A7: Computer Networking (II)

Computer Systems 2019
Department of Computer Science
University of Copenhagen

Vivek Shah

Due: Friday, 10th of January, 23:59
Version 1 (December 13, 2019)

This is the eighth and final assignment in the course on Computer Systems at DIKU and the second on the topic of Computer Networks. We encourage pair programming, so please form groups of 2 or 3 students. Groups cannot be larger than 3, and we strongly recommend that you do not work alone.

For this assignment you will receive a mark out of 4 points; You must attain at least half of the possible points to be admitted to the exam. For details, see the *Course description* in the course page. This assignment belongs to the CN category together with A6. Resubmission is not possible.

It is important to note that only solving the theoretical part will result in 0 points. Thus, the theoretical part can improve you score, but you *have* to do socket programming.

The big deal about the Internet design was you could have an arbitrary large number of networks so that they would all work together.

— Vint Cerf

Overview

This assignment has two parts, namely a theoretical part (Section 1) and a programming part (Section 2). The theoretical part contains questions on topics that have been covered by the lectures. The programming part builds a distributed chat service employing a combination of both client-server and peer-to-peer architectures using socket programming in C. The implementation task of this assignment builds upon the client-server implementation of the previous assignment (A6). In this assignment, the programming effort relies solely on building the peer-to-peer portion of the architecture to allow peers to directly send chat messages to each other. If you did not implement A6 or do not want to build upon it, there are instructions in the programming section regarding the assumptions that you can make. More details would follow in the programming part (Section 2).

1 Theoretical Part (30%)

Each section contains a number of questions that should be answered **briefly** and **precisely**. Most of the questions can be answered within 4 sentences or less. Annotations have been added to questions that demand longer answers, or figures with a proposed answer format. Miscalculations are more likely to be accepted, if you account for your calculations in your answers.

1.1 Transport protocols

The questions relating to TCP in this section do not assume knowledge of the flow and congestion control parts of TCP.

1.1.1 TCP reliability and utilization

Part 1: Is the 3-way-handshake in TCP really needed? Can you suffice with less? Explain why or why not.

Part 2: How does TCP facilitate a *full-duplex*[1] connection?

1.1.2 Reliability vs overhead

Part 1: Considering the data transferred over a network, how does a TCP connection add overhead relative to UDP?

Part 2: Explain the TCP notion of reliable data transfer. In which way are TCP connections reliable? Are packets **always** delivered?

1.2 TCP: Principles and practice

Some of the questions below refer to one or more *Request For Comments* (RFC) memorandums posted by the *Internet Engineering Task Force* (IETF). In most cases, you can find answers to the question in the book, but we recommend that you browse through these RFCs to see how internet standards are formulated and distributed in practice.

Be careful when answering the questions below. If any ambiguity may arise, remember to specify the point of view of your answers – is it from the server or the client or both?

1.2.1 TCP headers

Part 1: The TCP segment structure is shown in figure 3.29 in K&R (or section 1.3 of RFC793[2]).

1.1. What is the purpose of the RST bit? Give an example of when a TCP stack may return a RST packet.

[1] Full speed bidirectional data transfer

[2] `https://tools.ietf.org/html/rfc793#section-3.1`

1.2. What does the sequence number and acknowledgement number headers refer to for a given connection? Is there any relation between server and client acknowledgement and sequence numbers?

1.3. What is the purpose of the window? What does a positive window size signify?

1.4. If the window size of a TCP receiver is 0, what happens then at the sender? How can the sender find out when the receiver window is not 0?

Part 2: In one of the previous questions, you were asked why the 3-way handshake was necessary. One of the practical reasons for this 3-way-handshake was the synchronization of sequence numbers. In some situations, the client can initiate its transfer of data after having received the SYN-ACK-packet. Explain when this is possible. *(Hint: What does the server need to tell the client in the SYN-ACK-packet?)*

1.2.2 Flow and Congestion Control

Part 1: Explain the type of congestion control that modern TCP implementations employ. Is it network assisted congestion control?

Part 2: Congestion control translates directly to limiting the transmission rate. How does TCP determine the transmission rate when doing congestion control?

