COP4635 - Spring2024

Chase Lamkin

Project 1

Feb 18th 2024

Protocols

Server Initialization – A high level initialization overview is that the server will retrieve a socket file descriptor using the socket function. It will then specify which network interface (for this server it will listen to 0.0.0.0, which is all network interfaces), and the port to listen on using the bind function. Finally it will use the listen function to tell the socket file descriptor to start passively listening to the port, it will also specify how many waiting connections to hold before rejecting new ones. At this point the server is ready to begin establishing connections and will call the accept function to wait for incoming connections.

Client Initialization – The client (or browser in this case) will similarly create a socket. However, instead of calling listen and accept, to listen for connections passively, it will call connect to tell the socket where to establish a connection with. The server, which has already called accept, will recognize the connection and establish it with the client.

Runtime – After the connection is established the server will call write (or dprintf) to write information to the socket file, which will then be sent to the client. The client like the server will call accept, and will wait for the response until it is received. The client will then call write to send more information and the process will continue until either the server or client ends the connection. To end the connection the client or server will call close which will end the connection and tell the other to do the same.

TCP – TCP is a part of the transport layer and establishes a reliable connection during the http server and client process of this program. During the establishment of a connection between the server and client (via accept and connect) a three way handshake is done. The client sends a SYN packet to initiate the connection and the server responds with a SYN-ACK packet to indicate its willingness to establish the connection. Finally the client sends a ACK packet to acknowledge this. Once this connection is established the client server connection can be reliably used until a the connection is ended (via the close function). To end the connection a four way handshake is done. First the client or server sends a FIN packet to the other and then the client or server will respond with a ACK packet. Again the client or server will send its own FIN packet and the initiator will respond with an ACK packet to finally close the connection completely. Overall, TCP allows for the communication with the server and client to be reliable and enables HTTP to be reliable as well.

HTTP – This program follows a request response pattern because HTTP is a request response model. Now that TCP ensures reliable data transfer the client can send requests to the server which it will respond to over the established connection. HTTP is stateless, so each request-response pair are independent of eachother. HTTP also utilizes headers to provide metadata about the request or response so the server and client can know things such as the status of the request/response, what the request/response is to, what sort of method the request or response is for, and what the request/response content type is. The content itself is in the “body” of the request/response, and using the content type information, the server or client can understand how to interpret that data and respond to it or display it.

To explain this in a more practical way I will describe a client accessing the homepage. Over an established connection the client will send a GET request to the server for the path ‘/’ and using HTTP 1.1 protocol. The server will see that the GET request is to the path ‘/’ and will find which file/content it needs to respond with (in this case it is index.html). It will then respond with that file content utilizing the “text/html” content-type so the client (browser) knows how to display it. This is just one request-response instance (there also may be other meta data in the header such as URI parameters etc, or data in the body. This is just a simplistic example). During this request-response cycle the no other request-responses are affected, therefore making it stateless.