COMP3331/9331 Computer Networks and Applications

Assignment for Term 3, 2020

Version 1.3

Due: 11:59am (noon) Friday, 20 November 2020 (Week 10)

Updates to the assignment, including any corrections and clarifications, will be posted on the subject website. Please make sure that you check the subject website regularly for updates.

1. Change Log

Version 1.0 released on 5th
Version 1.1 released on 8th
Version 1.2 released on 19t
Version 1.3 released on 4th

hread/file names.

ion for 2 clients.

2. Goal and learning objectives

Online discussion forums are widely used as a means for large groups of people to hold conversations on topics of mutual interest. A good example is the online forum used for this course. In this assignment, you will have the opportunity to implement your own version of an online discussion for t server model consisting of one se tially (i.e. one at a time) or concurrentl TCP. Your application will support a range of functions that are typically found uding authentication, creation and deletion of threads and m ads, uploading and downloading files. However, unlike typical online f through HTTP, you will be designing a custom application protocol.

2.1 Learning Objectives

On completing this assignment, you will gain sufficient expertise in the following skills:

1. Detailed understanding of how online discussion forums work.
2. Expertise in socket programming.
3. Insights into designing an application layer protocol and a fully functioning networked application.

The assignment is worth **20 marks**. We will test it in two distinct configurations. In the first instance, we will test the interaction between the server and a SINGLE active client. All outlined functionality will be tested. Multiple clients will connect to the server but sequentially – one client connects, interacts, exits, the second client connects, interacts, exits and so on. **The first configuration is worth 14 marks (70% of the total mark)**. In the second instance, we will test the interaction of the server with multiple **concurrent** clients. All outlined functionality will be tested. **The second configuration is worth 6 marks**. Submissions from CSE students will be tested in both configurations. Submissions from non-CSE students will only be tested in the first configuration. The marking guidelines are thus different for the two groups and are indicated in Section 7.

Non-CSE Student: The rationale for this option is that students enrolled in a program that does not include a computer science component have had very limited exposure to programming and in particular working on complex programming assignments. A Non-CSE student is a student who is not enrolled in a CSE program (single or double degree). Examples would include students enrolled exclusively in a single degree program such as Mechatronics or Aerospace or Actuarial Studies or Law. Students enrolled in dual degree programs that include a CSE program as one of the degrees do not qualify. Any student who meets this criteria and wishes to avail of this option MUST email cs3331@cse.unsw.edu to seek approval before **5pm**, **16th October (Friday, Week 5)**. We will assume by default that all students are attempting the CSE version of the assignment unless they have sought explicit permission. **No exceptions**.

3. Assignment Specification

In this programming assi programs of a discussion forum applicati we use for this course. The difference bein TP) but uses a custom application layer pr erver must communicate over TCP. Your application will support a range of operations including create a new user account, create or delete a new thread, post a message on a thread, edit or delete messages, upload and download attachments to/from a thread, read a thread, list all threads and shutting down the server. You will implement the application protocol to implement these functions. The server will listen on a port specified as the command line argumen, and will wait for a client to connect. The client program will initiate a TCP connection with the server. Upon connection establish ss. The client will interact with the user throug authentication, the user will initiate one of t imple request response interaction between the client and server. The user ommands (one after the other) and eventually quit. Both the client and ser gful messages at the command prompt that capture the specific intera are free to choose the precise text that is displayed. Examples of client iven in Section 8.

The assignment will be tested in two configurations. In the **first configuration**, the server will interact with a single client at any given time. Multiple clients can connect with the server in a serial fashion, i.e., one client connects, interacts and quits, the second client connects, interacts and quits, and so on. The server design is significantly simplified (i.e. you won't need to use multi-threading) if you only wish to implement this portion of the assignment. A correct implementation of this first part is worth **70% of the assignment mark**s (14 marks, see Section 7). In the **second configuration**, the server must interact with multiple clients concurrently. The client design will only require minimal changes to meet this requirement. The server design, however, would require a significant change, in that, the server would need to send and receive messages to and from multiple clients concurrently. We strongly recommend using **multi-threading** to achieve this. The interaction with a single client, would however be similar as in the first configuration. Note that, a correctly implemented multi-threaded server should also be able to interact correctly with a single client at any given time. So, if you design your client and server to achieve all functionality expected for the second configuration, it should work as expected in the first configuration.

3.1 File Names & Execution

The main code for the server and client should be contained in the following files: `server.c`,

or `Server.java` or `server.py`, and `client.c` or `Client.java` or `client.py`. You are free to create additional files such as header files or other class files and name them as you wish. Submission instructions are in Section 5.

The server should accept the following arguments:

- `server_port`: this is the port number which the server will use to communicate with the clients. Recall that a TCP socket is NOT uniquely identified by the server port number. It should thus be possible for multiple TCP connections to use the same server-side port number (in Part 2).
- `admin_passwd`: this is the admin password for the server. It is required to shut down the server (see operation SHT later).

