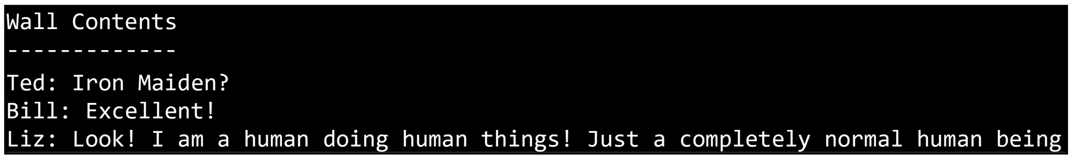
Ex8: Networking

Overview

This exercise serves as a brief introduction to TCP servers and the network toolset. In this activity, you will create a “wall” application, in which a single user can connect in order to “tag” (leave a message on) the wall. Wall programs were common in the days of Bulletin Board Systems (BBSs) before other Internet services became popular. As connections were made via telephone hardline dial-up, only one user could connect at a time, so the “wall” program served as a community board:



```
Wall Contents
-------------
Ted: Iron Maiden?
Bill: Excellent!
Liz: Look! I am a human doing human things! Just a completely normal human being
```

Figure 1. Example of a wall program’s contents

As text mode screens are generally 25 rows by 80 columns, wall messages should not exceed a certain length, otherwise the wall will look ugly! Users should be told what the maximum message length is, and messages longer than this should be rejected. Additionally, a limited number of messages can be displayed on the wall; when the wall is full, the oldest message is removed from the message queue, and the new message is added to the end (FIFO). The server and client must communicate through a TCP socket (in **C** or **C++**) and **must compile and run on Reptilian**. The wall’s state should be maintained even between connections. The `netcat` command line utility should be used to test the server.

Specification

Your server will run as a standalone program from the command line and will use the protocol specified below.

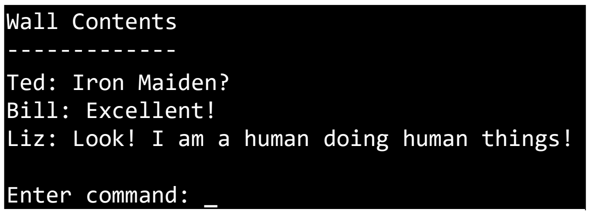
Command Line Execution

The server program will take up to two parameters, optionally – the maximum number of messages stored and the port. If not provided, the port should default to 5514, while the number of messages should default to 20:

```
$ ./wallserver         ←——— Queue size 20, Port 5514
$ ./wallserver 30      ←——— Queue size 30, Port 5514
$ ./wallserver 35 7777 ←——— Queue size 35, Port 7777
```

Server Behavior

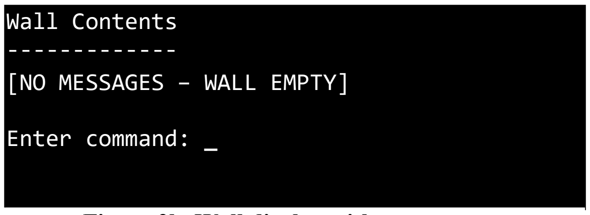
When a client connects, the server should send the wall’s contents and a prompt as shown below (**Figure 2a**). If there are no message entries, it should instead send “`[NO MESSAGES – WALL EMPTY]`” (**Figure 2b**).



```
Wall Contents
-------------
Ted: Iron Maiden?
Bill: Excellent!
Liz: Look! I am a human doing human things!

Enter command: _
```

Figure 2a. Wall display with contents.



```
Wall Contents
-------------
[NO MESSAGES – WALL EMPTY]

Enter command: _
```

Figure 2b. Wall display without contents.

