Network Assignment

z5160184

November 2019

1 Multi User Messaging Protocol (MUMP)

- Project was written in C
- All messages sent through both the p2p and server, will have a header "MUMP" followed by a newline.
- After the newline the request type is expressed, since there are limited requests, i decided to make almost all of them one character long. I did this to make switch statements easier, and hence faster processing time. These include:
  - 'M' - message
  - 'B' - broadcast
  - 'W' - whoelse is online
  - 'S' - whoelsesince
  - 'K' - block
  - 'U' - unblock
  - 'L' - logout
  - 'D' - 1st client p2p request
  - 'P' - 2nd client p2p reuqest
  - 'F' - client end p2p request
  - 'X' - client end p2p acknowledgement
  - 'E' - the server will send to a client followed by a number, so the client knows if its request failed.
- an example would be the message header:
  MUMP
  M
  [sender name]
  [receiver name]

- the other header tags are expressed above the functions that create or process them
- The design is interesting, the server itself is single threaded, using the select() function to listen to all client sockets at once. I went this way because the application is lightweight and does not require too many send calls. The client however is multithreaded, it has 2 main threads, one for sending commands from the command line, the other receives data from the server. Furthermore with every private connectio created, 2 more threads are created
- the program includes a list data structure to keep track of clients, both on the server side and on the p2p client side. An improvement to my design would be to create a union of p2p_client_data and client_data, to save memory, instead of having that many attributes in client_data. However CSE servers wont break from an extra few bytes in memory.
- If this program were to be used by hundreds at once, i would improve the design to a multi threaded server. Because that approach is much more scalable. To be the most efficient and scabale,I would have 1 listening thread, and then I would put a certain amount of clients together on a thread (e.g 20) and then create a new thread for the next 20.