The server should be executed before any of the clients. It should be initiated as follows:

If you use Java:

```
java Server server_po
```

If you use C:

```
./server server_port admin_passwd
```

If you use Python:

```
python server.py server_port admin_passwd OR
python3 server.py server_port admin_passwd
```

The client should ac

- `server_IP`: this is the IP address of the ma er is running.
- `server_port`: this is the port number being argument should be the same as the first argument of the serve

Note that, you do not have to specify the port to be used by the client. You should allow the OS to pick a random available port. Each client should be initiated in a separate terminal as follows:

If you use Java:

```
java Client server_IP server_port
```

If you use C:

```
./client server_IP server_port
```

If you use Python:

```
python client.py server_IP server_port OR
python3 client.py server_IP server_port
```

Note: When you are testing your assignment, you should run the server and one or more clients on the same machine in separate terminals. In this case, use 127.0.0.1 (local host) as the server IP address.

3.2 Authentication

You may assume that a credentials file called *credentials.txt* will be available in the current working directory of the server with the correct access permissions set (read and write). This file will contain username and passwords of authorised users. They contain uppercase characters (A-

Z), lowercase characters (a-z) and digits (0-9) and special characters (~!@#$%^&*_-+=`|\(){}[]:;"'<>,.?/). An example *credentials.txt* file is provided on the assignment page. We may use a different file for testing so DO NOT hardcode this information in your program. You may assume that each username and password will be on a separate line and that there will be one white space between the two. There will only be one password per username. A sample credentials file is provided on the assignment page. We may use a different file while testing.

Upon execution, a client should first attempt to setup a TCP connection with the server. Assuming the connection is successful, the client should prompt the user to enter a username. The username should be sent to the server. The server should check the credentials file (*credentials.txt*) for a match. If the username exists, the server sends a confirmation message to the client. The client prompts the user to enter a password. The password is sent to the server, which checks for a match with the stored password for this user. The server sends a confirmation if the password matches or an error message in the event of a mismatch. An appropriate message is displayed to the user. In nter a username. If the username does not ex ient. The client prompts the user to enter a The server creates a new username and password entry in the credentials file (appending it as the last entry in the file). A confirmation is sent to the client. The client displays an appropriate message to the user. You should make sure that write permissions are enabled for the credentials.txt file (type "`chmod +w credentials.txt`" at a terminal in the current working directory of the server).

When your assignm er should also check that a new client tha to login with a username that is already being be used concurrently by two clients). The server should keep track of all active users and check that the username provided by an authenticating client does not match match is found, then a message to this effect should be sent to the s prompt for the user and they should be prompted to enter a username.

3.3 Discussion Forum Operations

Following successful login, the client displays a message to the user informing them of all available commands and prompting to select one command. The following commands are available: CRT: Create Thread, LST: List Threads, MSG: Post Message, DLT: Delete Message, RDT: Read Thread, EDT: Edit Message, UPD: Upload File, DWN: Download File, RMV: Remove Thread, XIT: Exit, SHT: Shutdown Server. All available commands should be shown to the user in the first instance after successful login. Subsequent prompts for actions should include this same message.

If an invalid command is selected, an error message should be shown to the user and they should be prompted to select one of the available actions.

In the following, the implementation of each command is explained in detail. The expected usage of each command (i.e. syntax) is included. **Note that, all commands should be upper-case (CRT, MSG, etc.)**. All arguments (if any) are separated by a single white space and will be one word long (except messages which can contain white spaces). **You may assume that all arguments including thread names, file names and the message text may contain uppercase characters (A-Z), lowercase characters (a-z) and digits (0-9) and the following limited set of special characters (!@#$%.?,).**

If the user does not follow the expected usage of any of the operations listed below, i.e., missing (e.g., not specifying the title of the thread when creating a thread) or incorrect number of arguments (e.g., inclusion of additional or fewer arguments than required) , an error message should be shown to the user and they should be prompted to select one of the available commands. Section 8 illustrates sample interactions between the client and server.

There are 11 commands that users can execute. The execution of each individual command is described below.

CRT: Create Thread

```
CRT threadtitle
```

The title of the new thread (*threadtitle*) should be included as an argument with this command. Thread titles are **one word**), the title of the thread and the username to c equrrent working directory of the se *tle*, DO NOT add ".txt" extension to the name). The first line of the file should contain the username who created the thread. Each subsequent line should be a message, added in the chronological sequence in which they were posted. The server should first check if a thread with this title exists. If so, an error message should be conveyed to the client and displayed at the prompt to the user. If the thread does not exist, a new file with the provided title should be created as per the convention noted above (the first line of this file should be the username of the creator). You may assume that the server progr nt working directory. A confirmation messa prompt to the user. The client should next pr nds.

MSG: Post Message

```
MSG threadtitle message
```

The title of the thread that the message should be posted to and the message should be included as arguments. Note that, the message may contain white spaces (e.g. "hello how are you"). The client should send the command (MSG), the title of the thread, the message and the username to the server. **In our tests, we will only use short messages (a few words long)**. The server should first check if a thread with this title exists. If so, the message and the username should be appended at the end of the file in the format, along with the number of the message (messages within each thread are numbered starting at 1):

```
messagenumber username: message
```

An example:

```
1 yoda: do or do not, there is no try
```

A confirmation message should be sent to the server and displayed to the user. If the thread with this title does not exist, an error message should be sent to the client and displayed at the prompt to the user. The client should next prompt the user to select one of the available commands.

