National University of Singapore

School of Computing

CS2105 **Assignment 1** (8 Marks) Sem 2 AY21/22

Submission Deadline

20th Feb (Sunday) 11:59 pm. 2 marks penalty will be imposed on late submission (Late submission refers to submission or re-submission after the deadline). The submission folder will be closed on **27th Feb (Sunday) 11:59 pm**.

Objectives

In this assignment, you will implement a client which would hack into a server that communicates using an HTTP-like protocol. After completing this assignment, you should

- be able to implement a simple TCP-based client, and
- have a good understanding of communication protocols.
- have a good understanding of FSM.

Group Work

All the work in this assignment should be done individually. However, if you find the assignment too difficult,

- you are allowed to form a group with another student
- maximum two students per group.
- Group submission is subject to **2 marks penalty** for each student.

Under no circumstances should you solve it in a group and then submit it as an individual solution. This is considered plagiarism. There will be no acceptance for excuses such as forgetting to declare as group submission. Please refer to the "Special Instructions for Group Submission" and "Plagiarism Warning" on page 3 for more details.

Grading

We will test and grade your program on the `sunfire` server. Please make sure that your program run properly on `sunfire`. Moreover, you are allowed to use libraries installed in public folders of `sunfire` (e.g. `/usr/lib`) only.

We accept submission of Python 3 (in particular, 3.7), Java, or C/C++ program, and we recommend that you use **Python 3** for your assignments. Programming languages other than Python 3, Java, and C/C++ are not allowed. For Python 3, we use the `python3` program installed in folder `/usr/local/Python-3.7/bin` on `sunfire` for grading. If you use Java or C/C++, we will compile and run your program for grading using the default compilers on `sunfire` (`java 9.0.4` installed in `/usr/local/java/jdk/bin`, or `gcc 4.8.4` installed in `/usr/local/gcc-4.8/bin`). The grading script infers your programming language from the file extension name (.py, .java, .c). Therefore, please ensure your files have the correct extension names. For a Java program, the class name should be consistent with the source file name, and please implement the static `main()` method so that the program can be executed as a standalone process after compilation. We will **deduct 1 mark** for every type of failure to follow instructions (e.g. wrong program name).

Note that for **Java** programs, name your main class as Hacker and hence source file name Hacker.java during development. When you want to test your program using the provided script on Sunfire, or to make submission, rename your file name according to program submission and group submission section **while keeping the class name as Hacker**. Grading script will rename your file accordingly during compilation.

We will grade your program based on its correctness only. A grading script will be used to test your program and no manual grading will be provided.

Testing Your Program

To test your program, please use your SoC UNIX ID and password to log on to `sunfire` as instructed on Assignment 0 paper. To make the `python3` alias permanently and thus avoid typing the `alias` command every time you login, you can run the following command **once** to store the shortcut into the configuration file:

```
echo alias python3=/usr/local/Python-3.7/bin/python3 >> ~/.bash_profile
```

- Your program should receive one **command-line argument** which is the student_key as the following command shows:

  ```
  python3 Hacker-A0165432X.py <student_key>
  ```

- The 6 digit `<student_key>` has already been mailed to you.
- Note that your program should not read from stdin. Your program can print anything to stdout or stderr, and our test script will silently ignore them.
- We also release a set of grading scripts to you under the **test** folder. There is no hidden test cases during grading. However, passing all the test cases does not guarantee that you will get full marks.
- To use the grading script, please upload your program along with the `test` folder given in the package to `sunfire`. Make sure that your program and the `test` folder are in the same directory. Then, you can run the following command to test your server program:

  ```
  bash test/Hacker.sh <student_key>
  ```

- If you ever encounter this error: **tput: unknown terminal "xterm-256color"** when testing your program using script provided, run the command:

  ```
  export TERM=xterm
  ```

  once after you log in and before you run Hacker.sh.

All of you will be connection to a single server, hence start the assignment early to avoid "congestion" during last few days.

Program Submission

For individual submission, please name your single source file as `Hacker-<Matric number>.py` and submit it to the `Assignment_1` folder of LumiNUS Files.

Here, `<Matric number>` is your matriculation number which starts with letter A. An example file name would be `Hacker-A0165432X.py`. If you use Java, C, or C++ to implement the web server, please use `.java`, `.c`, or `.cpp` respectively as the extension name. Note that file names are **case-sensitive** on `sunfire`.

You are not allowed to post your solutions to any publicly accessible site on the Internet.

