**Architecture**

**Server Side:**

Server was running in two processes, one to create connection with client called ‘Conn’ and one to handle the specific client called ‘client handler’ or ‘CH’.

In order to store process information of a client, struct “processInfo” was created, and to store process information for all clients, array of “processInfo” was used. For client handler’s information, struct was used, in which its id, msgsock, isopen (whether its closed or opened) and two pips id to communicate with the connection was stored. Array of this struct was used to store information of all client handlers called ‘cHandlers’. For both arrays counters were used to update the array. CH’s id was given what the count of ‘cHandler’ was at the time of CH creation.

4 threads, one main and three others in order to make the server interactive and do parallel processing.

**Main thread:**It first take necessary steps to establishes socket, then creates a thread\_read\_usr to read command from terminal and write to CH’s pipe. Then in loop, waits for accepting the connection with a client. Then it forks for client handler, establishes pipes with them through two fds, for reading and writing, (i.e. Conn will write to first pipe and CH will read from it and CH will write to second pipe and Conn will read from). It creates struct for new child handler, increase count, and adds it to the ‘cHandlers’. The child process, which is for client handler, will create a thread “thread\_CH\_read\_pipe” in which client handler info (a struct is passed) so that it can write msg to its client and lists or any msg to pipe2 fds. It then in while loop, read command of client from client through socket (i.e its msgsock fd), then process the command and write the output back to client.

**Signal Handlers:**Two signal handlers, one for conn and other for client handler are initialized with two signals sigchld and sigpipe. Conn handler with Sigchld signal will make child handler’s info updated, by making isClosed to true. Client handler with Sigchld signal will update its the terminated process’s endTime to current time (which was -1 previously to know if its active or ended).

**Thread “thread\_read\_usr”:**   
Conn's thread, this will continuously read command from server. It also checks for which client handler the command is and sends command to corresponding child handler's pipe1 through its fd(which it get from CH’s struct stored in ‘cHandlers’). For all, it sends commands to all client handlers. When it is writing command to a client handler’s pipe, it knows client handler will write to its pipe which it needs to read so it creates another thread “thread\_Conn\_read\_pipe” in which that client handler’s info is passed.

**Thread “thread\_CH\_read\_pipe”:** This thread is of a child handler and it will continuously read command from its reading pipe (i.e pipe1) to which Conn process has written, process it and then sends data to conn through writing on other pipe and also to client if command was print.  
If list is asked then, printProcess function and pipefd and a Boolean is given as an argument so that function knows which fd to write and whether active list is asked or only full list. If message was given to write to client so it sends the messageg which it gets as argument and write to client with its msgsock which was stored in struct which has been passed to it in thread and write acknowledgement to pipe that message was written successfully if no error occurs.

Thread “thread\_Conn\_read\_pipe”: This thread is of conn and it will finally read output from pipe (i.e. pipe2) to which child handler has given output. Then the output is shown to screen.

**Client-Side:**

It has 3 threads, main thread and other two threads which concurrently read data and process it and sends output.

Socket’s fd is global so that both thread can access it.

Main thread: establish socket, and connect to the server, sockFd is assigned. It then just creates two threads, and call pthread\_exit to terminate itself.

Thraed1: Continuously reads commands from user and writes to socket

Thread2: Continuously read data from server and write it to screen. Each output is separated by ‘:’ delimiter which is given by server when writing to socket so it separates outputs through this delimiter, so that if multiple output are given to it, it shows them one by one.