Computer network assignment2

2019082851
정승연

A. How to compile assignment2?

1. **Server program:**
   a. **Goto Server directory. And then use `javac Server.java` to compile.**
   b. **And then use `java Server [port_number_1] [port_ number_2]` command to execute server program : ex) ` java Server 1227 1228`**

      #NOTICE!!

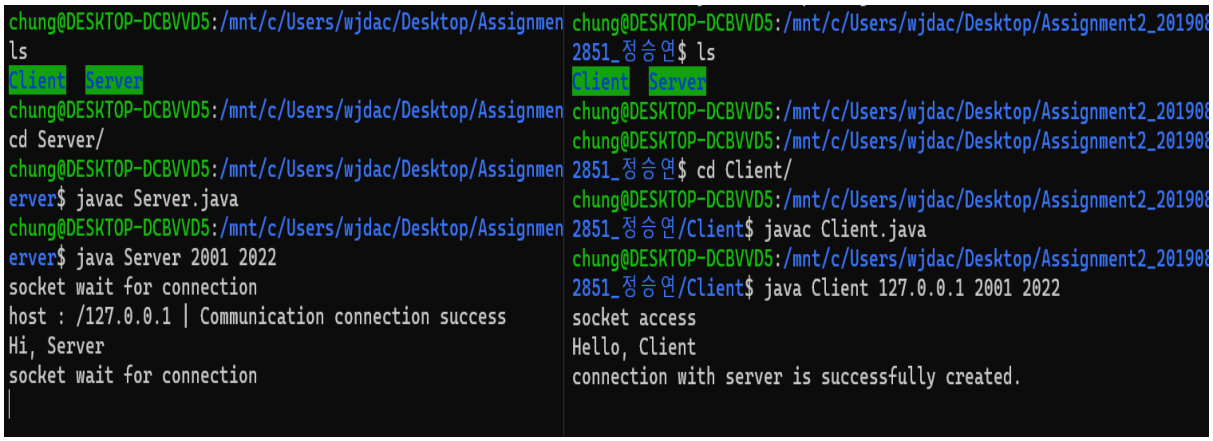
      In this project, only one server program can be executed with same argument. So, if you want to use more than one server, please use unused port numbers to prevent conflict.

      #NOTICE!!

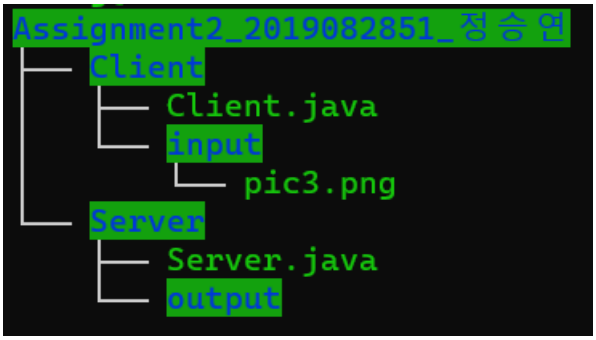
      Clients suppose that server is always running. So please execute server program with correct argument before execute client programs.
2. **Client program:**
   a. **Client Server directory. And then use `javac Client.java` to compile.**
   b. **And then use `java Client [Server IP address] [port_number_1] [port_ number_2]` command to execute server program : ex) ` java Client 127.0.0.1 1227 1228`**

These are the example of executing server program and client program.

- **Execute server first and then execute client later.**



In addition, this is tree for initial root directory.



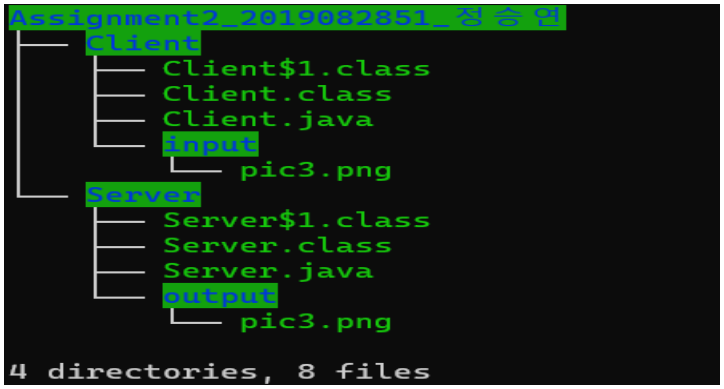
#NOTICE!!: pic3.png is sample file for upload operation.

User can upload file by putting file to Client/input and use command #PUT filename.

Also, Server/output is used to contain uploaded files.

In initial state, output directory does not contain files.

After execute A.1.a command and A.2.a command, this is tree structure for root directory.



#NOTICE!!

<port value> should be higher than 1024.