DLT: Delete Message

```
DLT threadtitle messagenumber
```

The title of the thread from which the message is to be deleted and the message number within that thread to be deleted should be included as arguments. **A message can only be deleted by the user who originally posted that message**. The client sends the command (DLT), the title of the thread, the message number and the username to the server. The server should check if a thread with this title exists and if the corresponding message number is valid and finally if this user had originally posted this message. In the event that any of these checks are unsuccessful, an appropriate error message should be sent to the client and displayed at the prompt to the user. If all checks pass, then the server should delete the message, which entails deleting the line containing this message in the corresponding thread file (all subsequent messages in the file should be moved up by one line and their message numbers should be updated appropriately) and a confirmation should e user. The client should next prompt the use

EDT: Edit Message

```
EDT threadtitle messagenumber message
```


The title of the thread from which the message is to be edited, the message number within that thread to be edited and the new message should be included as arguments. **A message can only be edited by the us** ould send the command (EDT), the title of th the username to the server. The server s he corresponding message number is valid and f event that any of these checks are unsuccessful, an appropriate error o the client and displayed at the prompt to the user. If all checks pas replace the original message in the corresponding thread file with the ne the details associated with this message, i.e. message number and username should remain unchanged) and a confirmation should be sent to the client and displayed at the prompt to the user. The client should next prompt the user to select one of the commands.

LST: List Threads

```
LST
```

There should be no arguments for this command. The client sends the command (LST) to the server. The server replies back with a listing of all the thread titles. Only the thread titles should be listed, not the messages. The client should print the list on the terminal (one thread per line). If there are no active threads, then a message to that effect should be displayed at the prompt to the user. The client should next prompt the user to select one of the available commands.

RDT: Read Thread

```
RDT threadtitle
```

The title of the thread to be read should be included as an argument. The client should send the command (RDT) and the title of the thread to be read to the server. The server should check if a thread with this title exists. If so, the server should send the contents of the file corresponding to

this thread (excluding the first line which contains the username of the creator of the thread) to the client. The client should display all contents of the file including messages and information about uploaded files (see next action) at the terminal to the user. If the thread with this title does not exist, an error message should be sent to the client and displayed at the prompt to the user. The client should next prompt the user to select one of the available commands.

UPD: Upload file

```
UPD threadtitle filename
```

The title of the thread to which the file is being uploaded to and the name of the file should be included as arguments. You may assume that the file included in the argument will be available in the current working directory of the client with the correct access permissions set (read). You should not assume that the file will be in a particular format, i.e., just assume that it is a **binary file**. The client should sen o the server. The server should check if a thr ppropriate error message should be sent to t the thread exists, then a confirmation message should be sent to the client. The client should next send the username and file name to the server. Following this, the client should transfer the contents of the file to the server. The file should be stored in the current working directory of the server with the file name threadtitle-filename (DO NOT add an extension to the name. If the filename has an extension in the name, retain it, e.g., *test.exe* should be stored as *threadtitle-test.exe*). File names are **case sensitive** and **one word long**. You may assume that the server program will have permission to create sume that the file name will be unique for ea een uploaded to a thread, then no other user w However, a file with the same name coul le should be noted on the thread, i.e., an entry should be added at the end of g to the thread title indicating that this user has uploaded a file with the sp a should be as follows (note the lack of a message number which d essage):


```
Username uploaded filename
```

The entries for file uploads cannot be edited using the EDT command or deleted using the DLT command. They should however be included when a thread is read using the RDT command. Finally, the server should send a confirmation message to the client and a message to this effect should be displayed at the prompt to the user. The client should next prompt the user to select one of the available commands.

DWN: Download file

```
DWN threadtitle filename
```

The title of the thread from which the file is being downloaded and the name of the file should be included as arguments. The client should send the title of the thread and the name of the file to the server. The server should check if a thread with this title exists and if so whether the file with this name was previously uploaded to the thread. If either check does not match, then an appropriate error message should be sent to the client and displayed at the prompt to the user. If a match is found, then the server should transfer the contents of the file to the client. The client should write the contents to a local file in the current working directory of the client with the same name (*filename,* DO NOT include *threadtitle* in the file name). You may assume that the

client program will have permission to create a file in the current working directory. You may also assume that a file with this same name does not exist in the current working directory of the client. Once the file transfer is complete, a confirmation message should be displayed at the prompt to the user. The client should next prompt the user to select one of the available commands. Note that, the file should NOT be deleted at the server end. The client is simply downloading a copy of the file.

TESTING NOTES: (1) When you test the operation of this command, you will likely first upload a test file from the client to the server using the previous command UPD and then try to download the same file from the server using the DWN command. You should make sure that you remove this file from the current working directory of the client between these two commands (to be consistent with the assumption stated in the description above). You can do this by opening a separate terminal and deleting this file from the client's working directory. (2) For similar reasons, when testing your program under the second configuration, make sure that the multiple clients are exe

RMV: Remove Thread

```
RMV threadtitle
```


The title of the thread to be removed should be included as an argument with this action. **A thread can only be removed by the user who originally created that thread.** The client should send the operation (RMV), the title of the thread and the username to the server. The server should first check if r who created the thread matches with tch, then an error message should be sent to the se, the thread is deleted including the file sto d to the thread and any state maintained about the thread at the server. A conf d be sent to the client which is displayed at the prompt to the user. T mpt the user to select one of the available actions.

