CSC0056 Data Communication, Homework 1

Department of Computer Science and Information Engineering
National Taiwan Normal University

Sep. 17, 2024

Submit your work before the deadline shown on the Moodle module. Put all your work into one single zip file and name it after your student ID. Follow the instruction carefully; when in doubt, post your questions on our Moodle forum. This is an EMI course, and therefore you should use English whenever applicable.

Contents

1 Written assignment (80 points)

Write your answer to the following questions in a PDF file and name it *hw1-part1.pdf*. Label your answer clearly.

1.1 Model, design, and implementation (10 points)

In Weeks 1 and 2, we've discussed the differences between model, design, and implementation. Now, consider a general problem of finding the latest file in a

folder. Label each of the following six statements with either M (a description of model), D (a description of design), or I (a description of implementation). Associate each statement with exactly one label. Note that multiple statements can be in the same category, and there could be design variations and implementation variations:

Statemen-1 Each file, when created, will be associated with a time stamp representing the file creation time. Each file will have only one time stamp associated with it. And each file will be in only one folder. Copies of the file are considered as different files.

Statemen-2 To find the latest file in a folder, given multiple files and the time stamp of each, we scan this list of time stamp values and pick the smallest one. Get the file name corresponding to this smallest time stamp value.

Statement-3 To find the latest file in a folder, we maintain a two-column table, where the first column is a list of distinct folder names and the second column is a list of file names. Every time we created a file, we update the file name entry of the corresponding folder in this table. Whenever we need to find the name of the latest file, we take a look at file name entry of our target folder in the table.

Statement-4 A piece of C program that calls an API provided by our operating system to get the time stamp information of each file and do the scanning and printing tasks as described in Statement-2.

Statement-5 An Excel file for us to update the records as those described in Statement-3.

Statement-6 An operating system that automatically keeps the records as those described in Statement-3 and provides a button in its file browser window for us to click and get the name of the latest file in the current folder.

1.2 The pub/sub model (30 points)

1. (10 points) In the pub/sub model, does a publisher need to know the identity of at least one of its subscriber? Why or why not?

2. (10 points) For the following statement, if you think what it describes is true, write `T`; otherwise, write `F` and explain.

   Statement: In the scope of the pub/sub model, a publisher may directly communicate with a subscriber without the help of a broker.

3. (10 points) In a typical design under the client-server model, a server can return its service result back to the client who requested it, without prior knowledge of the client's network address; in other words, the server does

not need to know the network address of the client before that client has connected to it. This is made possible by having the client provide its network address along with its request when connecting to the server. This is great saving to the server because the amount of network address information it needs to keep depends on the number of the current connections (i.e., the working set), not on the number of all potential connections. Now, in the scope of the pub/sub model, if we are to have a design that can achieve the same effect, what would you do? Specifically, use the pub/sub terminology (topic and message) and describe your design that meets the following requirements:

(a) There is one machine running a database (called machine M) and multiple machines running user programs. Each user program will have a remote access to the database.

(b) Each machine running a user program knows the network address of machine M.

(c) Machine M does not need to know in advance how many machines will connect to it, nor does it need to know the network address nor any other identity associated with each machine.

(d) Machine M has a public topic T that is known by every machine running a user program. And the database on machine M works as a subscriber of topic T.

(e) Each user program accesses the database by working as a publisher, publishing a message with topic T. Part of the body of the message includes the detail of the database access request.

(f) The database, when returning the requested data back to each user, works as a publisher. And each user now works as a subscriber to receive the data from the database.

(g) For privacy, no user shall receive a result that has nothing to do with its original request.

Now it is your job to figure out a design using topics and messages to meet the above requirements. One thing to consider: how does the database know which topic to use when publishing a result to the specific user? Using a global topic will not work because it will violate requirement (g).

1.3 The MQTT protocol (40 points)

1. (20 points) For each of the following statements, if you think what it describes is true, write `T`; otherwise, write `F` and explain.

   Statement-1 (10 points) In MQTT, the message is encapsulated in the payload of the control packet PUBLISH.

Statement-2 (10 points) In MQTT, the messaging latency using QoS 1 will be approximately double of that using QoS 0.

2. (20 points) Answer the following question in your own words, and then compare it with the answer from ChatGPT (it is sufficient to use the free version):

   Question: Study the QoS 2 semantics described in the MQTT v5 specification, and in that context explain the meaning of packet identifier and the purpose of control packet PUBREL.