Special Instructions for Group Submission

For group submission, please include matriculation numbers of both students in the file name, i.e. `Hacker-<Matric number 1>-<Matric number 2>.py`. Submit it to the same `Assignment_1_student_submission` folder. An example file name would be `Hacker-A0165432X-A0123456Y.py`. For each group, there should be one designated member who submits the file, to avoid problems caused by multiple branches within a group. **Do not change the designated submitter!** If the group needs to upload a new version, it should be done by the same designated submitter as well.

Plagiarism Warning

You are free to discuss this assignment with your friends. However, you should refrain from sharing your program, program fragments, or detailed algorithms with others. If you want to solve this assignment in a group, please do so and **declare it as group work**.

We employ zero-tolerance policy against plagiarism. If a suspicious case is found, student would be asked to explain his/her code to the evaluator in face. Confirmed breach may result in zero mark for the assignment and further disciplinary action from the school.

Question & Answer

If you have any doubts on this assignment, please post your questions on Piazza forum before consulting the teaching team. However, the teaching team will NOT debug programs for students and we provide support for language-specific questions as a best-effort service. The intention of Q&A is to help clarify misconceptions or give you necessary directions.

FAQ

We will collate your questions here: link

The Hacker

In this assignment, you will be hacking into server `137.132.92.111` running a `TCP` server on port `4444`.

- The server has 8 files protected by different passwords
- Each password is 4 digits long (`0000-9999`)
- You need to
  - connect to the server with a handshake
  - guess the correct password
  - login
  - get the file
  - calculate the hexadecimal MD5 hash of the file
  - write the hash on the server
  - logout
- You earn 1 mark per correct hash written on the server.
- Forgot to mention; the sever really hates making friends. It will timeout a connection in 4 seconds. So you have 4 seconds to steal all the files.

The Protocol

Except for the "file contents", all messages are strings encoded in `utf-8`.

Request Messages

- These are messages sent from the client to the server
- All messages have a 5-byte "method" field, followed by a content

Method	**Content**	**Interpretation**	**Server Action/Response**
`STID_`	6-byte `<Student_key>`	Handshake	This is the first handshake message sent by the client to the server. If the `<Student_key>` is valid server responds with code `200_`. If not, the server disconnects.
`LGIN_`	4-byte `<Password>`	Login request	If one of the 8 valid passwords, the sever responds with code `201_` and gives access to the stored file. If the password is invalid, you get code `403_`
`LOUT_`	-	Logout request	If the client is already logged in, the server logs the client out of the file access and responds with code `202_`. Now the client is free to initiate a new login.
`GET__`	-	Request to get the file data in raw binary format	If the client is already logged in, the server will respond with code `100_`, followed by the file content (to be described later).
`PUT__`	32-byte hexadecimal hash of the corresponding file	Request to write the "hash" corresponding to the file content	The server would verify the correctness of the hash. If correct, the server responds with code `203_` and `404_` if incorrect.
`BYE__`	-	Final message, goodbye	Connection closed

Table 1: Request Message

Code	Interpretation
`100_`	File data
`200_`	Handshake successful
`201_`	Login successful
`202_`	Logout successful
`203_`	Hash Matched
`401_`	Invalid `Student_Key` Handshake failure
`402_`	Invalid Operation, client request in violation of the current server state.
`403_`	Invalid Password
`404_`	Invalid Hash
`405_`	Permission Denied, the client tried to get a file without login.
`406_`	Invalid Request from the client. The Method in the request message is invalid.

Table 2: Response Message

Response Messages

- These are messages sent from the server to the client
- All messages have a 4-byte "code" field
- Response code `100_` corresponds to the file data. It is followed by the file content in the format `<length>_<data>`.
  - e.g `100_5_ABCDE`
  - Note, data content is not in string format. It is the binary file content.

The Server

The best way to understand the server is through the FSM.

Common Errors/Issues

- The request codes are case-sensitive
- To generate the MD5 hash you may use `str(hashlib.md5(data).hexdigest())`
- It is necessary to detect disconnection events reliably. If the `bytes` object returned by `recv()` is of zero length, then no more data could be `recv()`'ed from the connection.
- Like in the case of Assignment 0, ensure that you handle header and the data separately

