Pun Server and Client

Build a network server and client to implement the "pun service". The server and client will use port 5432*.

There are no command line arguments or output for the server (no interface - it is a daemon). The server shutdown should be handled by catching the SIGTERM signal. The server should use select(2) to terminate immediately when the SIGTERM signal is received.

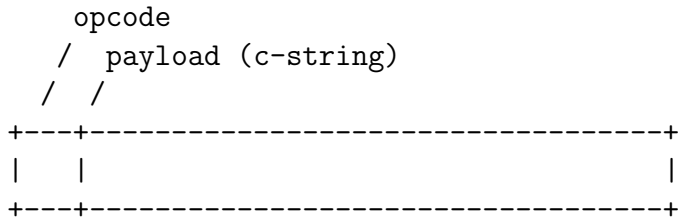
The client has one required command line argument, the hostname or IP address of the server. On success, the client will display the joke message payload to stdout and then terminate. The client does not have a menu or accept user input.

Both the server and client should handle errors gracefully. Graceful means your programs should recover from the error and continue if possible, otherwise they should display an informative message, cleanup resources, and terminate.

The pun protocol has seven message types as follows:

OpCode	Description	Example payload
100	Hello server, sent from client	"tell me a joke"
200	Hello client, sent from server	"knock, knock"
300	Ask who setup, sent from client	"who's there"
400	Who response, sent from server	"yoda lady"
500	ask *response* who?, sent from client	" yoda lady who?"
600	Punchline, sent from server	"good job yodeling"
700	Error, may be sent from server OR client	"Protocol violation"

The format for each message is as follows:



- Opcode, 3 bytes, ASCII
- Payload, 128 bytes, ASCII, Null terminated

Notice that the entire message is ASCII (plain text, no integers). This is conventional with

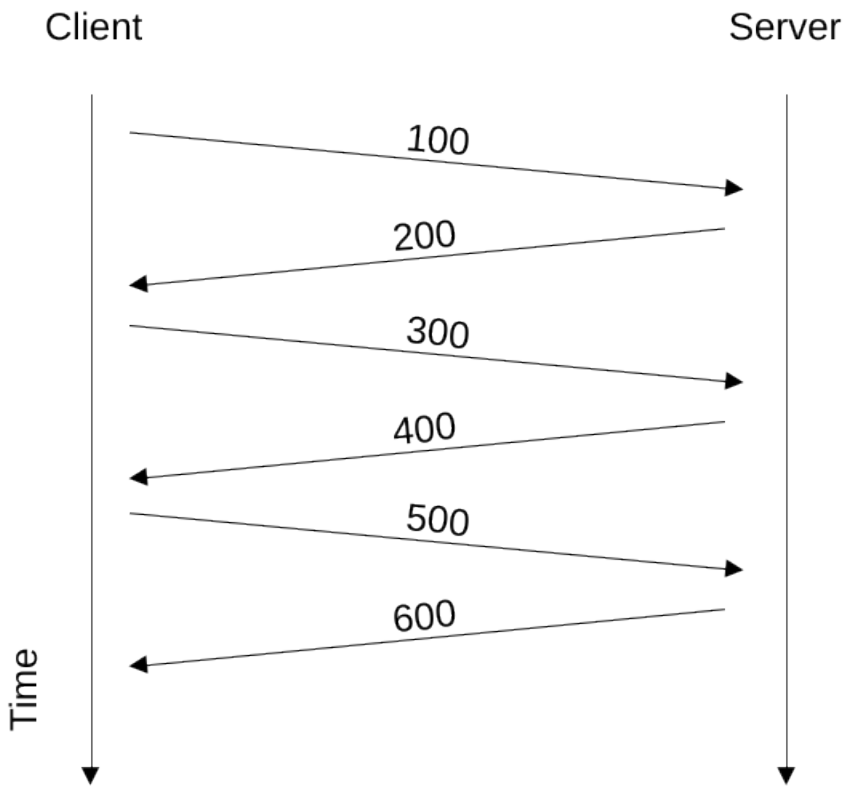


Figure 1: Pun protocol message exchange

most Internet protocols. The client always starts the protocol and only executes one pass through the protocol. Messages are exchanged as shown in Figure 1.

