**Architecture**

**• Client:**

The client is a process which is constantly running in a while loop. When the client starts running, it prompts the user to provide the server’s IP address and its Port Number. If the server is active and the user enters the correct addresses, the client is connected to the server. There are two threads that run in the client process. One is the input thread and the other is the output thread. The input thread is running in an infinite loop and takes input from the user and writes it onto the socket’s file descriptor. The output thread is also running in a loop where it is constantly reading from the socket’s fd and printing the results on the screen. The user can send commands, one after the other or at most ten commands as batch. Output thread reads the results and displays it to the user. If a user enters “exit” as a command, it is read by the server and the client-server process attending to the client is killed along with its children (that have been created by forking and exec). The client is informed about this, which in return calls the exit () system call and the entire process terminates. However, if the user sends the “disconnect” command, the same would happen on the server side but the client process will not exit. Instead the output thread will send a pthread\_cancel () request to input thread and would terminate itself as well. The main thread which had been joined on these two threads, will go into the next iteration, again asking for a new connect command.

**• Server:**

The server is the program that is serving the client’s requests. To make the server interactive, the server’s main or parent process is blocked on two fds using the poll () System call. One of the fd is the STDIN\_FILENO which is waiting for input from the user at the server. The other fd is the socket’s fd that is waiting on new connection requests. If either of the two fd’s have data to read, the poll system call will be unblocked and the data at the respective fd is read. The main server process maintains a list of connections, that contains information about the IP addresses and port numbers of all connected clients. For a new connection, the accept system call returns the client’s socket fd. A child process (server-client) is created which is connected to the parent process by two pipes (one for sending data and one for receiving). The fds of the pipes are also stored in the connection list. The child process contains the code to facilitate client requests. For each connection, a child is created, and its details are stored in the connection list. If the user at the server requests to display all active connections, the list is provided at the server screen. The server-child process relates to the main server process through pipes and with the client though socket. Since this process can read commands from two fds (socket fd and the pipe), the two are polled. The process becomes unblocked when either of the two fds has data to read. The user at the server can also ask for a list of all processes running at the server. For that, a command is sent to all server-client processes, which in return prepare their respective process lists and send them back to the parent. The server parent process then displays results on the screen. The client is communicating with the server-child process using the socket as the source of medium. The client can request anything like adding, subtracting, multiplying, dividing numbers or running new processes on the server. (Refer to the help section for more details.) If multiple commands are received at the server, they are tokenized and stored in an array. Once the array is populated, the command at each index is processed and the results are sent back to the client. The addition, subtraction, multiplication and division commands, are further tokenized based on spaces and respective operations are performed. For a run command, the server-client process tokenizes the command and fills up an array. It then forks and calls the exec system call passing on the array. This child process is connected to the server-client process through a unidirectional pipe. If the exec is successful the pipe fd closes and the server-child process reads 0 byte from the pipe, indicating that exec was successful. It will then add the process information in the process list (Each server-child process maintains a list of all processes that it has forked and exec-ed). If the exec fails, the child process writes “fail” on the pipe fd which is in return read by the server-child process. The server-child process then informs the client about the failure of running a new process. The list all command, from the client returns a list of all active and nonactive processes whereas the list command only displays the active processes. If the newly created process terminates gracefully or if it is killed by a SIGTERM signal, it will send a SIGCHLD signal to its parent (server-child) process, which will update the terminated child’s status in the process list. If a user disconnects by sending either the exit or disconnect command from the client, the client-server process sends a SIGTERM signal to all its children that it has created by forking and calling exec. After that the server-child process calls exits and terminates gracefully. Since the server parent process has been handling the SIGCHLD signal, it receives news of the demise of its child (server-child process) and updates the connection list by removing the node related to that connection.