# CMCS 481 - Programming Assignment: Socket Programming

## Due Date: no later than 11:59pm on Sunday, May 4, 2025

### Goal of the project

In this assignment, you will build a simple client-server system, where you use the client to chat with a "dictionary" server. The protocol between the client and server is as follows.

- The server is first started on a dedicated port. Any client wishing to connect with the server will need to know the server's IP address and port number. The server will need to have access to a dictionary file.
- The client program is started (with the server IP and port number being provided on the command line).
- The client connects to the server and then asks the user for input. The user enters a single "word". The user's input (a single "word") is sent to the server via the connected socket.
- The server reads the user's input from the client socket, retrieves the definition, and sends the result back to the client.
- The client should display the server's reply to the user, and prompt the user for the next word, until the user terminates the client program with Ctrl+C.

You are responsible for writing the client and **three (3)** versions of the server:

- "Server1" will be a single process server that can handle only one client at a time. If a second client tries to chat with the server while one client's session is already in progress, the second client's socket operations should see an error.
- "Server2" will be a multi-process server that will **"fork"** a process for every new client it receives. Multiple clients should be able to simultaneously chat with the server.
- "Server3" will be a single process server that uses the "**select**" system call to handle multiple clients. Again, much like server2, server3 will also be able to handle multiple clients concurrently.

Several good tutorials and useful documentation on socket programming and designing servers for concurrent clients (using both fork and select) are available online. Please make use of these resources to learn the intricacies of socket programming on your own during this assignment.

**You are required to write your code for this assignment in C/C++ or Python.** Please talk to the course TA if you have a compelling reason to use any other language.

---

### The client

The server should be run on port 5000 in another terminal. The client program is then given the server IP (localhost 127.0.0.1 in this case) and port (5000) as command line inputs. See example illustrations below:

```
$ gcc client.c -o client

$ ./client 127.0.0.1 5000

Connected to server

Please enter the word that you need defined: word 1

Server replied: definition of word 1

Please enter the word that you need defined: word 2

Server replied: definition of word 2

…

…
```

In parallel, here is how the output at the server looks like this (you may choose to print more or less debug information). Note that the server's output shown below is only for illustration, and we will not grade you based on your server's debug output. We will primarily grade you based on whether your server returns correct responses to the clients or not.

```
$ gcc server1.c -o server1

$ ./server1 5000

Connected with client socket number 4

Client socket 4 sent message:

Sending reply:

Client socket 4 sent message:

Sending reply:

…

…
```

---

**The servers**

**Part1: Single process server**

First, you will write a simple server in a file called "server1.c". The server1 program should take the port number from the command line and start a listening socket on that command line. Whenever a client request arrives on that socket, the server should accept the connection, read the client's request, and return the result. After replying to one message, the server should then wait to read the next message from the client, for as long as the client wishes to chat. Once the client is terminated (socket read fails), the server should go back to waiting for another client. The server should terminate on Ctrl+C.

Your simple server1 should NOT handle multiple clients concurrently. That is, when server1 is engaged with a client, another client that tries to chat with the same server must see an error message. However, the second client should succeed if the first client has terminated.

**Multi-process server**

**Note:** The behavior of server1 and server2, with respect to handling multiple clients correctly is the same. Both servers should behave like any real server. They should be able to handle any number of clients concurrently. Servers 1 and 2 should work correctly as clients come and go. Servers 1 and 2 should always keep running (until terminated with Ctrl+C) and should not quit for any other reason. If you are in doubt about any functionality of the server, think of what a real server would do, and use that as a guide for your implementation.

**Part2: Concurrent server with "fork ( )"**

For part 2, you will write server2.c to be able to handle multiple clients simultaneously by forking a separate process for each client.

**Part 3: Concurrent server with "select ( )"**

Now you will write server3.c to handle multiple clients using the "select" system call. The difference from part 2 is that you will not do a fork () to create a new process for every client, but you will handle all the client sockets in a single process.