Part 3: How does a TCP sender infer the value of the congestion window, that is, what is the basis for calculating the congestion window? *(Answer with at most 10 sentences)*

Part 4: The fast retransmit extension, as described in RFC 2581[3] allows for retransmission of packets after receiving 3 duplicate ACKs for the same segment. What does the sender need to implement in order to allow for fast retransmit? What does the receiver need to implement? If a sender supports fast retransmit and needs to do fast retransmit for a specific segment, how many ACKs should the sender receive for the previous packet?

2 Programming Part

For the programming part of this assignment, you will build the peer-to-peer portion of the distributed chat service. The distributed chat service comprises of a peer client (`peer.c`) and a chat name server (`name_server.c`). The name server contains nicks and passwords of known users. The users must use the peer client to first sign in to the name server with their password and register their addresses which users can lookup to engage in a peer to peer chat. Once

[3] `https://tools.ietf.org/html/rfc2581`

the users sign in and their identities are known, they can engage in a peer-to-peer chat with other signed in users.

In this assignment you build upon the client-server functionality that you implemented in the previous assignment (A6). You need to add functionality (1) to send chat messages to other signed in users and (2) to view chat messages received from users. The code handout for this assignment consists of `peer.c`, `name_server.c` and their corresponding `.h` files. For this assignment, we have also provided `buffer.h` and `buffer.c` which contain a simplified implementation of a queue abstraction based on linked lists. In addition, we have provided `csapp.h` and `csapp.c` files which contain the helper functions from the robust IO library and other helper functions contained in the B&OH[4] book. We have also provided a `Makefile` to make it easier for you to build the sources. *Use of the helper functions in csapp.h, csapp.c, buffer.h and buffer.c is optional. You can choose to ignore them if you wish.*

For the practical part of this assignment you are expected to hand in your C source code, tests showing validity (or invalidity) of your implementation and a document containing the answers to the questions as well as a short explanation of how you implemented your code (if your code has any shortcomings you should document them here). *You are **not** allowed to use any external libraries other than the standard C library for this assignment. You can use the helper functions that have been handed out with the assignment sources.*

Dependency with A6

Although this assignment builds upon the implementation of A6 and we strongly encourage you to build upon your previous implementation to have a working distributed chat service at the end of A7, you *are not strictly* required to have implemented A6 in order to solve this assignment. If you do not want to build upon your implementation of A6, you can assume that there is no chat name server and the static configuration of peers (IP, port, nick) can be supplied by a configuration file to each peer. The only commands you need to support are `/msg` and `/show` as elaborated in Section 2.1 and you can ignore all the other commands that were built in A6 (`/login, /logout, /lookup, and /exit`). Remember that the sample interaction outlined in Figure 1 should remain identical except for the commands that are not supported. You *should additionally ensure* that you document your assumptions and provide instructions on how to configure and run your implementation.

2.1 Peer Messaging (45%)

In this assignment, you need to augment the peer client with two extra commands: (1) to send chat messages to other users and (2) to view chat messages received from users. The client should allow the user to perform the following additional interactions:

2.1. Send message - The user should be able to send chat messages to the desired user. Non-existent nicks or users not signed in must be flagged by

[4]Computer Systems: A programmer's perspective, Randal E. Bryant and David R. O'Hallaron, Pearson, 3rd and Global Edition

appropriate errors. Only logged in users should be able to send messages to other logged in users. The syntax for the command is :

```
/msg <nick> <message>
```

2.2. Show message - The user should be able to see all the messages received from the desired user since the last invocation of show message. If no argument is provided for show message, then all new messages received from all users must be displayed. Only logged in users must be able to see messages received from other users. *You do not need to preserve messages across user log in and log out, you can discard all messages received by a user but not viewed by her if she logs out.*. Appropriate errors must be flagged. While displaying the message the nick of the user who sent the message must be pre-pended. The syntax for the command is either :

```
/show <nick>
```

or

```
/show
```

You should implement the above mentioned functionality in the file `peer.c` and `peer.h`. You are also allowed to create additional files if it helps to make your code modular. A sample interaction with the peer client program by two users bonii and michael is outlined in Figure 1.