XIT: Exit

```
XIT
```

There should be no arguments for this command. The client should close the TCP connection and exit with a goodbye message displayed at the terminal to the user. The server should update its state information about currently logged on users. Note that, any messages and files uploaded by the user must not be deleted.

SHT: Shutdown

```
SHT admin_password
```

The admin password should be provided as the argument. The client should send the command (SHT) and the admin password to the server. **Note that, the admin password is provided to the server as the second command line argument during execution. It is NOT included in the credentials file**. The server should check the provided password against the admin password. If the passwords do not match, then an error message should be sent to the client and displayed at the prompt to the user. The client should next prompt the user to select one of the available actions. If the passwords match, then the server should initiate shutdown process. This includes

sending a shutdown message to all active clients (in the case when we are testing with multiple concurrent clients). Each client will display an appropriate message at the terminal to the user indicating that the discussion forum is shutting down and close the socket. The server should delete all files that were (only) created by the server program in the current working directory including files for all active threads and any files uploaded to the threads and the credentials file. The client need not delete any files in the current working directory. All sockets should be closed.

3.3 Program Design Considerations

Transport Layer

You MUST use **TCP** for this assignment. This ensures that your client and server programs do not have to worry about reliable delivery of messages to each other. **The use of UDP is likely to attract a heavy penalty**.

Client Design

The client program should be fairly straightforward. The client needs to interact with the user through the command line interface and print meaningful messages. Section 8 provides some examples. You do not have to use the exact same text as shown in the samples. Upon initiation, the client should establish a TCP connection with the server and execute the user authentication process. Following authentication, the user should be prompted to enter one of the available commands. Almost all commands require simple request/response interactions between the client with the server tate about the discussion forum.

Only one minor change would be needed in the client design as you progress the implementation from the first configuration to the second configu the shutdown process. In the first configuration, only the client cur server can initiated the shutdown process. However, in the second configur rrently connected clients could initiate the shutdown process. This would mean a client instance may receive a message from the server at any time indicating that it is shutting down (due to the issuance of the SHT command by one of the other clients). This is the only difference with the first configuration on the client end. A client program that correctly implements functionality for the second configuration should be able to correctly accomplish all interactions expected in the first configuration.

Server Design

The server code will be fairly involved compared to the client as the server is responsible for maintaining the message forum. However, the server design to implement functionality for the first configuration of testing should be relatively straightforward as the server needs to only interact with one client at a time. **When the server starts up, the forum is empty – i.e., there exist no threads, no messages, no uploaded files**. The server should wait for a client to connect, perform authentication and service each command issued by the client sequentially. After the client exits, the server should wait for a new client to connect. Note that, you will need to define a number of data structures for managing the current state of the forum (threads, posts, files). Implementing functionality for the second configuration will require a significant change as the server must interact with multiple clients simultaneously. A robust way to achieve this to use **multithreading**. In this approach, you will need a main thread to listen for new connections. This can be done using the socket accept function within a while loop. This main thread is your

main program. For each connected client, you will need to create a new thread. When interacting with one particular client, the server should receive a request for a particular operation, take necessary action and respond accordingly to the client and wait for the next request. This process is exactly similar to what you would have implemented to meet the functionality of the first configuration. You may assume that each interaction with a client is **atomic**. Consider that client A initiates an interaction (i.e., a command) with the server. While the server is processing this interaction, it cannot be interrupted by a command from another client B. Client B's command will be acted upon after the command from client A is processed. Once a client exits, the corresponding thread should also be terminated. You should be particularly careful about how multiple threads will interact with the various data structures. Code snippets for multi-threading in all supported languages are available on the course webpage. A server program that correctly implements functionality for the second configuration should be able to correctly accomplish all interactions expected in the first configuration.

4. Additional Notes

- This is NOT group assi ally
- **Tips on getting starte** task is to do it in stages. We recommend that you first implement the functionality for the first configuration, i.e., the server interacts with a single active client at any time. A good place to start would be to implement the functionality to allow a single user to login with the server. Next, add functionality to implement one command. Ensure you thoroughly test the operation of each command, including typical error conditions, and then progress to the next. We recommend that you start wi efore progressing to more complex c thoroughly tested your code for the firs t is imperative that you rigorously test y teractions can be correctly executed. **Test, test and test**.
- **Application Layer Protocol:** Remember that y pplication layer protocol for realising a fully functional discussio to design the format (both syntax and semantics) of the messages exchanged between the client and server and the actions taken by each entity on receiving these messages. We do not mandate any specific requirements with regards the design of your application layer protocol. We are only considered with the end result, i.e. the functionality outlined above. You may wish to revisit some of the application layer protocols that we have studied (HTTP, SMTP, etc.) to see examples of message format, actions taken, etc.
- **Transport Layer Protocol:** You should use TCP for transferring messages between each client and server. The TCP connection should be setup by the client on initiation and should remain active until the user exits or one of the other concurrently connected users initiates shutdown (only in the second configuration). The server port is specified as a command line argument. The client port does not need to be specified. Your client program should let the OS pick a random available port.
- **Backup and Versioning:** We strongly recommend you to back-up your programs frequently. CSE backups all user accounts nightly. If you are developing code on your personal machine, it is strongly recommended that you undertake daily backups. We also recommend using a good versioning system so that you can roll back and recover from any inadvertent changes. There are many services available for both which are easy to use. We will NOT entertain any requests for special consideration due to issues related to computer failure, lost files, etc.

