

# Final Project Report

## Architecture of the Program

There are 3 processes that are used in the entire project:

1. Socket.c
2. Server.c
3. Client.c

As hinted by the names, the SOCKET process performs the functionality of allowing clients to connect to the created socket – from any machine on the same network – and after accepting, it will **fork** into a worker process which is **exec**'ed into the server code (all relevant file descriptors are passed as parameters). The socket process maintains a list of all active clients, their relevant IP and port numbers, the corresponding worker process ID as well as pipe file descriptors to provide communication between the socket server and worker process (this is done through the CLIENT struct). This process has two kernel level threads – one which is in charge of accepting connections and storing relevant client data into a struct and the second which is waiting for user input of commands from terminal (STDIN), thus achieving server interactivity.

The SERVER process (aka worker process) has a connection with both the socket process (via pipes) and the connecting client (via socket). This means that the server process also has two kernel level threads – one which handles reads from the socket (i.e. communication with the client) and one which handles reads from the pipe (i.e. communication with the parent server: the socket process). This server process does not accept any input from STDIN. The overall structural working of this process is to take a command, tokenize it, and execute a function based on the values received. In the case of running another program (example: gnome-calculator), it is important to note that this process **forks** and **execs** into the specified program. To determine the success of an exec, a pipe using the pipe2() API had been created prior to fork. In combination with the O\_CLOEXEC flag, the pipe will close if the exec succeeds and this can be detected in the parent process.

The CLIENT process is the process run on any machine that initially connects to the socket process (with the relevant command). This connection is then passed on to the middle worker process (i.e. server process). The process must use the connect command in combination with the server IP and port to form the connection. This process may also disconnect from the server without terminating itself via the disconnect command. Without a connection, this process is restricted to serving only 3 commands, namely connect, disconnect, and help. Once connected – the rest of the commands will be served. This process also has two running threads to provide continuous interaction with the user. Meaning the user can enter another command before the output of the previous command has been displayed (i.e. it is still processing with the server). In the case where the socket process kills a client from its own terminal, the client will receive a “Server down – disconnected” message as the middle worker process and all spawned processes are also killed.

A diagram is drawn for ease of understanding on the next page.

## Limitations

A few limitations of the program include:

1. Any processes to be run from the client must have the executable located in /usr/bin.
2. Must place run before process names.
3. Moderate error checking/handling.

## List of Commands

The list of commands used in the project are:

Command	Function
<u>Socket Process</u>	
HELP	Displays list of commands in program during execution
BROADCAST <MSG>	Sends a message to all active clients
LIST CLIENTS	Displays list of connected clients
LIST ALL	Displays list of all active processes of all connected clients
LIST <IP> <PORT>	Displays list of all active processes of specified client
KILL <IP> <PORT>	Kills all active processes and disconnects specified client
<u>Client Process</u>	
HELP	Displays list of commands in program during execution
CONNECT <IP> <PORT>	Connects to server
DISCONNECT	Kills all spawned processes and disconnects from server
ADD <VALUES>	Add given space or comma separated values
SUB <VALUES>	Subtract given space or comma separated values
MUL <VALUES>	Multiply given space or comma separated values
DIV <VALUES>	Divide given space or comma separated values
RUN <PROCESS NAME>	Runs the program specified by name
LIST	Displays list of active processes
LIST ALL	Displays list of all processes
LIST DETAILS	Displays list and details of all processes
KILL <PROCESS ID>	Kills process specified by ID
KILL <PROCESS NAME>	Kills first instance of process specified by name
KILL <PROCESS NAME> ALL	Kills all instances of process specified by name
KILL ALL	Kills all processes
EXIT	Kills all processes and exits program

## Extra Features

Some extra features of the program include:

- Exception handling – will prompt if command is invalid, process not found, etc.
- Client interactivity (not blocked once command given)
- Server interactivity

