CSE3300/CSE5299: Computer Networking

Programming Assignment 1: Socket Programming

Due Date: by the end of Tuesday, October 22, 2024. Submission through HuskyCT. Full score: 100 points.

1 Purpose of Assignment

In this assignment, you will practice TCP socket programming. In the first part of the assignment, you will write a client and a single-threaded server. In the second part of the assignment, you will extend the server to a multi-threaded server. Whie you can use any programming language that you are comfortable with, I would highly encourage you to use python since the program in Python is conceptually simple and the sample code we covered in class is written in python. In addition, the sample multi-threaded server program that you are given in this assignment is in Python (see Section 3).

2 Basic Assignment

In the basic assignment, you will implement a client that queries for English words that are stored in a word list at a server. The program needs to be done using TCP socket programming since we would like reliability that is guaranteed by TCP.

The client sends a wildcard query that you enter through the keyboard to the server. The only type of wildcard you need to support is the ones with one or multiple questions marks. Specifically, a query can be 'a?t', where '?' means an arbitrary letter. When the client sends such a query to the server, it means to get all the words that contain three letters *as partial string*, i.e., containing 'a', an arbitrary letter, and then 't'. For instance, the matching words can be 'ant', 'anticipate', 'mantis', 'rant'. The server has a text file (`wordlist.txt`, enclosed in this assignment in HuskyCT), which has a list of English words. The server's job is waiting for queries from clients to come in. Once receiving a query, the server looks up the wordlist and returns all the matching words back to the client. The client will then display the number of matching words, each matching word, and then terminate. The server only needs to serve one client at one time. The client only needs to send one query. Note that the queries have either one or multiple '?'.

Application Protocol. Design and describe your application level protocol, which specifies the format of the messages (queries and responses) and actions that the server and client need to take. Your implementation should follow the application level protocol you design. It's a good idea to consider special cases, e.g., no queries found. You can eve design action code, e.g., "200 OK" or "404 not found" as in HTTP. You can use any of the protocols (e.g., HTTP) that we covered as an example. The same application level protocol can be used for both the basic assignment and the multi-threaded version. In your application level protocol, the request from the client must contain a **command**, and the response from the server must contain a **status code** and also specify the number of matching words.

3 Multi-threaded Server

Now leave a copy of your basic server and client programs aside (please submit the code of *both* the basic and multi-threaded version for this assignment). Our next step is to extend both the server and the client. Specifically, (i) you are going to make your server to be multi-threaded so that it can serve multiple clients simultaneously, and (ii) you will also extend your client to allow it to send multiple queries; each query is input by you through the keyboard. When you enter "quit", the client will terminate.

To help you write the multi-threaded server program, we provide a sample source code (called `thread-server.py`) to you. The code is taken from the book "Programming Python" (by Mark Lutz). It has detailed documentation in the source code. In particular, read `dispatcher` function, which delegates a client to a newly created thread that runs `handleCient` function. As a result, the server can serve multiple clients simultaneously.

Test your code and make sure the server can serve multiple players simultaneously. To do this, you can (i) run the server in one terminal, (ii) open another terminal to run one client (do not terminate the client), and (iii) open yet another terminal to run the second client.

4 What to Turn In

When you hand in your programming assignment, please include the following (please submit to HuskyCT):

- A program listing containing in-line documentation. Uncommented code will be heavily penalized. Submit *both* the versions for the basic server and client, as well as the multi-threaded version of the server and client.
- A separate (typed) document describing the overall program design, a verbal description of how it works", application-level protocol between the server and client, and design tradeoffs considered and made. Also describe possible improvements and extensions to your program (and sketch how they might be made). You *only* need to provide the design document for the multi-threaded version of the server and client. The format of the description file should be as follows (If your turn-in does not follow the above format, you'll get 10 points off automatically):
  - Description: describe the overall program design, "how it works", and your application protocol. You can use the application protocols that we have learned (e.g., HTTP, DNS, SMTP) as examples.
  - Tradeoffs: discuss the design tradeoffs you considered and made.
  - Extensions: describe possible improvements and extensions to your program, and describe briefly how to achieve them.
  - Test cases: describe the test cases that you ran to convince yourself (and us) that it is indeed correct. Also describe any cases for which your program is known not to work correctly. Please use screenshots to show the results.

    Here are some example client queries:

```
Client query:  ?????????
Client query:  ?
Client query:  (a)
Client query:  -?-
```

You must include the above testing cases. Feel free to include more.

5 Grading policy (Total: 100 points)

- Program Listing
  works correctly: 50 points (shown by testing results)
  in-line documentation: 10 points
  quality of design: 10 points
- Design Document
  description: 5 points
  tradeoffs discussion: 5 points
  extensions discussion: 5 points
- Thoroughness of test cases: 15 points

Notes: A full 10 points for quality of design will only be given to well-designed, thorough programs. A correctly working, documented and tested program will not necessarily receive these 10 points.

6 For CSE5299 students

This is mandatory for CSE5299 students. So far, we have been using TCP socket programming. Now repeat the assignment of basic client and server using UDP socket programming. You can run client and server on the same host (i.e., use 127.0.0.1 as server IP address), and assume that the channel is reliable so you don't need to worry about reliable data transfer.

Once you are comfortable with the basic client and server programs using UDP, change your server to be multi-threaded. Note that you may need to worry about risk conditions since multiple threads can be reading from the same socket simultaneously. You can use any synchronization mechanisms that you are comfortable (e.g., mutex lock) to resolve this issue.

Submit your code and relevant document, including

- basic server and basic client code using UDP
- multi-threaded server and client using UDP
- design document for multi-threaded version with UDP (only what differs from the design doc for TCP)

The same grading policy as described in Section 5 applies here.

7 A Few Words About Borrowing Code...

As many of you probably already know, there is a wealth of sample socket programming code out on the Web and in books that you can use in your projects. Often learning from sample codes is the best way to learn. I have no problem with your borrowing code as long as you follow some guidelines:

- You may not borrow code written by a fellow student for the same project.
- You must acknowledge the source of any borrowed code. This means providing a reference to a book or URL where the code appears (you dont need to provide reference for the code that was given to you).
- In your design document, you must identify which portions of your code were borrowed and which were written by yourself. This means explaining any modifications and extension you made to the code you borrowed. You should also use comments in your source code to distinguish borrowed code from code you wrote yourself.
- You should only use code that is freely available in the public domain.