   (a) Give your own answer here first.

   (b) Copy and paste the answer you got from ChatGPT.

   (c) Compare the answer of yours and that of ChatGPT and write down your findings: in which ways does your work looks better, and in which ways does the ChatGPT work looks better? Give your qualitative assessment.

2 Software development assignment (20 points)

In this assignment we will do some simple exercise to get ourselves a bit more familiar with both the use of Mosquitto and the use of Git.

2.1 Eavesdropping a Mosquitto public broker (10 points)

For testing purpose, the Eclipse Mosquitto community has a broker open for public access, at here: https://test.mosquitto.org/

Have you ever wondered what somebody else on earth is using MQTT for? The topic "#" is a wildcard for a subscriber to subscribe to *all* topics that are publishing to our target broker. In this assignment, you are asked to use that to get a copy of every message published to the public broker.

Read through the information on https://test.mosquitto.org/ very carefully, and then use your *mosquitto_sub* program to eavesdrop all messages through port 1884 of that broker. Use *-v* option to show the topic name. Take a screenshot of what you got (i.e., the topics and the content of those messages), as an evidence of what you did. Save your proof as an image file.

2.2 Modifying the `mosquitto_sub` for message counting (10 points)

Following Question 2.1, now we are going to make a simply change to our subscriber to show the number of messages we've received from a broker. Make the following change in *sub_client.c* in the folder *./client* of our codebase.

Add a global int variable named `total_count`. Then find the line where the code calls function *print_message(...)*. Comment out that statement and right after it add the following line of code:

```
printf("%d ", total_count++);
```

After that, type 'make' at the client folder and re-try the above eavesdropping. This time it should show an accumulation of the number of messages received, instead of the content of each message.

Now use the following to create a patch of your version of code, which will take record of the changes you've made to the codebase:

```
git diff sub_client.c > hw1.patch
```

The resulting file *hw1.patch* is the patch. Now, we may use cat to view the content of the patch:

```
cat hw1.patch
```

Do some googling to understand the meaning of the syntax in the patch file.

This *hw1.patch* is what you should submit for this homework.

Our TA will apply your patch to the original codebase and grade your work. Make sure you cloned the repo at https://github.com/wangc86/mosquitto. This is the version we will apply your patch to.

It is good practice to first test it yourself that your patch can indeed be applied successfully. Figure 1 in the next page gives you an example of the process. The first git apply check failed because the source code is of the same version as the patch is supposed to create. So we restored the code back to the original version, and then the patch can be applied. The final diff implied that the patch did work correctly.

The purpose of this small exercise is, besides having a look at the traffic of the public broker, to have you go through a process of creating a code patch using git and to apply the created patch.

```
cw@Chao-Zenbook-2:~/courses/csc0056/mosquitto/client$ git apply --stat hw1.patch
 client/sub_client.c |    5 ++++-
 1 file changed, 4 insertions(+), 1 deletion(-)
cw@Chao-Zenbook-2:~/courses/csc0056/mosquitto/client$ git apply --check hw1.patch
error: patch failed: client/sub_client.c:47
error: client/sub_client.c: patch does not apply
cw@Chao-Zenbook-2:~/courses/csc0056/mosquitto/client$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   sub_client.c

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        hw1.patch

no changes added to commit (use "git add" and/or "git commit -a")
cw@Chao-Zenbook-2:~/courses/csc0056/mosquitto/client$ git restore sub_client.c
cw@Chao-Zenbook-2:~/courses/csc0056/mosquitto/client$ git status
On branch master
Your branch is up to date with 'origin/master'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        hw1.patch

nothing added to commit but untracked files present (use "git add" to track)
cw@Chao-Zenbook-2:~/courses/csc0056/mosquitto/client$ git apply --check hw1.patch
cw@Chao-Zenbook-2:~/courses/csc0056/mosquitto/client$ git apply hw1.patch
cw@Chao-Zenbook-2:~/courses/csc0056/mosquitto/client$ git diff sub_client.c > another.patch
cw@Chao-Zenbook-2:~/courses/csc0056/mosquitto/client$ diff hw1.patch another.patch
cw@Chao-Zenbook-2:~/courses/csc0056/mosquitto/client$
```

Figure 1: An example of applying a patch.