Lab 11

Due Dec 1, 2023 by 6:30p.m.   **Points** 1

Lab 11: Select

Introduction

The purpose of this lab is to practice using the `select` system call to read from multiple inputs. This system call is used in assignment 4.

As usual, obtain the starter code for this lab from MarkUs.

Like last week, you should change the port in the `Makefile` before doing anything else. Take the last four digits of your student number, and add a 5 in front. For example, if your student number is 998123456, your port would be 53456. Using this base port, you may add 1 to it as necessary in order to use new ports (for example, the fictitious student here could also use 53457, 53458, 53459, 53460). Sometimes, when you shutdown your server (e.g., to compile and run it again), the OS will not release the old port immediately, so you may have to cycle through ports a bit.

An Echo System

Start by carefully reading through the starter code carefully making sure you understand what it already does.

Start with the `chat_client.c` code. This program creates a connection to a server. Then, it reads input from the user, sends it to the server, and waits for a response which it displays back to the user on standard out. It strictly alternates these reads (from the keyboard and from the server).

What happens in `chat_server.c`? It accepts a connection from a client. Then, it waits for input from the user and then echos it (it sends what is received right back). It keeps information about multiple clients in an array and does this echoing operation independently with each of them. In other words, input from one particular client is echoed *only* to that same client—not to all of the others. The server uses `select` in a loop to process input from whichever client is talking without waiting for others.

The goal of the lab is to create a functioning chat system where the messages sent by one client are delivered to all the other clients and participants are not required to talk in some pre-defined artificial turns. To accomplish this, you will need to change both the client and the server.

Task 1. Identify Yourself

Once we start broadcasting messages from the server, it will get confusing if we can't identify who the message comes from. The client asks the user for a username, reads the username from `stdin`, and writes that username to the socket.

The server should be modified so that it reads the username after accepting the connection. But we can't just add the `read` call in the `accept_connection` function because the `read` might block. Remember that we should only call read on sockets where select has indicated that there is something ready to read. Instead, your task is to modify `read_from` to see if we have received a username yet (how can we tell?), and allocate memory for the username field in the `sockname` struct and copy the name to the field for that client. After the username has been set, all new incoming messages from that connection will be messages to echo.

The username should be added to the table of `struct sockname`s it is managing. Then, whenever the server echoes a message, it should prefix the message with `<USERNAME>:`, where "`<USERNAME>`" is the username of the connection that sent the message.

Task 2. Start Echoing

This is a quick task. Change the `read_from` function so that it broadcasts any message received to all connected clients. Then, test by connecting more than one client to the server and making sure that every client receives any message that the other clients send. It'll be a bit awkward since our clients are reading from `stdin` before reading from the socket. You'll have to hit `Enter` on the client that didn't send the message to see what the other client has sent, and that will lead to the other clients receiving "empty" messages.

Task 3: Monitor `stdin` and the Socket in the Client

Take a look at how the server uses `select`. In particular, look at the `fd_set` variables it is managing and how it checks which file descriptor is ready for reading using `FD_ISSET`. Pay special attention to how the `fdset` used in the `select` call is reset for each new call. Make sure you understand how the `chat_server.c` starter code works *before* you try to use `select` in the client.

Your task is to update the client so it monitors just two file descriptors: the socket with the server and `stdin`. Whenever a message is received on either, your program should read it and pass it to the correct output stream.

Sample interactions

To help you better understand our chat system, we provide two sets of sample interactions. The behaviour of your program should be consistent with these interactions.

Please note that these interactions do not cover all possible scenarios and your program must take

other situations into account. For example, the number of clients can vary, and clients can connect
and disconnect at any time.

- Interaction 1:

```
$ ./chat_client
Please enter a username: Karen
Hi
[Server] Karen: Hi
How is it going?
[Server] Karen: How is it going?
I'm glad it is Friday!
[Server] Karen: I'm glad it is Friday!
Bye
[Server] Karen: Bye
^C
```

```
$ ./chat_server
Accepted connection
Echoing message from client 4
Echoing message from client 4
Echoing message from client 4
Echoing message from client 4
Echoing message from client 4
^C
```

- Interaction 2:

```
$ ./chat_client
Please enter a username: Karen
[Server] Jared: Hi
Hello
[Server] Karen: Hello
[Server] Jared: How's it going?
Great. We had fun with the select example in class.
[Server] Karen: Great. We had fun with the select example in class.
[Server] Jared: Good to hear.
Have a good weekend.
[Server] Karen: Have a good weekend.
[Server] Jared: bye
^C
```

```
$ ./chat_client
Please enter a username: Jared
Hi
[Server] Jared: Hi
[Server] Karen: Hello
How's it going?
[Server] Jared: How's it going?
[Server] Karen: Great. We had fun with the select example in class.
Good to hear.
[Server] Jared: Good to hear.
[Server] Karen: Have a good weekend.
bye
[Server] Jared: bye
^C
```

```
$ ./chat_server
Accepted connection
```

```
Echoing message from client 4
Accepted connection
Echoing message from client 5
Echoing message from client 5
Echoing message from client 4
Echoing message from client 5
Echoing message from client 4
Echoing message from client 5
Echoing message from client 4
Echoing message from client 5
Echoing message from client 4
Echoing message from client 5
Client 4 disconnected
Client 5 disconnected
^C
```

MarkUs checker: only checks compilation

We have provided a checker program for Lab 11 on MarkUs, but it only checks that your submission compiles. It does not check the correctness of your program.

Submission

Submit your final `chat_client.c,` `chat_server.c,` and `Makefile` to MarkUs under the `Lab11` folder in your repository. Remember not to commit executable files.

Congratulations! You've finished the last lab of the course. Good luck with the last assignment!