Computer Science 367

Program 2

Read all of the instructions. Late work will not be accepted.

Overview

In our final network programming assignment, you will implement the server for a simple online chat application. The chat application will consist of two distinct types of clients, which will be provided. The first type of client is a *reader*. Reader clients receive and display a stream of messages from the server in real-time; they do not send any data. The second type of client is a *writer*. A writer's primary job is to send messages. In a more sophisticated program, both of these clients would be integrated into a single display. To simplify the implementation of Program 2, we treat these separately. The server should be able to support up to 255 readers, and an equal number of writers. The following shows an example output of an reader client.

```
>A new reader has joined
>A new writer has joined
>A new reader has joined
>A new writer has joined
>hello!
>hi
>bye
A writer has left
>bye!
A new writer has joined
> I'm back!
> welcome back
>thanks
A writer has left
A writer has left
```

In this program, a range of events can happen at arbitrary times, and your program must be able to respond to them immediately. At any given time, a new writer or reader could join, a writer or reader could quit, or any of the writer could send a message. To accommodate this, you must use `select(2)` (or `poll(2)`) in your implementation of Program 2. `select(2)` is a system call that allows a program to monitor several socket (or file) descriptors, and return as soon as any of them are ready to read (or write, or report an error). Letting `select` babysit your socket descriptors permits you to react immediately to the aforementioned events. `poll(2)` is a similar system call with roughly equivalent functionality.

You are responsible for implementing (in the C programming language) the server, using `select(2)` (or `poll(2)`).

Command-Line Specification

The server should take exactly two command-line arguments:

1. The port on which your server will listen for readers (a `uint16_t`).
2. The port on which your server will listen for writers (a `uint16_t`).

An example command to start the server is:

```
./prog2_server 36799 46799
```

Either client should take exactly two command-line arguments:

1. The name or address of the server (e.g. `cf416-01.cs.wwu.edu` or a 32 bit integer in dotted-decimal notation)
2. The port on which the server is running, a 16-bit unsigned integer

An example command to run the reader client is:

```
./demo_reader 127.0.0.1 36799
```

An example command to run the writer client is:

```
./demo_writer 127.0.0.1 46799
```

Compilation

Your code should be able to compile cleanly with the following commands on CF 416 lab machines:

```
gcc -o prog2_server server.c
gcc -o demo_reader reader.c
gcc -o demo_writer writer.c
```

Protocol Specification

The protocol is summarized by the following high-level rules:

- Reader and writer clients may connect and disconnect at any point.
- All messages sent from all writer clients must be sent to all reader clients.
- The server should respond immediately to all messages being sent to it (e.g. messages from active participants, connect attempts, etc.). It should do so by using the `select` call to monitor all sockets. *No blocking on recv, send or accept!*
- Connecting and disconnecting clients to the server
  - When the server dectects that a new writer has connected
  - Similarly, when the server dectects that a new reader has connected, it should send the message`"A new reader has joined"`.
  - When the server detects that an a writer has disconnected, it should send the message `"A writer has left"` to all readers.
  - When the server detects that an reader has disconnected, no message is generated.

Submitting your work

Submit your implemented `prog2_server.c` file to canvas. Your submission need not and should not contain your compiled binaries / object files. I will compile your program, run it in a series of test cases, and review your code.

Academic Honesty

To remind you: you must not share code with anyone other than your professor: you must not look at any one else's code or show anyone else your code. You cannot take, in part or in whole, any code from any outside source, including the internet, nor can you post your code to it. If you need help from any other groups, all involved parties *must* step away from the computer and *discuss* strategies and approaches - never code specifics. I am available for help during office hours. I am also available via email (do not wait until the last minute to email). If you participate in academic dishonesty, you will fail the course.