Documentation, IN2140 -- TCP Server Implementation

Connection. c:

The connection. c function consists of all the various methods for attempting to connect to a socket. Thereby while creating this file, diverse decisions and methods were used to attempt to make a functioning connection. c.

Tcp_connect: The tcp_connect function establishes a TCP connection with a server using a host address and port number. It creates a socket with characteristics AF_INET, SOCK_STREAM, and 0, signifying a reliable communication channel over an IPv4 network. During this process, error checking prevents critical faults and returns -1 if any error occurs. After setting the server's IP address and port, it uses the connect system call to establish the connection and return the socket file descriptor upon success. This descriptor is a unique identifier for the established connection, crucial for data transfer in the TCP proxy service.

Tcp_read: is designed to facilitate data transfer within the proxy. In this case, specifically used to read the files between XML and binary. It is implemented with the idea that it reads up to “n” bytes from the socket, stores it in a buffer for processing, and handles errors by returning -1 to ensure the read has been done accurately.

Tcp_write: uses the write system call to send data over a network socket. Like tcp_read, it processes data until the “n” amount of bytes are written.

Tcp_write_loop: This function primarily sends data by calling tcp_write continuously in the while-loop until the total amount of data sent equals the desired amount to send. The variables total_writtern and written are used to hold track and move the pointer further in the buffer to read the data. Likewise, the subtraction between bytes and total_written is used as an argument to ensure that the remaining data is sent.

Tcp_create_and_listen: is designed to set up a server socket that listens for incoming TCP connection on a given port number. This is done by creating a socket and specifying the IP address and port number. It also binds the server's socket to the client's address and listens for incoming connection requests.The server's socket can listen for incoming connection requests by invoking the listen system call. The length of the queue of active connections is represented by the number 3.

Tcp_accept: the purpose of this function is responsible for accepting a connection with an incoming client. It takes a single parameter server_sock, the server's socket descriptor. The socklen_t is a datatype in socket programming used to set the initial size of the client_addr structure to be called in accept system call. The accept function causes the server to be blocked until a client connects to the server socket.

Tcp_wait: This function will pause the process by taking a set of descriptors and the highest descriptor value, and then return the number of ready descriptors. The select part is used here to use multiple clients to wait for the process efficiently. The program's design plays a crucial role in managing several simultaneous connections.

Tcp_timeout: This function serves the same purpose as tcp_wait. However, the struct timeval makes the process wait for a given time. This method does not hold indefinitely. Instead, it will take an extra argument for the duration.

Proxy.c and RecordFromFormat:

This proxy is made to manage up to 200 clients and functions on a singly linked list to manage the various clients. The linked list stores information from the struct: the fd, type, and id, and the next pointer for multiple clients. The main reason for implementing this design is the efficiency of traversing between the clients. Furthermore, the main loop runs the program by ensuring the proxy server waits for incoming clients. Thereby when a new client is connected, indicated by the server socket, it adds the new client to the linked list via the handleNewClient() function. With a loop, it then helps the server to continuously check for incoming new connections and iterate over the file descriptors to find active ones until the clients have disconnected when num_clients is 0 with the termination code. Also, as new clients are added, their corresponding file descriptors are added to the set by FD_SET(), and the FD_ISSET() checks for incoming active connections. These are with another loop used to traverse the client list to see if the file descriptors correspond to their respective clients.

Moreover, after adding the clients to the list, the handleClient manages their activity (). The function works the communication by using tcp_read to read from the client file descriptor into a buffer before taking account of its relative type. Read_bytes are used to hold the amount of data read, and the variable currpos tracks the buffer's position. The data will be sent to either XML or binary, specified as type “X” or “B.” Then it is parsed into XMLtoRecord() to convert into an intern format. Data is sent to their desired destination by calling forwardMessaage(). However, if the reading fails, the client is removed, and the message is not sent. If successful, the position in the buffer will be updated, and the remaining bytes will be updated so incoming data is handeld regardless if it's a large or small file.

The forwardmessage() ensures that the clients in the list get their corresponding message based on their ID and destination. It checks for error handling if the client isn't on the list. Nonetheless, if the client is found, the conversion between the record and its type is done by calling either the recordtoXML or recordTOBinary() function, depending on the type the client is sending to. Then if the conversion is successful, the tcp_write_loop() is sued to write and convert the message to the receiver.

The direct conversion between the record and its type occurs in the recordtoformat file. The XMLtoRecord reads the data in the buffer being sent by the client and parses the XML tags based on the tags in the records. For this function, pointers traverse the buffer and find the needed titles. It uses helper functions to create the new record and its different tags from the file record.c. To find the corresponding tags strstr is used to efficiently find the first occurrence of a substring in a string. The end tags must be found first to null-terminate the record if found. This is done so it can handle if sleep is set to 0. Likewise, a while loop is run as long as there is data within the buffer and when there is no end tag found. To keep track of the lastprocessedrecordEnd variable, mark the end of the last processed record to help calculate the total bytes read. For grade and course, strcmp is used for string comparison to convert the string into integer values. If the record is not found, the record deletes the incomplete record. The bytesread pointer helps update the number of bytes processed in the buffer, which is done at last.

Similar to XML, the binaryToRecord function converts binary data in the buffer into a record. The record employs a specific byte to detect various tags using a bitwise AND operation. Source, destination, semester, and grade are assigned their values directly, and the pointer is incremented by one due to each flag consisting of a single byte. Multi-byte fields increment based on the number of bytes they take up. For the username, whose byte count is unknown, its length is determined by reading the unit32_t value from the buffer and converting it to host byte order using ntohl, followed by memory allocation. Other fields use unsigned bytes to denote the number of integer bytes in the buffer. Multi-byte fields undergo conversion from network (big endian) to host byte order using ntohl and ntohs functions. All fields are processed, and the total bytes read from the buffer are assessed for error handling. If the bytes read exceed the buffer size, it indicates an incomplete record.

Performance:

- AnyReceiver functions correctly in that it waits for the given seconds to allow the proxy to receive data; however, it requires a forced stop to halt the AnyReceiver.