CSC0056 Data Communication, Homework 2

Department of Computer Science and Information Engineering
National Taiwan Normal University

Oct. 2, 2024

See Homework 1 for the submission requirement. Pull the latest version of our Mosquitto codebase from my GitHub repository. Type 'git log' to verify your code version. Figure 1 shows an example.

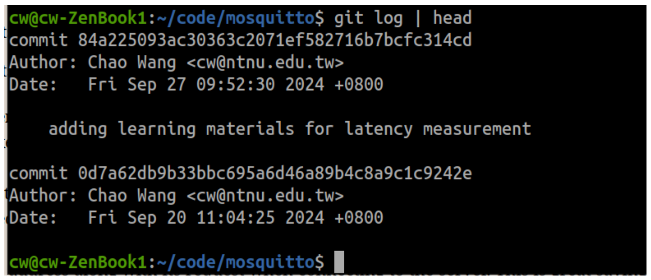
Contents

1 Written assignment (40 points)

Write your answer to the following questions in a PDF file and name it *hw2-part1.pdf*. Label your answer clearly.

1.1 The epoll event notification facility (30 points)

1. (20 points) Does our Mosquitto version use the edge-trigger or level-trigger option of epoll? Give your reasoning and evidence.
2. (10 points) Independent from Mosquitto, use your own words to illustrate a run-time scenario where edge-trigger is better than level-trigger, in the context of using epoll.



```
cw@cw-ZenBook1:~/code/mosquitto$ git log | head
commit 84a225093ac30363c2071ef582716b7bcfc314cd
Author: Chao Wang <cw@ntnu.edu.tw>
Date:   Fri Sep 27 09:52:30 2024 +0800

    adding learning materials for latency measurement

commit 0d7a62db9b33bbc695a6d46a89b4c8a9c1c9242e
Author: Chao Wang <cw@ntnu.edu.tw>
Date:   Fri Sep 20 11:04:25 2024 +0800

cw@cw-ZenBook1:~/code/mosquitto$ 
```

Figure 1: Use the latest version of our Mosquitto codebase!

1.2 The broker-side implementation (10 points)

1. (10 points) Give your own reasoning about why the Mosquitto broker only implements a reactive server design and not those more advanced ones (review the Week-2 slides for server designs)? Think about some practical scenarios: for example, what assumptions would make implementing a reactive server more appealing than implementing one that runs the leader/followers pattern? Your reason can be both very simple and correct. A good reason needs not to be complicate.

2 Software development assignment (60 points)

In this assignment, submit in terms of *one single git patch file* and name it *hw2.patch*. This file must include all the changes you've made to the codebase, relative to my commit 84a225093ac30363c2071ef582716b7bcfc314cd. You can create your patch like this:

```
git diff 84a225093ac30363c2071ef582716b7bcfc314cd > hw2.patch
```

or just

```
git diff 84a2250 > hw2.patch
```

because git can use a prefix of a commit to locate the commit.

Unlike creating a patch for just file x ('git diff x > out'), running 'git diff y > out', where y is the commit name, will give you a patch file including all the changes since commit y, possibly touching multiple files. You should try to apply your created patch to make sure it will work.

2.1 Implementing a greeting server and its client (20 points)

In the pub/sub model, a server can be implemented as a subscriber plus a publisher. In this assignment, you are going to implement a greeting server: the server will return the greeting message for the name submitted by a client. For example, after a client sent "Chao Wang", the server will reply "Hello, Chao Wang". Your testing client should use topic "toServer" for publishing messages to the server, and it should use topic "fromServer" for subscribing replies from the server. Name your server code *greet_server.c* and your client code *greet_client.c*. Your implementation only needs to support taking message as a command-line argument.

2.2 Implementing a client callback function (20 points)

Modify the default *pub_client.c* to print a timestamp upon receiving a QoS 1 PUBACK control packet. You need to add both a callback function named *my_puback_callback* and a setup function named *mosquitto_puback_v5_callback_set*. When receiving a PUBACK from broker, your publisher should call your callback function and take a timestamp therein. Your setup function is used for establishing the link for your message-handling library to know that it should call *my_puback_callback*. Review what we've covered in Week 3 and learn from the Mosquitto implementation of the callback function and the setup function for PUBLISH: *mosquitto_publish_v5_callback_set* and *my_publish_callback*. In addition to modifying *pub_client.c,* you will also need to make changes in the Mosquitto library.

2.3 Measuring the time it takes between PUBLISH and PUBACK (20 points)

Following Section 2.2 above, now run your PUBACK-callback-enabled publisher to publish 50 messages and record the time intervals between each call for a message publication and its PUBACK return. Take your publish-timestamp right before calling *mosquitto_publish_v5*, and take your puback-timestamp right after entering *my_puback_callback*. Plot two CDF curves, one for the case of using your local broker, and the other for the case of using the public broker at test.mosquitto.org. Review what we've covered in Week 4 for CDF plotting.