CS-392 Systems Programming Spring 2022

Assignment 7

Deadline: see Canvas

1 Objective

This assignment asks you to work with the sockets API in the C programming language. You will create a chat client that is able to communicate with a server. The client will send messages to the server, which is then able to broadcast the message to all the other clients on the system. Similarly, the client will be able to receive messages directly from the server, thus enabling the user to read messages sent by all other clients on the system.

2 Specifications

Your program should be written in a source file called `chatclient.c`.

2.1 Command-line Arguments

The program requires two command-line arguments: server IP and port. If no arguments have been supplied, the following usage message should be printed to `stderr`:

```
Usage: %s <server IP> <port>
```

where `%s` is the name of the executable, and the program should terminate in failure. The IPv4 address should be written in dot-decimal notation: `a.b.c.d`, where a, b, c, and d can be any integer from 0 through 255. If you are running the server locally, you specify the address as `127.0.0.1`. Do not worry about getting the word `'localhost'` to work. Use `inet_pton()` to check the IP address for validity. If the IP address is invalid, print the message

```
Error: Invalid IP address '%s'.
```

and exit in failure. In this case, `%s` is the string the user entered for the IP address, which is invalid.

The port must be an integer in the range [1024, 65535]. Use `parse_int()` , supplied in `util.h` to help you with this task. Associated error messages are as follows:

```
Error: Invalid input '%s' received for port number.
Error: Integer overflow for %s.
Error: Port must be in range [1024, 65535].
```

2.2 Prompting for a Username

After parsing the command-line arguments, the program should prompt the user for a username. If the length of the username is 0, the program should reprompt. The same holds if it is too long, in which case the program outputs

```
Sorry, limit your username to %d characters.
```

where `%d` is `MAX_NAME_LEN` and then reprompts the user. Do not worry about the case where the user enters only spaces. Do not worry about trimming leading and trailing whitespace the username either. If these conditions bother you, feel free to implement solutions, but we will not test your program against said conditions.

After obtaining a user name, print

```
Hello, %s. Let's try to connect to the server.
```

where `%s` is the username.

2.3 Establishing Connection

After obtaining the user name, which may actually be the same as someone already logged into the system, your program should create a TCP socket. If that fails, error out with the message

```
Error: Failed to create socket. %s.
```

where `%s` is obtained by `strerror()`. Throughout the rest of this document, the `%s` in any error message ending with `"%s.\n"` can be assumed to be from `strerror()`.

Then your program should attempt to connect to the server. If it fails, error out with the message:

```
Error: Failed to connect to server. %s.
```

The program should then try to receive the welcome message from the server. If it fails, error out with the message:

```
Error: Failed to receive message from server. %s.
```

Interestingly, if the number of bytes received is 0, that means the server closed the connection on the client. In that case, error out with the message:

```
All connections are busy. Try again later.
```

and exit the client program.

Note: You should store all messages received from the socket in `inbuf` and produce all outbound messages from `STDIN` in `outbuf`. It makes the code easier to understand.

If the connection is successful, print the following: a new line, the welcome message received from the server, and two more new lines.

Finally, send the username to the server. If it fails, error out with the message:

```
Error: Failed to send username to server. %s.
```

2.4 Handling Activity on File Descriptors (Sockets)

Using `fd_set` and `select()`, your program should now loop forever, prompting the user for input and determining if there is activity on one of the two file descriptors. Specifically, both the socket and `STDIN_FILENO` should be added to the `fd_set`. If there is activity on

(1) `STDIN_FILENO`: Get the string from standard input and store it in `outbuf`. Use `get_string()` supplied in `util.h` to help you with this task. If it's too long, print to standard output:

```
Sorry, limit your message to %d characters.
```

where `%d` is `MAX_MSG_LEN`.

Otherwise, send the message to the server, regardless of its contents. Then, if the message is equal to "bye", print `"Goodbye.\n"` and shut down the client.

(2) Client socket: Receive data from the socket and store it in `inbuf`. If the number of bytes received is -1, warn the user with the message:

```
Warning: Failed to receive incoming message. %s.
```

If the number of bytes received is 0, the server abruptly broke the connection with the client, as in the server crashed or perhaps the network failed. Error out by telling the user:

```
Connection to server has been lost.
```

Null-terminate the string, and compare it to "bye". If it's equal, print

```
Server initiated shutdown.
```

and exit the program. Otherwise, print the message from the server.

Make sure your close the socket before your program terminates. Be sure to check for memory leaks as well.

3 Starter Code

Following is the starter code for client that you can build on.

```c
#include <arpa/inet.h>
#include <errno.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <unistd.h>

#include "util.h"

int  client_socket = -1;
char username[MAX_NAME_LEN + 1];
char inbuf[BUFLEN + 1];
char outbuf[MAX_MSG_LEN + 1];

int handle_stdin() { /* Your code here */ }

int handle_client_socket() { /* Your code here */ }

int main(int argc, char *argv[]) { /* Your code here */ }
```

4 What To Submit

Submit a zip containing called `chatclient.zip` that contains:

- `chatclient.c`;
- `util.h`;
- Makefile (let's revisit Makefile, shall we).

Do not make any modifications to the server code, and do not include the server code or binary in your zip. We will compile the server code as-is and test your client against it. Your Makefile should generate a binary `chatclient`.