PA4 Q&A

1. Is the maximum number of entries in the updateStatus table 50, or are there entries in the updateStatus table for connections that aren't concurrent?
   □ You can assume that the maximum number of entries of the table is 50.

2. When does a client connection close? Is this done when the mapper client checks out? Does the mapper client check out more than once?

□ The mapper client closes the connection after getting the response of CHECKOUT request, and then the mapper process is terminated. At the server side, you should implemented an error case that the server gets a CHECKOUT request if there is no entry for the corresponding mapper client or if the check in/out flag is set to checked-out (0).

3. Does a mapper client have only one check-in request that it sends?

□ A mapper client has only one check-in request.

Do they check in, then check out for every request they have, or do they check in, send all requests, and then check out?
   □ They check in, send all requests, and then check out.

It says that for a check-in request, if there is an existing entry in the updateStatus table, the check-in field would be changed.
   □ Yes (0 checked-out □ 1 checked-in)
   Why would a mapper process need to check-in more than once?

□ A mapper client sends CHECKIN request once. However, you should implement various error handling at the server side. We will test whether your server is handling the error cases. For example, when the server receives a CHECKIN request, there should be no entry in the table, or the check in/out field should be 0 (checked-out). The other cases are error cases.

4. Is each request an integer array of size 28?
   □ Yes

5. In Table 5, under Data Returned, is this the data that the server gives back to the client in the response, or is this the data contained in the request?
   □ This is the data that the server gives back to the client in the response.
   Does the client do anything with the data returned?
   □ No, you just need to print out the response as guided in section4.3

6. When a reducer thread is created, do we have to explicitly create a socket between it and the mapper client, or does accept() already give us the socket information for this connection?

□ The return value of accept() is the socket for the mapper client, so you can use it without having to create a new socket.

7. How do we know what Server IP to put as a command line argument for the client? In the lab, it looks like the server used 127.0.0.1 as the IP address? Should we use this again?

- You can use “ifconfig” command to figure out the IP address of the machine. When you enter “ifconfig | grep inet” in the terminal, you will see “127.0.0.1” (local host), and the other IP address (“XXX.XXX.XXX.XXX”) is the actual IP address of the machine.
- When you are running both client and server in the same machine, you can use either of them. However, when you test them in different machine, you should use the actual IP address.

8. Does "checked-in" mean that the mapper process is currently communicating to a reducer thread? Does "checked-out" mean that the mapper process isn't currently communicating to a reducer thread?
   - Yes.

9. When the client does a GET_AZLIST request, would it check that the list is updated correctly? Does it do anything with the list that it gets back? Similarly, for GET_MAPPER_UPDATES, is the client supposed to do anything with what it gets back?

- They are simple getting the current azList values and the number of updates. Clients do nothing with the return values.

10. Do mapper clients need to follow the following order (1-6) under Phase 2 when sending requests?
    - Yes

It sounds like they do, but later it says that they need to be checked-in to send requests, and then they can send the other types of requests. Thus, is the order fixed or not?

- The order is fixed. 1 through 6.

If the order of requests is set, if the mapper clients send a request outside of the order shown in Phase 2, should we be able to handle it or do error checking?

- Yes, you should handle such errors at the server side. For example, when the server receives the following requests (UPDATE_AZLIST /GET_AZLIST /GET_MAPPER_UPDATES/ GET_ALL_UPDATES), you should first figure out the type of client based on mapperID (mapper clients have their mapperID > 0, master client has mapperID == -1), and then If the client is a mapper client, the server makes sure the mapper client is already checked in based on check in/out field in the updateStatus table.

11. Will our final result be tested on multiple machines, or will the client and server be on the same local host?
    - We will test your programs in both ways.

12. In section 1 of the instructions, it states that "The code must be originally written by your group. No code from outside the course texts and slides may be used—your code cannot be copied or derived from the Web, from past offerings, other students, programmer friends, etc." I know of several people who are taking CSCI 4211 Introduction to Computer Networks concurrently with CSCI 4061. The second project for that class involved creating chatroom called "Gopherchat" with a client-server architecture in C. Can we use the helper functions made for that project in this project (ie encoding messages, encoding integers to chars and vice-versa, etc).

□ You can use the codes that YOU wrote if you want. Please comment the self-reference information in the source file. However, please read/write messages in a form of integer data type, without encoding them to chars.

For example, let's say you have request[28] and response[28]. You can simply call the following functions for sending the request, and receiving the response.

```
write(socket_file_descriptor, request, sizeof(int) * 28)
```

```
read(socket_file_descriptor, response, sizeof(int) * 28)
```

13. The provided code does not compile. For example, in clinet.c, createLogFile() is defined as taking no arguments, but arguments are passed to createLogFile() on line 46. Also, it seems that the include chain from #include "../include/protocol.h" does successfully perform the include. I'm wondering if these errors are intentional, or if an old version of the example code was uploaded to Canvas by accident.

    □ Thanks for letting me know. Line 46 should be createLogFile() without having any input parameters. However, the header #include "../include/protocol.h" works. The given codes and makefiles are templates to get started. If that doen't work for you, so you can feel free to modify them. I have updated the client.zip, so please download it again.

14. Under section 4.3, various responses have "(request[#])" or "(response[#])" after a variable which fills a format specifier within a string. When sending requests and responses, is it expected that a command will be split into multiple smaller packets? Since the size of every type of message is fixed, isn't it possible to encode every necessary bit of information into one message?

    □ I am not clear about your question. Based on my understanding, you don't need to think about the packet-level handling or Little-endian / Big-endian for this assignment.

    □ As I mentioned in your first question, request[#] and response[#] are integer arrays, so you can simply access data from the integer array with an index (#).

15. For error handling, what should the client and server output when an error is encountered? For example, if client A's mapper requests mapper ID 3 but mapper ID 3 is already occupied by instance B of the client program, should client output anything while it waits for mapper ID 3 to become available?

    □ There are no further actions taken by clients based on the response from the server. Mapper clients send 6 fixed requests in the same order that stated in section 3.2 [Phase 2], and print out the responses based on the format stated in section 4.3.

16. If the client or the server unexpectedly goes offline, what should the client and server do? Is the server expected to recover from a client's Mapper process unexpectedly disconnecting from the server? Should an error message be displayed?

    □ First of all, you can assume there is no such unexpected network disconnection while executing the programs. However, you still have to handle with failures related to the connection initiation. For example, socket creation failure, bind failure, listening failure, accept failure, etc. at the server side, and socket creation failure, connection failure, etc.

at the client side. To handle with such error cases, you can simply print out error messages and terminate the program.

17. What is the LONG_RESPONSE_MSG_SIZE definition meant for in protocol.h? Are there any commands which need a 29 int response? Should that be 28 for a GET_AZLIST response?

     You are right. GET_AZLIST uses the LONG_RESPONSE_MSG_SIZE (=28) and the other requests use RESPONSE_MSG_SIZE (=3). The protocol.h has been updated. Thanks for pointing it out.