- **Language and Platform**: You are free to use C, JAVA or Python to implement this assignment. Please choose a language that you are comfortable with. The programs will be tested on CSE Linux machines. So please make sure that your entire application runs correctly on these machines (i.e. your lab computers) or using VLAB. This is especially important if you plan to develop and test the programs on your personal computers (which may possibly use a different OS or version or IDE). Note that CSE machines support the following: **gcc version 8.2, Java 11, Python 2.7 and 3.7. If you are using Python, please clearly mention in your report which version of Python we should use to test your code.** You may only use the basic socket programming APIs providing in your programming language of choice. You may not use any special ready-to-use libraries or APIs that implement certain functions of the spec for you.
- There is no requirement that you must use the same text for the various messages displayed to the user on the terminal as illustrated in the examples in Section 8. However, please make sure that the text is clea
- You are strongly enco questions and to discuss different approa **not** post your solution or any code fragments on the forums.
- We will arrange for additional consultations in Weeks 7-10 to assist you with assignment related questions. Information about the consults will be announced via the website.

5. Submission

Please ensure that y ou may of course have additional header file ubmit a makefile/script alon . This is because we need to know how to resolve the dependencies amo ave provided. After running your makefile we should have the followin er and `client`. In addition, you should submit a small report, `report` ages) describing the program design, the application layer message format and a brief description of how your system works. Also discuss any design trade-offs considered and made. Describe possible improvements and extensions to your program and indicate how you could realise them. If your program does not work under any particular circumstances, please report this here. If you have not implemented functionality for handling multiple concurrent clients, then you should indicate this in the report. Also indicate any code segments that were borrowed from the Web or other sources.

You are required to submit your source code and `report.pdf`. You can submit your assignment using the give command through VLAB. Make sure you are in the same directory as your code and report, and then do the following:

1. Type tar -cvf assign.tar filenames
e.g. `tar -cvf assign.tar *.java report.pdf`

2. When you are ready to submit, at the bash prompt type `3331`

3. Next, type: `give cs3331 Assign assign.tar` (You should receive a message stating the result of your submission). The same command should be used for 3331 and 9331.

Alternately, you can also submit the tar file via the WebCMS3 interface on the assignment page.

Important Notes

- The system will only accept `assign.tar` submission name. All other names will be rejected.

- **Ensure that your program/s are tested in the VLAB environment before submission. In the past, there were cases where tutors were unable to compile and run students' programs while marking. To avoid any disruption, please ensure that you test your program in the VLAB environment before submitting the assignment. Note that, we will be unable to award any significant marks if the submitted code does not run during marking.**

- You may submit as many times before the deadline. A later submission will override the earlier submission, so m the last moment to submit, as t l not have time to rectify it.

Late Submission Penalty: Late penalty will be applied as follows:

- 1 day after deadline: 10% reduction
- 2 days after deadline: 20% reduction
- 3 days after deadline: 30% reduction
- 4 days after deadline: 40% reduction
- 5 or more da

NOTE: The above be bmit your assignment 1 day lat final mark will be 10 – 1 (10% penalty) = 9.

6. Plagiarism

You are to write all of the code for this assignment yourself. All source codes are subject to strict checks for plagiarism, via highly sophisticated plagiarism detection software. These checks may include comparison with available code from Internet sites and assignments from previous semesters. In addition, each submission will be checked against all other submissions of the current semester. Do not post this assignment on forums where you can pay programmers to write code for you. We will be monitoring such forums. Please note that we take this matter quite seriously. The LIC will decide on appropriate penalty for detected cases of plagiarism. The most likely penalty would be to reduce the assignment mark to ZERO. We are aware that a lot of learning takes place in student conversations, and don't wish to discourage those. However, it is important, for both those helping others and those being helped, not to provide/accept any programming language code in writing, as this is apt to be used exactly as is, and lead to plagiarism penalties for both the supplier and the copier of the codes. Write something on a piece of paper, by all means, but tear it up/take it away when the discussion is over. It is OK to borrow bits and pieces of code from sample socket code out on the Web and in books. You MUST however acknowledge the source of any borrowed code. This means providing a reference to a book or a URL when the code appears (as comments). Also indicate in your report the portions of your code that were borrowed. Explain any modifications you have made (if any) to the borrowed code.

