Name: Muhammad Wasi Naseer

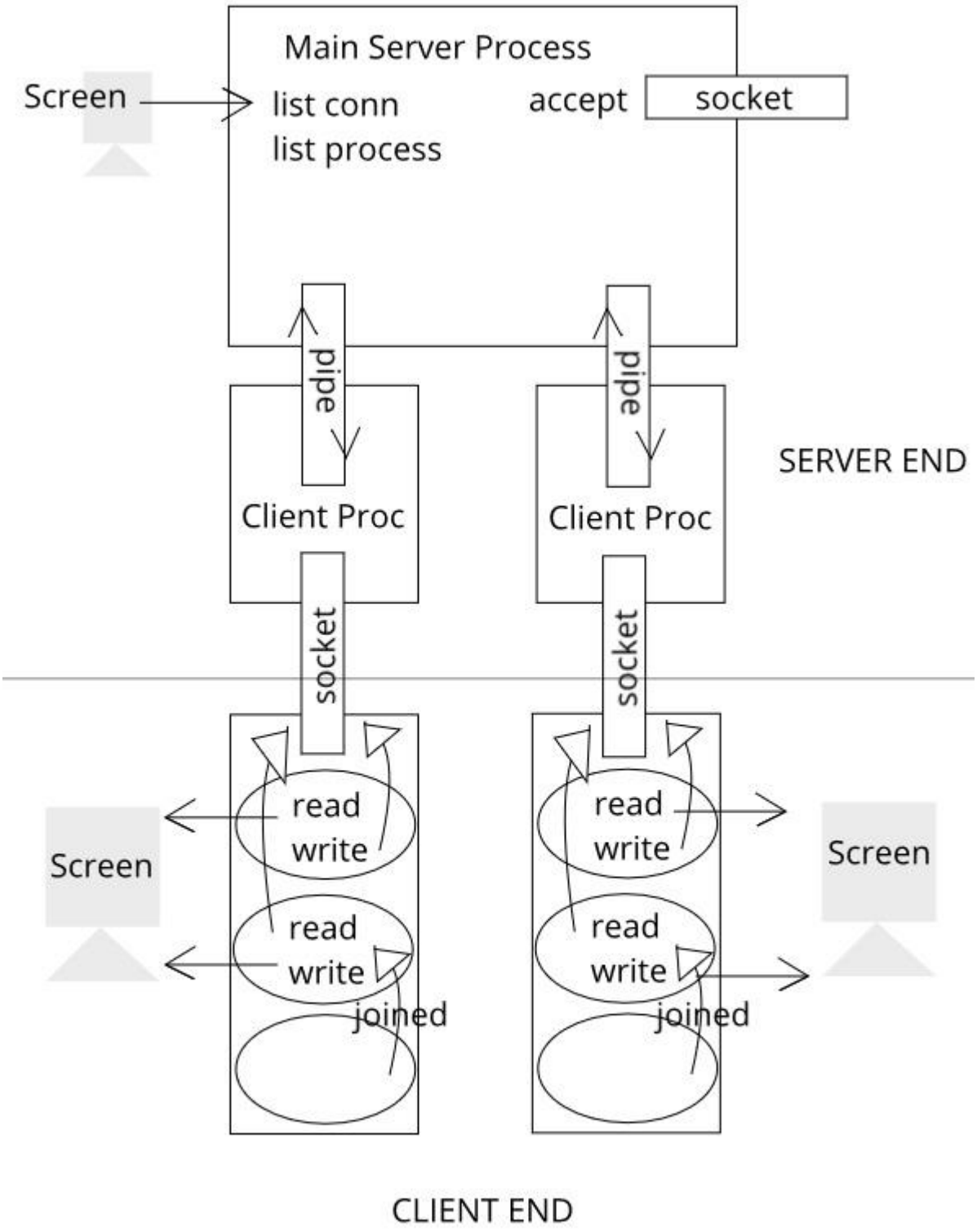
ERP ID: 11578

Systems Programming

Final Project Report

Content:

- Architecture Diagram
- Architecture Explanation
- User Manual
- Optional Work
- Limitations


Main Server Process
Screen
list conn
list process
accept
socket
pipe
pipe
SERVER END
Client Proc
Client Proc
socket
socket
read
write
read
write
read
write
read
write
Screen
Screen
joined
joined
CLIENT END

Architecture Explanation:

Server Side:

- Every time server starts, it creates a socket and use multiplexed I/O for two fds: one for accepting connection and one for reading from screen
- On reading from screen, it has following commands
    - list conn
    - list process
    - exit
- On accepting a connection from a client, it does two things
    - Registers the connection in ConnList
    - Creates two pipes and Fork the process for the client
        - The client process on server now implements multiplexed I/O for reading from client socket fd and reading end of the pipe
        - The client loops indefinitely and does either of the following things or both:
            - receiving commands from client end and processing them, and then writing on the socket
            - Receiving command from pipe(server process) for sending its process list.
- Signals Used:
    - SIGCHLD for client process on server: It is used when any of the processes, which are started from a client process, exit. It updates processList for the client process.
    - SIGCHLD for server process: It is used when any of the active connections disconnect. It updates the connList in the server process.

    Both the two signals reap as many zombies as they can using the combination of while loop and WNOHANG flag. It avoids the limitation of receiving only one signal when more than one signal of the same type arrived.

Client Side:

- Every time client starts, it prompts the user for connecting to a server with the following command:
    - connect <ip> <port>
- After connecting to server, two threads has been created.
    - Thread1: loops indefinitely for reading from screen and writing to server, it is detached thread. It is cancelled by thread 2.
    - Thread2: loops indefinitely until server sends it that it's process on server has been exited, and then it will disconnect and cancel the thread 1 and exiting itself. It is joined to main thread. It can exit the whole process if user wrote exit command.
    - Main thread: It waits for thread 2 to exit and the prompt the user again for connection. It will block to wait for thread 2 to exit. The thread 2 either exits the whole program, or exits itself and cancel the thread 1.
    - The process can only be exited if the user typed exit command. In that case, thread 2 will call exit. If the user typed disconnect , it will only exit the two threads and main thread will survive.

User Manual:

Server Side Commands:

1. list conn : It lists the all the connections including active and non-active.
2. list process : It lists all the process running in the each client processes.
3. exit: it exits the server and exits the client process

Client Side Commands:

1. connect <ip> <port>: connects the client to the server with the specified ip and port.
2. disconnect : disconnects the client from server , leaving the client process on client running and exiting the client process on server.
3. exit: disconnects the client from server , exiting both the client process on client and the client process on server.
4. add <number1> <number1> … : It adds the numbers
5. sub <number1> <number1> … : It subtracts the numbers
6. mult <number1> <number1> … : It multiplies the numbers
7. div <number1> <number1> … : It divides the numbers
8. help : prints help
9. run <program> : It runs the program
10. list: list the active process on the client
11. list all : list the all processes on the client
12. kill all: exit all the processes on the client
13. kill <name>: exit the process with the specified name on the client
14. kill <id>: exit the process with the specified id on the client

Optional Work:

- Each time client reads command from screen, it always first write its length in 3 bytes and the command. In the same way, the server first determines the length of command by reading only 2 bytes, and then reading those number of bytes and processing them.

Limitations:

- The server is not able to process a command with more than 100 bytes(possible in case of run and opening multiple files in gedit) because of fixing only 2 bytes for length of command.
- Kill by name will only kill the oldest process out of many processes with the same name.
- The commands are case sensitive