In the case of Linux, ports less than 1024 are called well-known ports and can only be run with root privileges.

#NOTICE!!

To communicate with others, users should share “same port value” and “same chat room name”.

B. Example of running program

C. Design of assignment2

a. Server role:

In the project, we suppose that server is always running.

Server should response to every socket connection.

To deal with all requests with concurrency, every socket responder get independent thread. Server program consists of one main thread and multiple sub threads which deals with each client.

a-1. Main thread role:

Main thread waits for socket connection. If socket connection is accepted, main thread open connections between server and client which are `Input Stream` and `Output Stream` to communicate! And deliver created objects to sub thread. And keep waits for socket connection and repeats.

a-2. sub thread role:

Notice! each sub thread gets unique id.

Since there are objects to communicate with clients, sub thread just responses to client's request.

At initial state, client should be joined at any chat room. So, sub thread waits for correct `#JOIN` or `#CREATE` request. In this state, other request except `join` and `create` operation will be denied. (Actually, client program will ben this operation, so that server only focuses on `join` and `create` operation.).

If correct `#CREATE` request comes, we save chat room name with id. And create member list for the chat room and add id to member list. Put output stream to output stream container which contain all output stream with unique id. Output stream container will used for delivering one user's message to all other users in the chat room. At last, save user id with unique id. This information will be used for `#STATUS` request.

If correct `#JOIN` request comes, we call chat room id (same as chat room creator id) with chat room name. Append thread id to member list. Also, save

output stream with id to Output stream container. At last, save user id with unique id.
If client join chat room successfully, then, sub thread enters `join chat` state.

At `join chat` state, sub thread should deal with 5 different request types repeatedly.

1. **`#EXIT`: sub thread will finish works after cleaning useless objects about client. If there is no member in the chat room, sub thread will clean useless objects about empty chat room.**
2. **"#STATUS": print chat room name and member name with index.**
   If member number is 3, server will return like this format.
   `Chat room name is OOO. 1. User1 2. cccc 3. yxx.`
3. **Just chat (input without # prefix): sub thread will deliver message to all other users in the chat room.**

Before explaining put and get operation, I will explain about prior knowledge. As you see tree structure of my zip file, Client directory has input directory and Server directory has output directory. I regard that no one can access output directory without permission of Server program. Also, if user want to upload file, then user should put file to input directory and then request to Server with `#PUT` operation. Also, downloaded file (By `#GET` operation) will be placed in input directory, so user can get the file by accessing input directory.

4. **`#PUT`: wait socket connection which is used for file transfer.**
   Open Input stream path whose data flow is from client to Server.
   Create file output stream with uploaded file name with output directory.
   Get file data from input stream and save it by using file output stream.
   And then, close socket, input stream and file stream.
5. **`#GET`: wait socket connection which is used for file transfer.**
   Open output stream path whose data flow is from server to client.
   Create file input stream with file name with output directory.
   Get file data from file input stream and save it by using output stream.
   And then, close socket, output stream and file stream.

b. Client role:

Before join chat room, client only allow user to use `#EXIT` `#JOIN', and `#CREATE` operation. After join chat room, one thread will keep showing messages sent by others to the user. And another thread will deal with user request.

If user send request without # prefix, we regard this as message and send it to server.
If user request `#STATUS`, we will send this command to server. Server will deal with it.
If user request `#EXIT`, we exit client program after notifying the server of this fact.
If user request `#PUT [filename]`, we first check this is valid command and then send file data to server. We will notice progress by printing `#` every 64KB.
If user request `#GET [filename]`, first, server will check this is valid command and then send file data to client. We will notice progress by printing `#` every 64KB.

D. Implementation of assignment2

a. Server.java:

To save information about chat room, we need static member to save information.

1. **`static AtomicInteger last_id` this is used to allocate unique id to sub thread.**
2. **`static HashMap<String,Integer> *chat_list*` this is used to save chat room name with chat room id.**
3. **`static HashMap<Integer,Vector<Integer>> join_member` this is used to save member id list with chat room id.**
4. **`static HashMap<Integer,OutputStream> *os_list*` this is used to save output Stream(data flows from server to client) with id.**
5. **`static HashMap<Integer,String> *member_name*` this is used to save member name with id.**

In addition to #GET operation, there is possibility that there is no file in output directory which user request. So, in this case we send `#FAIL` to client by using connection for file data transfer.

b. Client.java:

