PURPOSE OF YOUR PROGRAM:

- The purpose of this program is to make a local network than contains the Server that managing communication for all users through forking children. Child to work as a middle man between user and server for send/receive information and the client to create users and connect them to the server.
- Learning how communication works through pipes on non-blocking. As well as how signals are handled in such a program. Additionally, how errors are dealt with in multi-terminal program.

HO DID WHAT:

- **Khai**: Polling for server and child, interrupt handling, P2P, /exit and /seg
- **Yahya**: Readme, testing and organizing code, communication between user and server.
- **Oguz**: Error handling, kick, cleanup_user, debug.

How To Compile:

- Run “make” using the included “makefile”

HOW TO USE THE PROGRAM FROM THE SHELL:

- After run the command “make”, call the server by typing ./server in one terminal
- Open another terminal and run ./client ‘client_name’. Every single client should be running in their own terminal
- Make sure that server and client have the same server_id
- To run a command type “\command_name”

WHAT EXACTLY THE PROGRAM DOES:

- **void kick_user(int idx, USER * user_list)**
  - This function outputs which users is being terminated with its Child-Pid. Then call the function kill_user() to kill the user and cleanup_user() to reset the user’s slot in the server.

- **void kill_user(int idx, USER * user_list)**
  - The function is meant to kill the user by obtaining use’s child_pid and then kill it which lead the client’s user to be terminated.

- **void cleanup_user(int idx, USER * user_list)**
  - The function will reset the slot of the terminated user to be reused again.

- **void broadcast_msg(USER * user_list, char *buf, char *sender)**
  - broadcast_msg() function will send the received message to the server and every user that is connected to the server indicating the user/server that sent the message using concat() function.

- **void exit_server(USER * user_list)**
    - When calling this function, every single user that is connected to the server will be killed and their slots will be cleaned. Then, the server will exit the program.

- **void send_p2p_msg(int idx, USER * user_list, char *buf)**
    - The purpose of this function is to receive the information of user sender, user receiver, and the private message. Call an error if any of the information is missing/not valid. And then send the message to that specific user indicating the user that sends it.

- **void runChild()**
    - This function is monitoring and organizing pipes and the information pass through them. It keeps polling waiting to receive any information from server/user. runChild() will terminate once it finds a broken pipe.

- **int DupOflow(char * user_name, USER * user_list)**
    - The function is meant to monitor the added users by indicating raising a flag if there are no enough slots to add additional user of it the name of the added user already taken. Then return the corresponding flag as an integer type

- **void execCommand(int cmd, char *buf, USER * user_list, int i)**
    - This function works by orienting the received command from server/user to its proper function. As well as calling an error if any of the information is missing/not valid.

- **char* concat(const char *str1, const char *str2);**
    - concat() function is meant to concatenate two strings together and then return a pointer to the concatenated strings.

- **int add_user();**
    - This function has been modified to accommodate error detecting results from adding a new user.

- **int find_user_index(USER * user_list, char * user_id)**
    - The function is meant to return the index of the desired user. It will return an error if the user does not exist or the user was not passed to the function

ASSUMPTION:

- \exit and \seg were handled in the client. The server doesn't know that the client typed "\exit" or "\seg". The server only knows that it has lost connection to the client.
- The pipes of any client will be closed as soon as the client terminates.

STRATEGIES FOR ERROR HANDLING:

- Most of error handling were implemented in their local function instead of sending it to another function to be printed.
- We use the fact that read() will return 0 if the pipe is broken. If the user crashed (Control C, segmentation fault, closing the terminal), the pipes that belong to the user will also closed. Therefore, when the child read the pipe, read will return a 0. If read return a 0, the child will kill itself. Inturn, when the server read the pipe from the child, it will also return a 0, server then do cleanup and remove that user from the list.
- When the server crash, the pipe between child and server will be broken. When the child tries to read from the pipe, it will return a 0. Child then kill itself, which caused the user to kill itself.
- All the pipes are non-blocking read. Therefore, it will not give “broken pipe” error.
- To handle error system call, we used “perror()” to print out the error on the terminal. For user’s input error, it was handled by “printf()” to inform the user regarding the input error.