Implement your protocol code (common data structures and functions) in separate protocol.h and protocol.c files. Your protocol code must include functions to assemble a message (opcode + payload), extract an opcode from a message, and extract a payload from a message. Your protocol.c, server.c, and client.c code must include your protocol.h file and use your protocol code.

The opcode 400 payload should be randomly selected from one of the available responses in the joke file (ASCII text file). Download a sample joke file from the class web page into your program folder for development and testing. The joke file contains 2 lines per joke, the "who response" (used with opcode 300) and the punchline (used with opcode 600). Figure 2 shows one example entry from a joke file.

The joke file must be loaded into memory when your server program starts. The joke file should be closed immediately after the jokes are loaded into memory. Use a linked list to

```
Yoda lady
Good job yodeling
```

Figure 2: Example joke file entry

store your joke file in memory in the server. Be sure to deallocate the list the before your server shuts down.

Use the *arc4random(3)* function to sample the PRNG and determine which joke from the list to "tell". Do not use the rand(3) function. See the rand(3) man page for why it should never be used in critical code.

All messages received should be checked for validity. This means the message is the expected type code and the payload length does not exceed the allowable length. You may include a newline in the payload if you wish (you must include a null byte).

Message code 700 is only emitted when the protocol is not followed correctly or a message is invalid. If either side of the connection receives a message that fails the sanity check, then the receiving program should send a type 700 message and disconnect. Either side of the connection (server or client) must send a type 700 message and disconnect when the protocol rules are violated.

Your programs must be developed in *C* -std=c17 (the default) using the BSD socket API on the CS server. The programs must compile and run on the CS server with cc (clang).

Your program must conform to the **UNA CS Code Style Guide** linked on the class web page. Your program must compile without any compiler warnings to receive full credit.

Testing your server without your client:

See the note below about ports for testing your programs.

You may test your server, before you create your client, with the netcat utility nc(1). The netcat utility will send and receive messages over the network. To test your TCP server listening on port 8017, run your server in one terminal window (server's terminal). Then in a different terminal window (client's terminal) you would type the following:

```
echo "test message\r" |nc localhost 5432
```

If your server is working correctly you'll see the string displayed in the client's terminal. The port number must match the port your server is listening on in order to work.

Instead of logging in twice to the server, you may use the terminal multiplexer tmux(1). The following list shows the tmux commands needed to open two screens for testing:

- `tmux` - start the tmux program
- `CTRL+b "` - split the current pane into two panes, top and bottom
- `CTRL+b ↑` - move the cursor up a pane
- `CTRL+b ↓` - move the cursor down a pane
- `exit` - close a pane (closing the last pane exits tmux)

See the man page tmux(1) for additional information.

★ Note: To avoid conflicting with your classmate's programs when testing on the CS server, use the last three digits of your student ID L# plus 11000 for your port number.

Due dates:

Friday 8th before 11:59 p.m.
Completed server and Makefile commit, tag it "server"

Wednesday 13th before 11:59 p.m.
Completed client commit, tag it "client"

To tag the commit, use an annotated tag *after* your commit.

How to submit:

Create an empty git(1) repository in a folder named "program-5". Commit and tag your files as described above in "Due dates". You may, and should, commit more frequently than the required commit dates. You should build your server and client programs in stages (skeleton, startup, etc.) and commit the working version of each stage as you complete it.

Program evaluation:

Your program must conform to the **UNA C Code Style**. The "UNA C Code Style" is linked on the class web page. Code will be evaluated for correctness, efficiency, and style.

Helpful resources:

```
Command line arguments: https://www.geeksforgeeks.org/command-line-arguments-in-c-cpp/
--
man pages - err(3), printf(3)
            select(2), signal(3)
            SLIST_INIT(3),
            style(9), nc(1)
```

Additional resources:

```
https://beej.us/guide/bgnet/
https://csrc.nist.gov/Projects/ssdf
https://sourceware.org/gdb/current/onlinedocs/gdb.html
```