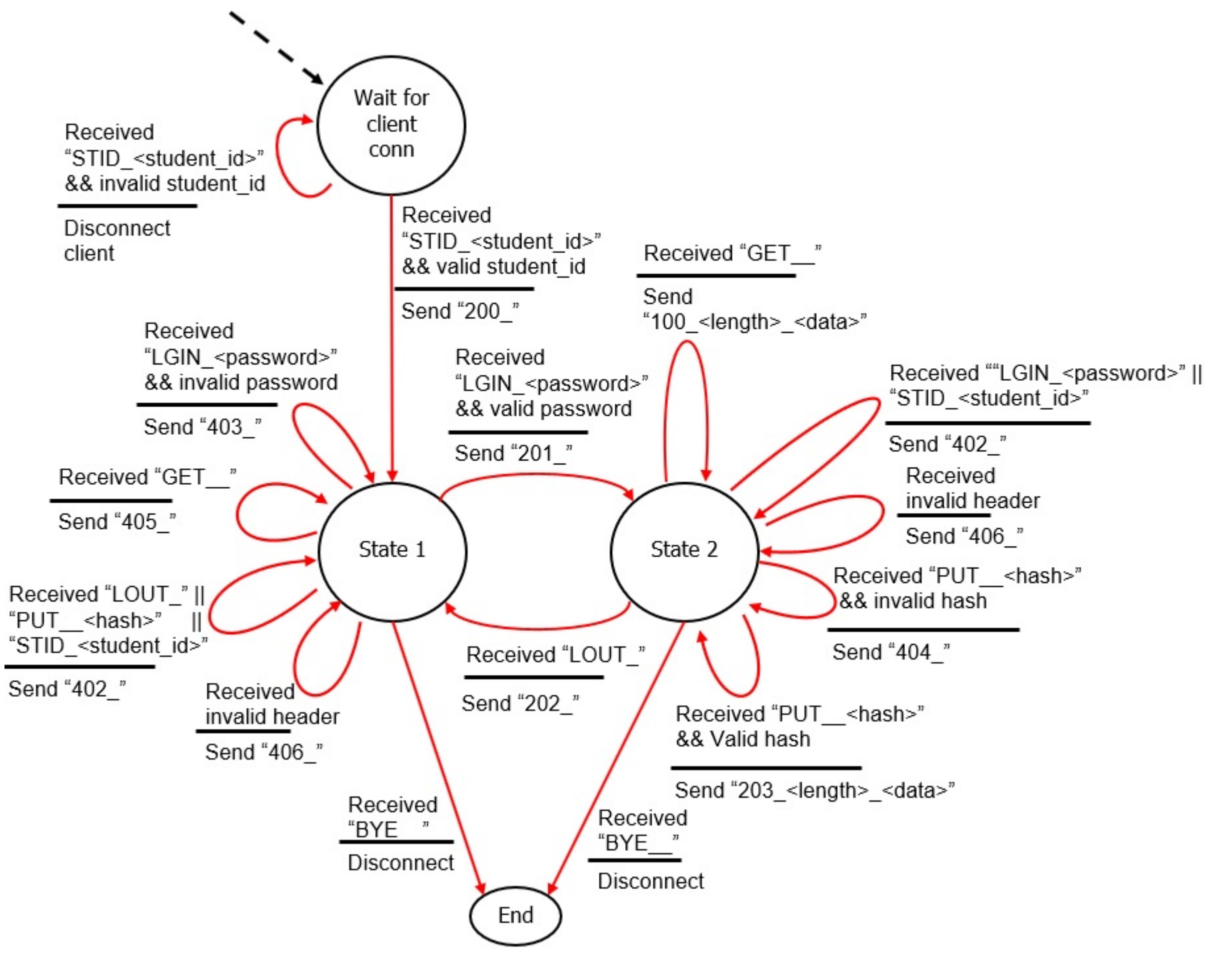
Wait for client conn
Received "STID_<student_id>" && invalid student_id
Disconnect client
Received "STID_<student_id>" && valid student_id
Send "200_"
Received "GET__"
Send "100_<length>_<data>"
Received "LGIN_<password>" && invalid password
Send "403_"
Received "LGIN_<password>" && valid password
Send "201_"
Received ""LGIN_<password>" || "STID_<student_id>"
Send "402_"
Received "GET__"
Send "405_"
Received invalid header
Send "406_"
State 1
State 2
Received "LOUT_" || "PUT__<hash>" || "STID_<student_id>"
Send "402_"
Received "PUT__<hash>" && invalid hash
Send "404_"
Received "LOUT_"
Send "202_"
Received invalid header
Send "406_"
Received "PUT__<hash>" && Valid hash
Send "203_<length>_<data>"
Received "BYE__"
Disconnect
Received "BYE__"
Disconnect
End

Figure 1: Server FSM.

- Like in the case of Assignment 0, data is to be treated as raw bytes.
- Print your debug messages to `stdout`, we have redirected `stderr` to a temporary file, hence you will not see the printed message.
- There is a low possibility that the Server gets overloadedt. So start the assignment early to avoid "congestion" during last few days.