The server will accept four distinct commands – `clear`, `post`, `kill`, and `quit`. For commands that do not cause the user to disconnect (`clear` and `post`), after processing the command, the server should display the wall's contents and prompt the user for an additional command, as shown in the examples below.

clear

Removes all messages on the wall. In addition, the server should send a message indicating that the wall has been cleared. Note that the server still displays the wall's contents after clearing.

```
Enter command: clear↵
Wall cleared.

Wall Contents
-------------
[NO MESSAGES – WALL EMPTY]

Enter command: _
```

post

Indicates the user wishes to tag the wall. The user should be prompted for their name and then their message. The entire post must not exceed 80 characters (including name and separator), so the maximum length of the message will be indicated to the user.

If the message is too long, the server should display the message "**`Error: message is too long!`**"; otherwise, it should display "**`Successfully tagged the wall.`**"

If the wall is full, the oldest message (at the top) should be removed from the wall to make room for the new message. An example (with queue size 2) is shown on the right.

```
Wall Contents
-------------
Jimmy Dean: Try my breakfast delights!

Enter command: post↵
Enter name: ~~~~~[[[[[[[[[[[THE PLAGUE]]]]]]]]]]]~~~~~↵
Post [Max length 36]: 12345678901234567890123456789012345678↵
Error: message is too long!

Enter command: _
```

```
Wall Contents
-------------
Jimmy Dean: Try my breakfast delights!

Enter command: post↵
Enter name: Johnny S↵
Post [Max length 70]: I'm alive!!!↵
Successfully tagged the wall.

Wall Contents
-------------
Jimmy Dean: Try my breakfast delights!
Johnny S: I'm alive!!

Enter command: post↵
Enter name: Bobo↵
Post [Max length 74]: Hullo.↵
Successfully tagged the wall.

Wall Contents
-------------
Johnny S: I'm alive!!
Bobo: Hullo.

Enter command: _
```

kill

Causes the server to shut down (terminate), and close the socket, disconnecting the user.

```
Enter command: kill↵
Closing socket and terminating server. Bye!
$ _
```

quit

Displays a termination message and closes the client's socket, but does not shut down the server or clear the wall.

```
Enter command: quit↵
Come back soon. Bye!
$ _
```

Debugging

It is recommended that students debug their server using the `netcat` command line utility (with alias `nc`). To do so, run your server in one ssh session, then open another and run `netcat`:

```
$ ./wallserver 2↵
Wall server running on port 5514 with queue size 2.
```

```
$ netcat localhost 5514 ↵
Wall Contents
-------------
Pigeon: You've got mail. Err, I mean, cooo. Coooo.

Enter command: quit↵
Come back soon. Bye!
$ _
```

Submissions

You will submit the following at the end of this exercise:

- Screenshots demonstrating your server's functionality (`func1.png`, `func2.png`)
- Compressed tar archive containing server source code and Makefile (`ex8.tar.gz`)

Screenshots

In Reptilian, do the following:

1) Start the server with a message queue size of 2
2) Connect to the server using `netcat`
3) Post three messages
4) *Take a screenshot of steps 1-3, name it "func1.png"*
5) Clear the server's wall
6) Post one message
7) Attempt to post a message that fails due to being too long
8) Quit from the server
9) Reconnect to the server
10) Kill the server
11) Attempt to connect again to show that the server has terminated
12) *Take a screenshot of steps 5-11 (include wall contents from before step 5), name it "func2.png"*

Compressed Archive (ex8.tar.gz)

Your compressed tar file should have the following directory/file structure:

```
ex8.tar.gz
    ↳ ex8 (directory)
          ├→ Makefile
          └→ (Server source files)
```

To build the server program and run it, we will execute these commands:

```
$ tar zxvf ex8.tar.gz
$ cd ex8
$ make
$ ./wallserver [lines] [port]
```

Resources

https://linux.die.net/man/2/socket - documentation for a function that you'll be getting comfortable with

https://www.unixfu.ch/use-netcat-instead-of-telnet/ - an http example using `netcat`

https://linuxhandbook.com/jobs-command/ - as an alternative to opening multiple terminals, you can keep the server running in the background by learning how to manage jobs (not necessary to complete the exercise, but may be interesting to students that like working in the terminal)