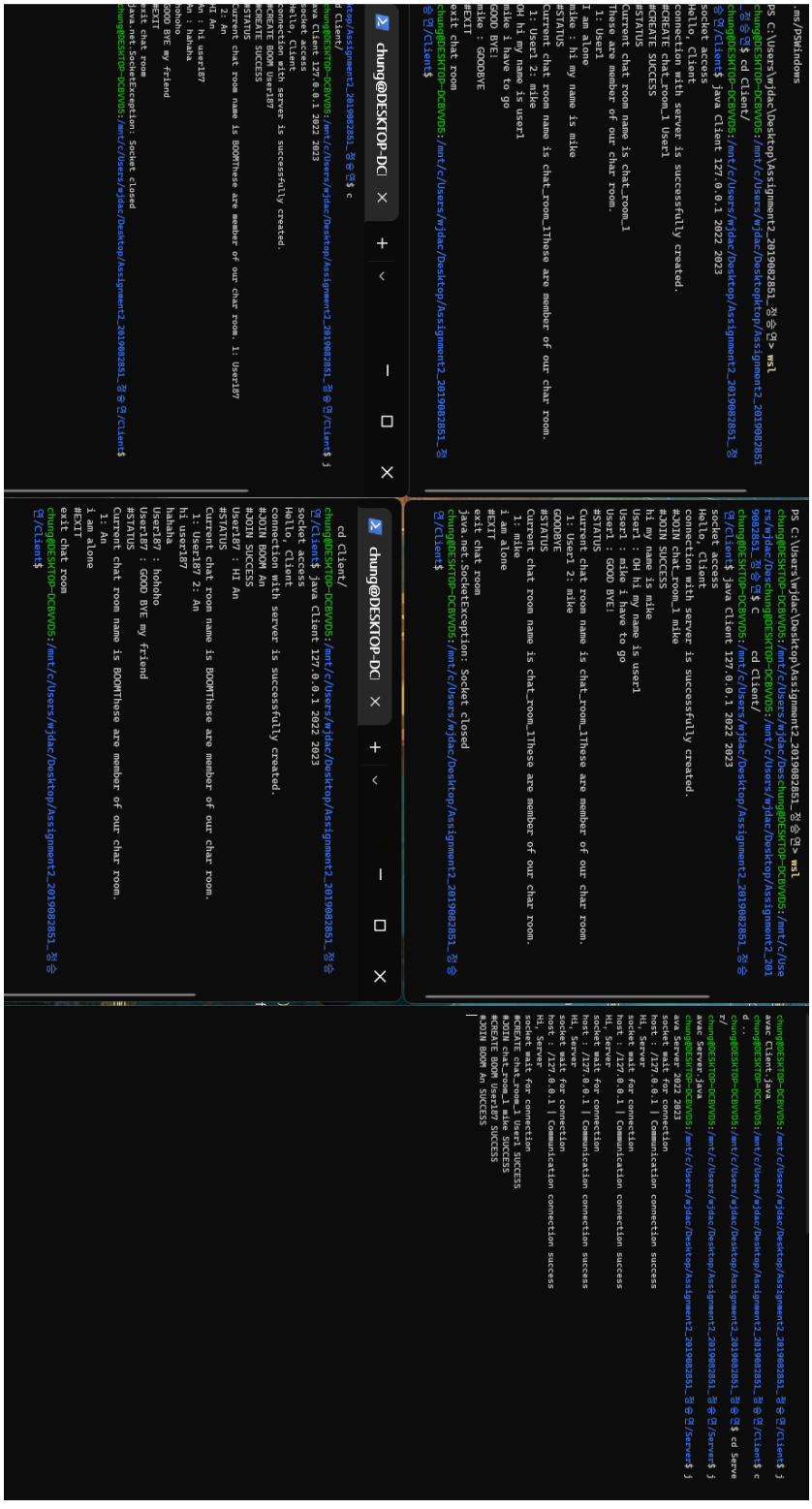
In addition to #GET and #PUT operation, you can see `Thread.sleep(300)`. I will explain why waiting process is needed.

In '#PUT' operation, we check whether socket connection for file transfer is correctly made. But there is possibility that before server allow socket for file transfer, client will check socket and think that socket connection is fail. So, to prevent that situation, client program will give server 300 milliseconds to accept socket connection. In the test, this method works well with low delay.

E. Program demonstration

1. Chatting:

User1 and mike join chat_room_1 and User187 and An join BOOM. Let see how it works.



2. **file transfer**

   As you know, there is no file in output directory and only one file named pic3.png is in the input directory.

```
socket access
Hello, Client
connection with server is successfully created.
#CREATE chat_room_1 user1
#CREATE SUCCESS
#GET pic1.png
you cannnot chat starting with # character
#PUT pic1.png
File is not found.
#GET pic3.png
#FAIL
File is not found.
#PUT pic3.png
#####
File transfer completed.
#GET pic3.png
#####
File transfer completed.
#STATUS
Current chat room name is chat_room_1
These are member of our char room. 1: user1
#EXIT
exit chat room
```