7. Marking Policy

The following table outlines the marking rubric for both CSE and non-CSE students. For CSE students, **14 marks** are attributed towards testing the interaction between the server and one active client (multiple clients will connect sequentially one after the other as in the sample interaction provided). **6 marks** are attributed towards testing the interaction between the server and multiple concurrent clients. You should test your program rigorously before submission. **All submissions will be manually marked by your tutors and NOT auto marked**. Some helper scripts may be used to assist with the marking. Your submissions will be marked using the following criteria:

Functionality	**Marks (CSE)**	**Marks (Non-CSE)**
Successful authentication error handling		1.5
Successful creation of a ne error handling		1.5
Successful creation of a new message (MSG command) including all error handling	1	1.5
Successful listing of active threads (LST command) including all error handling	0.5	0.75
Successful reading of an active thread (RDT command) including all error handling	1	1.5
Successful editing all error handling	1	1.5
Successful deletion of an existing message (DLT command including all error handling		1.5
Successful deletion of an active thread (RMV comm all error handling		1.5
Successful uploading of a file to a thread (UPD command) including all error handling	1.5	2.25
Successful download of a file from a thread (DWN command) including all error handling	1.5	2.25
Successful log off for a logged in user (XIT command) including all error handling	0.5	0.75
Successful shutdown of the server (SHT command) including all error handling	1	1.5
Properly documented report	1	1
Code quality and comments	1	1
Successful authentication of multiple concurrent existing and new users including all error handing	0.5	N/A
Successful execution of all 11 commands and associated error handling (11 x 0.5 marks each)	5.5	N/A

NOTE: While marking, we will be testing for typical usage scenarios for the above functionality and some straightforward error conditions. A typical marking session will last for about 15-20 minutes. When testing with multiple concurrent clients, we will spawn a maximum of 3 concurrent clients. However, please do not hard code any specific limits in your programs. We won't be testing your code under very complex scenarios and extreme edge cases.

8. Sample Interaction

In the following we provide examples of sample interactions for both configurations to be tested. Your server and client code should display similar meaningful messages at the terminal. You **do not** have to use the same text as shown below. Note that, this is not an exhaustive summary of all possible interactions. Our tests will not necessarily follow this exact interaction shown.

First Configuration

In this configuration, the server interacts with a single client at any given time. It is recommended to execute the client and server in different working directories. Ensure that write permissions are enabled on the credentials file. In the following, two clients with usernames Yoda and Obi-wan connect and interact with the server sequentially in that order. The inputs from the user are shown as underlined in the r terminal to align the output with corre

der

Client Terminal	
	>java Server 5000 destroyforum
>java Client 127.0.0.1 5000	Waiting for clients
Enter username: Yoda	Client connected
Enter password: gdrfdfs12	Incorrect password
Invalid passwor	
Enter username:	
Enter password:	
Welcome to the forum	
Enter one of the following commands: CRT, MSG, DLT, EDT, LST, RDT, UPD, DWN, RMV, XIT, SHT: LST	
No threads to list	Yoda issued LST command
Enter one of the following commands: CRT, MSG, DLT, EDT, LST, RDT, UPD, DWN, RMV, XIT, SHT: HELLO	
Invalid command	
Enter one of the following commands: CRT, MSG, DLT, EDT, LST, RDT, UPD, DWN, RMV, XIT, SHT: CRT 3331	Yoda issued CRT command
Thread 3331 created	Thread 3331 created
Enter one of the following commands: CRT, MSG, DLT, EDT, LST, RDT, UPD, DWN, RMV, XIT, SHT: CRT 3331	Yoda issued CRT command
Thread 3331 exists	Thread 3331 exists
Enter one of the following commands: CRT, MSG, DLT, EDT, LST, RDT, UPD, DWN, RMV, XIT, SHT: CRT 9331	Yoda issued CRT command
Thread 9331 created	Thread 9331 created
Enter one of the following commands: CRT, MSG, DLT, EDT, LST, RDT, UPD, DWN, RMV,	

XIT, SHT: LST 3331 Incorrect syntax for LST Enter one of the following commands: CRT, MSG, DLT, EDT, LST, RDT, UPD, DWN, RMV, XIT, SHT: LST The list of active threads: 3331 9331 Enter one of the following commands: CRT, MSG, DLT, EDT, LST, RDT, UPD, DWN, RMV, XIT, SHT: MSG 3331 Networks is awesome Message posted to 3331 thread Enter one of the fol MSG, DLT, EDT, LST, R XIT, SHT: RDT Incorrect syntax for Enter one of the following commands: CRT, MSG, DLT, EDT, LST, RDT, UPD, DWN, RMV, XIT, SHT: RDT 9331 Thread 9331 is empty Enter one of the following commands: CRT, MSG, DLT, EDT, LST, RDT, UPD, DWN, RMV, XIT, SHT: RDT 3 1 Yoda: Network Enter one of t MSG, DLT, EDT, LST, RDT, UPD, DWN, RMV, XIT, SHT: UPD 3331 test.exe test.exe uploaded to 3331 thread Enter one of the following commands: CRT, MSG, DLT, EDT, LST, RDT, UPD, DWN, RMV, XIT, SHT: RDT 3331 1 Yoda: Networks is awesome Yoda uploaded test.exe Enter one of the following commands: CRT, MSG, DLT, EDT, LST, RDT, UPD, DWN, RMV, XIT, SHT: RMV 9331 Thread 9331 removed Enter one of the following commands: CRT, MSG, DLT, EDT, LST, RDT, UPD, DWN, RMV, XIT, SHT: XIT Goodbye >java Client 127.0.0.1 5000 Enter username: Obi-wan Enter new password for Obi-wan: r2d2 Enter one of the following commands: CRT,	Yoda issued LST command Yoda issued MSG command Message posted to 3331 thread Yoda issued RDT command Thread 9331 read Yoda issued RDT command mand est.exe to 3331 thread Yoda issued RDT command Thread 3331 read Yoda issued RMV command Thread 9331 removed Yoda exited Waiting for clients Client connected New user Obi-wan successfully logged in