You should ensure that after a user successfully logs into the name server, the peer program should start the server to listen for chat messages sent by other peers. Similarly, when the user logs out, the peer program should ensure that the server to listen for chat messages is terminated. When a peer client is killed or a socket dies, connected peers and the name server should detect this and remove that socket and close it. *You should ensure that the peers can handle sending and receiving chat messages to and from multiple peers at the same time.*

For this assignment, you need to devise and implement the protocol (messages and their handling) that should be performed by the peer clients. In addition, you can make any necessary changes to the protocol devised to communicate between peer clients and name server in the previous assignment if you wish.

2.2 Report (25%)

In addition to implementing the programming part, you should also answer the following questions in your report.

2.1. Describe how to compile and run your code. How did you test your implementation? (Note: Automated test cases are not a strict requirement for this assignment; detailing the testing procedure or methodology in the report is sufficient. Test your implementation by running the name server and peer client on the same machine before different machines since some networks disable certain forms of traffic that can affect your testing procedure.)

2.2. Explain the protocol that you used to implement the functionality. Here you should document the formats of messages exchanged between the peer clients and the events that followed on sending and receiving the messages. Remember to refer to any relevant protocol details devised in your previous assignment for completeness (Hint: You can be brief if you are referring to details already documented in your previous assignment).

2.3. Discuss the non-trivial parts of your implementation and your design decisions (if any). Explain *why* you wrote your code the way you did, not merely *what* it does. *Purpose is more important than mechanism.*. Remember to reflect on the shortcomings (if any) of your implementation and how they can be fixed.

Note: Remember to provide your solutions to the theory questions in the report.

Submission

The submission should contain a file `src.zip` that contains the `src` directory of the handout, with a modified `peer.c`, `peer.h`, `name_server.h`, `name_server.c` and `Makefile`. Furthermore, it should include any new files or test programs that you may have written or is needed to compile and run your code.

Alongside the `src.zip` containing your code, submit a `report.pdf`, and a `group.txt`. `group.txt` must list the KU ids of your group members, one per line, and do so using *only* characters from the following set:

$$\{\texttt{0x0A}\} \cup \{\texttt{0x30}, \texttt{0x31}, \ldots, \texttt{0x39}\} \cup \{\texttt{0x61}, \texttt{0x62}, \ldots, \texttt{0x7A}\}$$

Please make sure your submission does not contain unnecessary files, including (but not limited to) compiled object files, binaries, or auxiliary files produced by editors or operating systems.

Figure 1: Sample interaction between users michael and bonii

```
$ ./peer 192.168.1.42 4567
/login bonii secret 192.168.1.41 8181
You are now logged in.
/lookup michael
michael is online.
IP: 192.168.1.121
Port: 15003
/msg michael I have the exam questions ready, can you take a look ?
/show
No new messages from any user.
/show
michael: Yes, I will.
/show michael
michael: The exam questions look very easy, I will make it a bit harder.
/show michael
michael: Now it is really hard, you can edit it.
/msg michael I will make it easier. I want the students to really love me.
/msg michael They laugh at my stupid jokes in the lecture.
/show
michael: OK. I was just kidding.
/msg michael So was I :-). See you later. Bye.
/logout
You are now logged out.
/exit
$
```

```
$ ./peer 192.168.1.42 4567
/login michael randompw 192.168.1.121 15003
You are now logged in.
/lookup bonii
bonii is online.
IP: 192.168.1.41
Port: 8181
/show
bonii: I have the exam questions ready, can you take a look ?
/show
No new messages from any user.
/msg bonii Yes, I will.
/msg bonii The exam questions look very easy, I will make it a bit harder.
/show bonii
No new messages from bonii.
/msg bonii Now it is really hard, you can edit it.
/show bonii
bonii: I will make it easier. I want the students to really love me.
bonii: They laugh at my stupid jokes in the lecture.
/msg bonii OK. I was just kidding.
/show
bonii: So was I :-). See you later. Bye.
/msg bonii Good to know. Bye.
Unable to send message. User is not logged in anymore.
/logout
You are now logged out.
/exit
$
```