Name: Abdul Moiz Sheikh

ERP: 09395

**Architecture and program flow**

The program consists of the main server, along with client programs attached to it.

The main server starts by initializing the socket which uses the TCP protocol to transfer and receive byte streams from however many clients connected to it.

There are 2 threads running in the server program; the main thread which is in the blocked state for most of the time as its only task it to accept connections from clients and create a new sub server (using the fork() API) which acts as the middle man between the main server and the client. On the other hand there is another thread running which only reads from the screen and parses it accordingly. The thread is basically for running server commands by the user.

The sub-process which is spawned when a new connection request from a client is made consists of the functions:

run <process name>                       -- to execute a program

kill <process PID> -- to kill a process by PID

kill <process name>                         -- to kill a process by name

kill <process name> all                     -- to kill all process by the same name

add <integer1> <integer2> ... <integerN>   -- to add multiple numbers (threaded)

sub <integer1> <integer2> ... <integerN>   -- to subtract multiple numbers (threaded)

mult <integer1> <integer2> ... <integerN>   -- to multiply multiple numbers (threaded)

exit                                       -- to quit

connect <ip> <port>                         -- to connect client

disconnect                                 -- to disconnect client

The functions add, sub, and mult are threaded so that it could cater multiple client requests at the same time.

The process list is maintained by the sub-process and the client list is maintained by the main server.

The client's architecture is similar to that of the main server, with the exception that it has 2 fixed threads which are running indefinitely until the client program exits.

One of these threads is used to constantly read from the socket's file descriptor and the other thread's task is to write to the socket file descriptor as the user proceeds to input commands from the terminal.

Whenever the client disconnects from the server, all of the processes executed from client are killed.

The list of commands the main server can execute are:

list                                       -- to display list of connected clients

list all                                   -- to display list of all currently running processes executed by clients

message <fd of client> <your message>       -- to send a message to a desired client

exit                                       -- to quit now

disconnect                                 -- to disconnect client

**Limitations**

Max clients = 50

Max process = 50

If the above capacity exceeds, the server will access memory other than its own and may malfunction and crash.

Invalid IP and Port from the client when the "connect <ip> <port>" will terminate the client program.

Invalid arguments of the command "message <fd of client> <your message>" will crash the server.

Since the implementation of the client's architecture is different, it may not be compatible to others server program.

| Task | Done |
| --- | --- |
| Mathematical Operations ( add , multiply, divide , subtract) |  |
| Run ( fork and exec – creation of a new process, return success or failure) |  |
| Kill ( by PID , by name , all -pertaining to client’s request) |  |
| 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) |  |
| 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) |  |
| Server must be able to handle multiple clients(pre-fork or fork upon accept) |  |
| 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) |  |
| Interactive client by using multi-threading ( thread for taking input from user, another for taking response from the server) |  |
| Connect and Disconnect on client |  |
| 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 |  |
| Exit and help on client |  |
| 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) |  |
| Kill (from central server to it’s children to kill a process based on PID, name or kill all processes) |  |
| Message from central server to child processes (sub servers) to print desired message |  |
| Disconnect <Client Name/IP> Disconnect a specific Client after termination all its processes and terminate its server peer and then client should disconnect too |  |