MSG, DLT, EDT, LST, RDT, UPD, DWN, RMV, XIT, SHT: CRT 9331	
Thread 9331 created	Obi-wan issued CRT command Thread 9331 created
Enter one of the following commands: CRT, MSG, DLT, EDT, LST, RDT, UPD, DWN, RMV, XIT, SHT: MSG 9331 Networks exam PWNED me	
Message posted to 9331 thread	Obi-wan issued MSG command Obi-wan posted to 9331 thread
Enter one of the following commands: CRT, MSG, DLT, EDT, LST, RDT, UPD, DWN, RMV, XIT, SHT: MSG 3331 Networks exam PWNED me	
Message posted to 3331 thread	Obi-wan issued MSG command Obi-wan posted to 3331 thread
Enter one of the fol MSG, DLT, EDT, LST, R XIT, SHT: LST	
The list of active th 3331 9331	mmand
Enter one of the following commands: CRT, MSG, DLT, EDT, LST, RDT, UPD, DWN, RMV, XIT, SHT: RDT 331	
Thread 331 does	Obi-wan issued RDT command
Enter one of t MSG, DLT, EDT, XIT, SHT: RDT 3	command
1 Yoda: Networks is awesome Yoda uploaded test.exe	
2 Obi-wan: Networks exam PWNED me	
Enter one of the following commands: CRT, MSG, DLT, EDT, LST, RDT, UPD, DWN, RMV, XIT, SHT: DWN 9331 test.exe	
File does not exist in Thread 9331	Obi-wan issued DWN command test.exe does not exist in Thread 9331
Enter one of the following commands: CRT, MSG, DLT, EDT, LST, RDT, UPD, DWN, RMV, XIT, SHT: DWN 3331 test.exe	
test.exe successfully downloaded	Obi-wan issued DWN command test.exe downloaded from Thread 3331
Enter one of the following commands: CRT, MSG, DLT, EDT, LST, RDT, UPD, DWN, RMV, XIT, SHT: EDT 3331 1 I PWNED Networks exam	
The message belongs to another user and cannot be edited	Obi-wan issued EDT commend Message cannot be edited
Enter one of the following commands: CRT, MSG, DLT, EDT, LST, RDT, UPD, DWN, RMV, XIT, SHT: EDT 3331 2 I PWNED Networks exam	
The message has been edited	Obi-wan issued EDT commend Message has been edited
Enter one of the following commands: CRT,	

MSG, DLT, EDT, LST, RDT, UPD, DWN, RMV, XIT, SHT: RDT 3331	
1 Yoda: Networks is awesome	Obi-wan issued RDT command
Yoda uploaded test.exe	Thread 3331 read
2 Obi-wan: I PWNED Networks exam	
Enter one of the following commands: CRT, MSG, DLT, EDT, LST, RDT, UPD, DWN, RMV, XIT, SHT: RMV 3331	
The thread was created by another user and cannot be removed	Obi-wan issued RMV command
	Thread 3331 cannot be removed
Enter one of the following commands: CRT, MSG, DLT, EDT, LST, RDT, UPD, DWN, RMV, XIT, SHT: RMV 9331	
The thread has been r	Obi-wan issued RMV command
Enter one of the fol MSG, DLT, EDT, LST, R XIT, SHT: LST	mmand
The list of active threads:	
3331	
Enter one of the following commands: CRT, MSG, DLT, EDT, LST, RDT, UPD, DWN, RMV, XIT, SHT: SHT	
Incorrect synta	
Enter one of t MSG, DLT, EDT, XIT, SHT: SHT monkey	command
Incorrect password	
Enter one of the following commands: CR MSG, DLT, EDT, LST, RDT, UPD, DWN, RMV, XIT, SHT: SHT destroyforum	
Goodbye. Server shutting down	Obi-wan issued SHT command
	Server shutting down
>	>

der

Second Configuration

In this configuration, the server interacts concurrently with multiple clients. In the following, two clients with usernames Yoda and R2D2 connect and interact with the server concurrently. The inputs from the user are shown as underlined. It is strongly recommended to execute the sever and each individual client in a separate working directory. Ensure that write permissions are enabled on the credentials file.

Note that, extra space is added in the two client terminals to simulate some delay before the users enter commands when prompted to do so. This is simply done to improve readability of the output below. You should not make such assumptions in your implementation.

