and returns a response message. If the format of the message is incorrect, the response will be 400 Bad Request, otherwise 200 OK.

To send a message to the users, the client needs the addresses of the other users. If the client does not already save the IP of the other user, the client sends an **HTTP GET** message to the server passing a query parameter "username"
(GET \userlist.txt?username=<username>)
If a user is already registered with the given user name, the server returns the IP address and PORT number of the user with the format *<IP>:<PORT>*, otherwise, it returns a 404 Not Found response. The client should save all client addresses it has received for the duration of the session.

While the client is online, it continuously asks the user for an operation until the abort command is given. When a message is received, the user should see a notification about the received message. Thus, you have to write your code in different threads, such that one thread asks the user for a command and runs the commands while the other thread only checks for the messages received. When the client sends or receives a message, your program will update the chat history between users. You can use a dictionary or other data structures to store the messages.

The client can send individual and multicast messages where multiple users receive the same messages.

In this assignment, you are required to implement five key functionalities on the client side:

- **LIST:** Lists all the user names known by the client.
  LIST
  e.g. *Waiting for a command (to exit write EXIT): LIST*
  >> [bora_tuncer, furkan_guzelant, jane_doe, john_smith]

- **SEND:** Send a message to a user. If the user is not in the client list, it should check if the user information is available on the server. If the user information is found in the server, the username is added to the client's known user list. Otherwise, "user does not exist" error message is displayed.
  SEND <user_name> <message>
  e.g. *Waiting for a command (to exit write EXIT): SEND bora_tuncer "hello"*
  >> Message Sent!

- **SEND_MULTI:** Send a message to multiple users.
  SEND_MULTI [<user_name1, user_name2>] "<message>"
  e.g. *Waiting for a command (to exit write EXIT): SEND_MULTI [bora_tuncer, furkan_guzelant, jane_doe] "hello"*

- **READ:** Displays all messages received from a user.
  READ <user_name1>
  e.g. *Waiting for a command (to exit write EXIT): READ bora_tuncer*

>> hello - bora_tuncer

>> hi - user

>> how are you? - bora_tuncer

>> yeah I'm fine - user

- **DELETE:** Deletes the last message sent to a user.

  DELETE <user_name1>

  e.g. *Waiting for a command (to exit write EXIT): DELETE bora_tuncer*

  >> hello - bora_tuncer

  >> hi - user

  >> how are you? - bora_tuncer

  >> Deleted - user

Example Outputs

python NetChatClient.py bora_tuncer

>> User registered in server!

Waiting for a command (to exit write EXIT): LIST

>> [bora_tuncer, furkan_guzelant, jane_doe, john_smith]

Waiting for a command (to exit write EXIT): READ furkan_guzelant

>> No messages found!

Waiting for a command (to exit write EXIT): SEND furkan_guzelant "hello"

>> Message sent!

Waiting for a command (to exit write EXIT):

>> You have a message from furkan_guzelant

Waiting for a command (to exit write EXIT): READ furkan_guzelant

>> hello - user

>> hi - furkan_guzelant

Waiting for a command (to exit write EXIT): DELETE furkan_guzelant

>> Deleted - user

>> hi - furkan_guzelant

Waiting for a command (to exit write EXIT): SEND_MULTI [furkan_guzelant, bora_tuncer] "I'm busy"

>> Error: Message cannot be sent to bora_tuncer, user does not exist

Waiting for a command (to exit write EXIT): READ furkan_guzelant

>> *Deleted - user*

>> *hi - furkan_guzelant*

>> *I'm busy - user*

As a **bonus assignment**, you can add additional features and mechanisms to the client part of NetChat. We will evaluate these features based on creativity and implementational complexity. The bonus part is not mandatory, and the grading of this part is separate from the original grading scheme. The bonus points that you will get will be added to the original grade.

Lastly, write a **report** (PDF file) in which you explain the important parts of your code. We should be able to navigate the source code just from the report. The number of pages should not exceed 5.

Assumption and Hints

- For threading, you can check Python Thread Tutorial or Java Thread Tutorial.

For Threading in Python:

1. Official Python Documentation: Start with the threading module in Python's official documentation. It provides a comprehensive overview and examples.
    - Visit: Python Threading Module
2. TutorialsPoint: Offers clear, beginner-friendly tutorials on threading in Python.
    - Visit: TutorialsPoint - Python Threading
3. Real Python: Find in-depth tutorials and examples here that cover threading basics and advanced topics.
    - Visit: Real Python - An Intro to Threading in Python

For Threading in Java:

1. Official Java Tutorials: The Concurrency lessons in the Java tutorials provide a solid foundation.
    - Visit: Oracle Java Docs - Concurrency
2. GeeksforGeeks: A good resource for Java threading tutorials that range from beginner to advanced levels.

    - Visit: GeeksforGeeks - Multithreading in Java
3. JavaPoint: Offers tutorials on threading with simple examples to understand the basics.
    - Visit: JavaPoint - Multithreading in Java

- Please contact your assistant if you have any doubts about the assignment.

Submission rules

You need to apply all the following rules in your submission. **You will lose points if you do not obey the submission rules below or your program does not run as described in the assignment above.**

- The assignment should be uploaded to the Moodle in a zip file. Any other methods of submission will not be accepted.
- The name of the submission should start with [CS421_PA1], and include your name and student ID. For example, the name must be

  [CS421_PA1]AliVelioglu20111222

  if your name and ID are Ali Velioglu and 20111222. You are not allowed to work in groups.
- All the files must be submitted in a zip file whose name is described above. The file must be a .zip file, not a .rar file or any other compressed file.
- All of the files must be in the root of the zip file; directory structures are not allowed. Please note that this also disallows organizing your code into Java packages. The archive should not contain:

  – Any class files or other executables,

  – Any third-party library archives (i.e. jar files),

  – Any text files,

  – Project files used by IDEs (e.g., JCreator, JBuilder, SunOne, Eclipse, Idea or NetBeans etc.). You may, and are encouraged to, use these programs while developing, but the end result must be a clean, IDE-independent program.
- The standard rules for plagiarism and academic honesty apply; if in doubt refer to Academic Integrity Guidelines for Students and Academic Integrity, Plagiarism & Cheating.

Important Notes

General Submission Rules:

1-) Your submission should include source code(s) (.java/.py).

2-) The format of the report should be **PDF**. (Do not upload .doc, .docx or any other types). The number of pages should not exceed 5.

4-) Your submission should not contain any other files other than the source code(s) (.java/.py), report (.pdf) and optional README. No .txt files, no folders, no IDE related files should be included.

5-) Compress these files with **.zip** format. (.rar, .7z or any other compressing types will not be accepted.)

6-) Make sure to follow rules in the "Submission Rules" section like name of the zip file, method of the submission etc.

For Python Submissions:

1-) The code should run with the "python PseudoGit.py <command> <repository> <branch><file_name><pr_number>" command.

2-) Python version should be **3.6 or higher**. Other versions (like Python 2) are **not accepted**.

For Java Submissions:

1-) The code should run with the following commands:

Compile: "javac *.java"

Run: "java PseudoGit <command> <repository> <branch><file_name><pr_number>" command.

2-) Java version should be **8 or higher.**

3-) The JDK should be Oracle JDK (**not** OpenJDK).