# NWEN 241 Assignment 5

Network Server Program

Release Date: **20 May 2019**

Submission Deadline: **9 June 2019, 23:59**

In this assignment, you will implement a network server program given a set of specifications and guidelines which are outlined in the tasks. Unlike the past four assignments, code framework and test codes will not be provided. You are free to design the program in any way you want and implement it in either pure C or C++.

Full marks is 100.

**Instructions and Submission Guidelines:**

- You should provide appropriate comments to make your source code readable. If your code does not work and there are no comments, you may lose all the marks. See the marking criteria at the end of this document for details about the marks for commenting.
- You should follow a consistent coding style when writing your source code. See the marking criteria at the end of this document for details about the marks for coding style.
- Submit the required files to the Assessment System (`https://apps.ecs.vuw.ac.nz/submit/NWEN241/Programming_Assignment_5`) on or before the submission deadline.
- Late submissions (up to 48 hours from the submission deadline) will be accepted but will be penalized. No submissions will be accepted 48 hours after the submission deadline.

**Program Specifications**

Upon execution, the program should perform the following:

1. Load the CSV database file `scifi.csv` which contains the 25 greatest science fiction films of all time. You can use any data structure for holding the records in memory. Note that `scifi.csv` is provided under the `files` directory.
2. Perform required socket operations to create and bind a socket.
3. Listen for clients at TCP port `12345`. (During testing, you may need to use a different port number to avoid conflicts with other students running tests in the same server.)
4. When a client successfully establishes connection, send a message with contents `HELLO`.
5. Wait for a message from the client.
6. Perform the following based on the received message from the client:
   (a) If the message has the contents `BYE` (case-insensitive), close the connection to the client and go back to step 3.
   (b) If the message has the contents `GET` (case-insensitive), send a message back to the client containing the entire database. You may format the content in any way you want, but all the fields of every record must be printed. Go back to step 5.
   (c) If the message has the contents `GET` (case-insensitive) followed by a number (example: `GET 12`), treat the number as the row number of the record to be fetched. That is, send a message back to the client containing the record at the specified row number. You may format the content in any way you want, but all the fields of the record must be printed. If the specified row number does not exist, send back a message containing `ERROR`. Go back to step 5.

**Testing**

As you are using the Linux `socket()` and `fork()` system calls, you will need to test your program in any of the computers inside CO246. Alternatively, you can perform remote testing following the procedures in `https:`

`//ecs.victoria.ac.nz/foswiki/pub/Courses/NWEN241_2019T1/LectureSchedule/NWEN241-Week02-Lab.pdf`. For remote testing you can connect to any of the following Linux-based ECS servers:

- `barretts.ecs.vuw.ac.nz`
- `embassy.ecs.vuw.ac.nz`
- `greta-pt.ecs.vuw.ac.nz`
- `regent.ecs.vuw.ac.nz`

You do not need to write a client program to test your server program. You can use the Linux program `nc` to receive and send messages to your server program. To do this (assuming you are in CO246):

1. Open a terminal. Compile and run your program.
2. Open another terminal. Run `nc localhost 12345` to establish a connection with your server program. You may need to replace `12345` with the actual port number that you specified in your code. Once `nc` is connected to the server program: whatever you type in this terminal will be sent to the server program, and whatever is sent by the server program will shown in this terminal.

If you are performing the test remotely, you will need to open 2 ssh/putty connections to the same server machine. In one ssh/putty terminal, you compile and run your program. In the other ssh/putty terminal, you run `nc localhost 12345` to establish a connection with your server program. You may need to replace `12345` with the actual port number that you specified in your code.

**Task 1.**

Basics [40 Marks]

Design the header file(s) to be used by your program. You can use any name for your header file(s), but their extension must either be `.h` (for pure C) or `.hh` for (C++).

**Task 2.**

Completion [30 Marks]

Provide an implementation (C/C++ source files) of the server program as specified in the **Program Specifications** above. You can use any name for your source file(s), but their extension must either be `.c` (for pure C) or `.cc` for (C++).

**Task 3.**

Completion [10 Marks]

Provide a `Makefile` to automatically build your program. The Makefile should have at least 2 rules:

1. `dbserver`: This rule should compile the server program that you have implemented in Task 2.
2. `clean`: This rule should delete any compiled file (object and executable files).

Important: If you opt not to perform this task, you must provide instructions on how to compile your code. Write these instructions in a text file named `README`.

**Task 4.**

Challenge [20 Marks]

One of the drawbacks of the server program specified in the **Program Specifications** above is that when a client is connected with the server, no other client can connect to the server.

Provide an enhancement of the server program by using the `fork()` system call as follows: At step 4, when a client successfully establishes connection, fork a child process. Upon successful fork, the parent process should go back to step 3. Whereas, the child process should perform the following:

1. Send a message to the client with contents `HELLO`.
2. Wait for a message from the client.
3. Perform the following based on the received message from the client:
   (a) If the message has the contents `BYE` (case-insensitive), close the connection to the client and exit the process.

(b) If the message has the contents `GET` (case-insensitive), send a message back to the client containing the entire database. You may format the content in any way you want, but all the fields of every record must be printed. Go back to Step 2.

(c) If the message has the contents `GET` (case-insensitive) followed by a number (example: `GET 12`), treat the number as the row number of the record to be fetched. That is, send a message back to the client containing the record at the specified row number. You may format the content in any way you want, but all the fields of the record must be printed. If the specified row number does not exist, send back a message containing `ERROR`. Go back to Step 2.

If the fork is not successful, the process should perform steps 4 and 5 (in the original Program Specifications).

Important: Provide instructions on how to compile your code for this task. Write these instructions in a text file named `README`. Alternatively, you can create a rule in the Makefile in Task 3.

**Marking Criteria for Task 1:**

| Criteria | Weight | Expectations for Full Marks |
|---|---|---|
| Commenting | 10% | Source code contains sufficient and appropriate comments |
| Coding Style | 10% | Source code is formatted, readable and uses a coding style consistently |
| Correctness | 80% | Uses syntax in the declarations and/or definitions |
| | 100% | |

**Marking Criteria for Tasks 2 and 4:**

| Criteria | Weight | Expectations for Full Marks |
|---|---|---|
| "Compilability" | 10% | Source code compiles without warnings |
| Commenting | 10% | Source code contains sufficient and appropriate comments |
| Coding Style | 10% | Source code is formatted, readable and uses a coding style consistently |
| Correctness | 70% | Implements all specifications correctly and handles all possible cases correctly |
| | 100% | |

**Marking Criteria for Task 3:**

| Criteria | Weight | Expectations for Full Marks |
|---|---|---|
| Correctness | 70% | Rules result in correct behaviour |
| Completeness | 30% | All dependencies are included in the rules |
| | 100% | |