Client 1 Terminal	Client 2 Terminal	Server Terminal
>java Client 127.0.0.1 6000 Enter username: Yoda Enter password: jedi*knight Welcome to the forum Enter one of the following commands: CRT, MSG, DLT, EDT, LST, RDT, UPD, DWN, RMV, XIT, SHT: (extra space added before user's response) LST No threads to list Enter one of the following commands: CRT, MSG, DLT, EDT, LST, RDT, UP RMV, XIT, SHT: (extra space added before user's response) CRT 3331 Thread 3331 exists Enter one of the following commands: CRT, MSG, DLT, EDT, LST, RDT, UPD, DWN, RMV, XIT, SHT: CRT 9331 Thread 9331 created Enter one of the following commands: CRT, MSG, DLT, EDT, LST, RDT, UPD, DWN, RMV, XIT, SHT: MSG 3331 Networks Rocks! Message posted to 3331 thread Enter one of the following commands: CRT, MSG, DLT, EDT, LST, RDT, UPD, DWN, RMV, XIT, SHT: (extra space added before user's response)	>java Client 127.0.0.1 6000 Enter username: Yoda Yoda has already logged in Enter username: R2D2 Enter one of the following commands: CRT, MSG, DLT, EDT, LST, RDT, UPD, DWN, RMV, XIT, SHT: (extra space added before user's response) CRT 3331 Thread 3331 Enter one of the following commands: CRT, MSG, DLT, EDT, LST, RDT, UPD, DWN, RMV, XIT, SHT: (extra space added before user's response) MSG 3331 Yes it does Message posted to 3331 thread Enter one of the following commands: CRT, MSG, DLT, EDT,	>java Server 6000 destroyforum Waiting for clients Client connected Yoda successful login Client connected Yoda has already logged in R2D2 successful login Yoda issued LST command issued CRT 331 created Yoda issued CRT command Thread 3331 exists Yoda issued CRT command Thread 9331 created Yoda issued MSG command Message posted to 3331 thread R2D2 issued MSG command Message posted to 3331 thread

	LST, RDT, UPD, DWN, RMV, XIT, SHT: RDT 3331 1 Yoda: Networks Rocks! 2 R2D2: Yes it does Enter one of the following commands: CRT, MSG, DLT, EDT, LST, RDT, UPD, DWN, RMV, XIT, SHT:	R2D2 issued RDT command Thread 3331 read
UPD 9331 test1.exe test1.exe uploaded to 9331 thread Enter one of the foll commands: CRT, MSG, DLT EDT, LST, RDT, UPD, DWN RMV, XIT, SHT: (extra space added before user's response)	(extra space added before user's response)	Yoda issued UPD command Yoda uploaded file 331
RDT 9331 Yoda uploaded t R2D2 uploaded t Enter one of th commands: CRT, MSG, DLT, EDT, LST, RDT, UPD, DWN, RMV, XIT, SHT: (extra space added before user's response)	9331 thread Enter one of the following commands: CRT, MSG, DLT, EDT, LST, RDT, UPD, DWN, RMV, XIT, SHT: test1.exe su downloaded Enter one of the following commands: CRT, MSG, DLT, EDT, LST, RDT, UPD, DWN, RMV, XIT, SHT:	sued UPD command R2D2 uploaded file test2.exe to 9331 thread sued RDT 9331 read N e downloaded ead 9331
EDT 3331 2 This assignment rocks The message belongs to another user and cannot be edited Enter one of the following commands: CRT, MSG, DLT, EDT, LST, RDT, UPD, DWN, RMV, XIT, SHT: MSG 3331 This assignment rocks Message posted to 3331 thread Enter one of the following commands: CRT, MSG, DLT, EDT, LST, RDT, UPD, DWN, RMV, XIT, SHT:	(extra space added before user's response) RDT 3331 1 Yoda: Networks Rocks!	Yoda issued EDT command Message cannot be edited Yoda issued MSG command Message posted to 3331 thread R2D2 issued RDT command

(extra space added before user's response) DLT 3331 2 The message belongs to another user and cannot be edited Enter one of the following commands: CRT, MSG, DLT, EDT, LST, RDT, UPD, DWN, RMV, XIT, SHT: DLT 33 The message has been deleted Enter one of the following commands: CRT, MSG, DLT, EDT, LST, RDT, UPD, DWN, RMV, XIT, SHT: (extra space added before user's response SHT destroyforum Goodbye. Server shutting down >	2 R2D2: Yes it does 3 Yoda: This assignment rocks Enter one of the following commands: CRT, MSG, DLT, EDT, LST, RDT, UPD, DWN, RMV, XIT, SHT: (extra space added before user's RDT 3331 2 R2D2: Yes it does 2 Yoda: This LST, RDT, UPD, RMV, XIT, SHT: 3331 Thread cannot be removed Enter one of the following commands: CRT, MSG, DLT, EDT, LST, RDT, UPD, DWN, RMV, XIT, SHT: RMV 3331 Thread removed Enter one of the following commands: CRT, MSG, DLT, EDT, LST, RDT, UPD, DWN, RMV, XIT, SHT: Goodbye. Server shutting down >	Thread 3331 read Yoda issued DLT command Message cannot be deleted n deleted R2D2 issued RDT command Thread 3331 read V 331 cannot be removed R2D2 issued RMV command Thread 3331 removed Obi-wan issued SHT command Server shutting down >