---

**The Dictionary files**

For this project, your servers will need access to a dictionary file. The servers will need to accept input “a single word” from the client and parse/search the dictionary file for the definition of the “word” submitted by the client. For this project, you will use *Webster's Unabridged English Dictionary*. You can access the open-source dictionary in various formats here: https://github.com/matthewreagan/WebstersEnglishDictionary. The dictionary files are .txt and .json formats. Note that there are three .json formatted files. The content is the same in all files. You can choose the version of your choice. Read the README.md at the above website, to understand the formatting structure of each .json file.

In your ReadME file (to be submitted with your project), you must specify which version works with your servers.

---

**Submission instructions**

**This is not a group project. Each student must complete and submit the project independently. Students may discuss strategies (e.g., optimization mechanisms, debugging, test cases, etc). However, each student should write his/her own code. All code will be**

**evaluated for cheating (including copying from classmates, downloading from online sources, using AI generated code, etc.)**

To submit your project, create a submission folder, where the name is a concatenation of your first and last name separated by underscore ("_"). For example, my folder name would be Dmitri_Perkins Create a tar gzipped file and submit no later than 11:59pm on Sunday, May 4, 2025.

Your submission folder must contain the following files.

- client.c, server1.c, server2.c, server3.c as described above. **Strictly follow the guidelines for the filenames.**
- A make/build script to compile your code.
- Readme file: Include detailed instructions for compiling/running your code. Include section labeled "Test Cases" that provides a comprehensive description of the various scenarios that you used to test your client and the three servers. You should describe the purpose for each test case (e.g., testing for invalid request from a client, testing concurrency, etc),

**Note:** You must document your code properly. Since we will read through your code while grading, cleanly-written and well-documented code could improve your grade, in addition to making the grader's job much easier.

**Testing (not exhaustive)**

Testing will be conducted from the client's point of view. All test cases assume that the server is running. Below are some example text cases that will be used to evaluate your code. Please note that these are only example test cases and additional test cases will be used to thoroughly test the correctness and robustness of your client and servers.

Server1

Test 1:

```
$ ./client 127.0.0.1 5000
Connected to server
Please enter the word that you need defined: word1
Server replied: definition of word1
Please enter the word that you need defined: word 2
Server replied: definition of word 2
```

Test 2: Concurrent Clients

```
$ ./client 127.0.0.1 5000
Connected to server
Please enter the word that you need defined:
word1
```

```
$ ./client 127.0.0.1 5000
connect() failed: Connection timed out
```

```
Server replied: definition of word1
Please enter the word that you need defined:
word 2
Server replied: definition of word 2
```

Server2

Test 1:

```
$ ./client 127.0.0.1 5000
Connected to server
Please enter the word that you need defined: word1
Server replied: definition of word1
Please enter the word that you need defined: word 2
Server replied: definition of word 2
```

Test 2: Concurrent Clients

```
$ ./client 127.0.0.1 5000
Connected to server
Please enter the word that you need defined:
word 2
Server replied: definition of word 2
```

```
$ ./client 127.0.0.1 5000
Connected to server
Please enter the word that you need defined:
word1
Server replied: definition of word1
```

Server3

Test 1:

```
$ ./client 127.0.0.1 5000
Connected to server
Please enter the word that you need defined: word1
Server replied: definition of word1
Please enter the word that you need defined: word 2
Server replied: definition of word 2
```

Test 2: Concurrent Clients

```
$ ./client 127.0.0.1 5000
Connected to server
Please enter the word that you need defined:
word 2
Server replied: definition of word 2
```

```
$ ./client 127.0.0.1 5000
Connected to server
Please enter the word that you need defined:
word1
Server replied: definition of word1
```

**Grading**

Below is the grading scheme for this assignment. This assignment carries 25 points (scaled to 10% of grade).

- 25 points for code inspection (readability, documentation, correctly following the specifications).
- 50 points for correct behavior and operations – as determined by the various test cases (including but not limited to the examples above) that we will use to evaluate your client and three servers.
- 25 points for the written project questions (to be shared in a separate document).