Project 1: Socket Programming

Due on 10-25-2025 Saturday Midnight

Please mark in your submission whether you are in CSE5299 or CSE3300 in your submitted file.

100 with up to 20 extra points for CSE3300 students (will be treated with 100 as full score in final grade calculation); 120 for CSE5299 students (will be treated with 120 as full score in final grade calculation).

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. While 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, starting with 'a', ending with 't' and having an arbitrary letter in the middle.

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 words 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 '?'. In your testing, it is OK to limit your test cases to contain up to 3 '?' in the query.

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 design action code, e.g., "200 OK" or "404 not found" as in HTTP. 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.

Objective. Your response should include only words that exactly match the given pattern in both length and letter placement.

- **Example:** If the query is a?t, then your response should include all 3-letter words that start with "a" and end with "t" (with any single character in the middle).

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.

3 What to Turn In

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

- A program containing in-line documentation. The 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.

4 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: 5points
  extensions discussion: 5 points
- Thoroughness of test cases: 15 points

Please use screenshots to show the results.

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.

5 For CSE5299 students; Extra credits for CSE3300 students (20 points)

This is mandatory for CSE5299 students and will give up to 20 extra points for CSE3300 students. Please mark in your submission whether you are in CSE5299 or CSE3300 in your submission. If you did not mark, your submission will be treated as a CSE3300 submission.

Continue your work on the multi-threaded server and multiple query client. In this part,

1) your server is expected to support queries that contain **any** number of wildcard(s) using partial match and print out the **total** number of words each response contains. In other words, the returned results should be '*a?t*' for a?t, where '*' is a wildcard match that can be an arbitrarily long string (i.e., length ≥ 0). Therefore, the matching words are 'ant', 'anticipate', 'mantis', 'rant', and so on. (e.g., one matching word is 'ant').

2) your client is expected to print out the **entire** response as well as the **total** number of words contained in the response from the server before making another query.

- e.g. if the client sends just one wildcard (question mark), your client should receive and print out the entire wordlist as well as the total number of words before you make another query.

Here are some example client queries:

Client query: ??????????

Client query: ?
Client query: ?(a)

Client query: -?-

You must include the above testing cases. The number of matches are: 24071, 69903, 414, and 13, respectively. Feel free to include more.

Grading Policy:

Your program with in-line documentation: 5 points

Thoroughness of your test cases (screenshots for both client side and server side): 5 points

Server prints out the correct number of words contained in each response for all test cases: 5 points.

Client receives the entire response and prints out the correct number of words for all test cases: 5 points

6 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 don't 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 extensions 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.