**Flow of code:**

The following report is for a multi-threaded, multi-processed, client-server application. The zip folder contains to files: client.c and server.c, each needs to be linked with the thread library at the time of compilation. On running the server executable file, a port number is given which can then be used by the client on a similar network to connect using the machine's IP and port no. provided. Upon successful connection, a list of all executable commands are displayed at the client end. We are now free to choose the command we wish to use.

There are two threads in the client, one is for reading from the server and the other is for writing to the server.

The server however is a bit more complicated. There is the main thread which is responsible for connection to the client, and completing the 3-way handshake. Forking is done after accept, which allows multiple clients to connect with our server seamlessly. By forking after accept we create a new process, or a sub-server. Each sub-server is in turn multi-threaded, having 4 threads (main thread + 3 others) each. There is a separate thread for running new processes through exec, a thread for all arithmetic operations (no need to create a separate thread for each since the operations are not very computationally intensive).

In order to make the server interactive and to introduce some basic server level commands we've introduced 2 threads in the main or 'master server'. One is used to listen for input from STDIN_FILENO and issue a number of related commands while the other thread is there to listen for input from pipes (this isn't used in the server as such but has been added should there be a need to implement pipes. There is a third thread, the main thread which listens for inputs from client.

Server commands are implemented In a rather unique way, there are no pipes involved. Instead, we send a message to client through msgsock, this message is then read and understood by the client through an if statement, the related operation is then executed on the client end. If it is a command which requires interaction between the sub-server and the master server then this command is passed to the master-server using the client. To make things easier to understand, consider the following example.

a) Send to client from sub-server : kill -all b) client reads kill -all c) client writes kill -all back to the master server d) master server parses input from client and executes the kill processes command.

**Limitations:**

1. Not very good exception handling, may crash if the input from server is unexpected
2. Some server level commands have yet to be implemented.
3. Following is a summary of of the requirements which have been met, which have been partially met and are under the process of getting done:

| TASK | DONE |
|---|---|
| Mathematical Operations ( add , multiply, divide , subtract) | DONE |
| Run ( fork and exec – creation of a new process, return success or failure) | DONE |
| Kill ( by PID , by name , all -pertaining to client's request) | DONE |
| List ( all processes, active processes – PID , name, status, start time, end time, elapsed time, should not contains processes which are terminated after failure in exec-pertaining to client's request, updating lists based on signals sent from within or outside the process) | DONE |
| Implementation of communication mediums between client and server processes, and between main/central server and sub/child servers ( sockets / pipes-take care of record boundaries) | DONE |
| Server must be able to handle multiple clients(pre-fork or fork upon accept) | DONE |
| Interactivity in server using threads for separate functions such as addition, taking input from clients, etc (re-use threads by blocking threads assisted with multiplexed I/O if necessary) | DONE |
| Interactive client by using multi-threading ( thread for taking input from user, another for taking response from the server) | DONE |
| Connect and Disconnect on client | DONE |
| Close sent by client should close all processes associated with that client on the associated server process, and should return to client and close message sock | DONE |
| Exit and help on client | DONE |
| SERVER COMMANDS | |
| List (all processes, display on central server, collect individual clients' lists-use pipes or sockets with identification of client whose list is being printed, use of dynamic arrays/linked list/vector) | PARTIALLY DONE |
| Kill (from central server to it's children to kill a process based on PID, name or kill all processes) | |

| TASK | DONE |
|---|---|
| | DONE |
| Message from central server to child processes (sub servers) to print desired message | DONE |
| Disconnect <Client Name/IP> Disconnect a specific Client after termination all its processes and terminate its server peer and then client should disconnect too | DONE |

```
Ashes and diamonds
Foe and friend
We were all equal in the end.
```