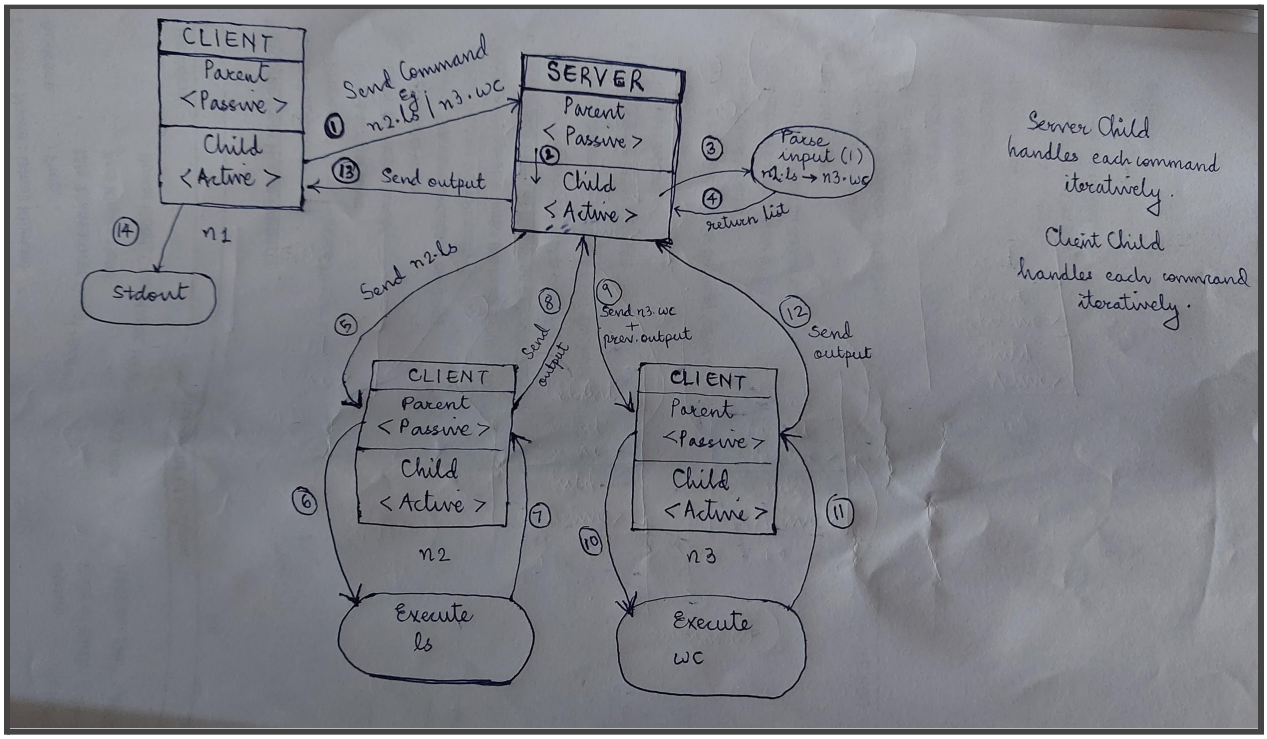
P2 - Cluster Shell

Submitted By : Abdul Kadir Khimani, Ayush Singh, Nayan Khanna
Submitted for : IS F462, Network Programming.
IC : Dr. Hari Babu

Design Decisions :

We implement a cluster shell as in the problem statement with support for process chaining using pipes, use of * operator, and cd command.



Assumptions :

1. Server must be started before starting the clients.
2. Config file contains node information in the form:
   n<num><space><IP Address>
   n<num><space><IP Address>
   ...
3. Includes support for IPv4 and IPv6 addresses making it address independent.
4. All nodes in the config file must be connected. If not, the command fails.
5. **nodes** command gives the nodes present in the system(connected).
6. Use **exit** or **quit** to close the client.

Implementation :

We use '$' as the end marker for every information sent through the socket. This is used as a way of encoding in our system. Input is read into the buffer from any socket until a '$' is encountered. Thus, '$' is appended to every information that is to be sent in one go.

Execution flow :

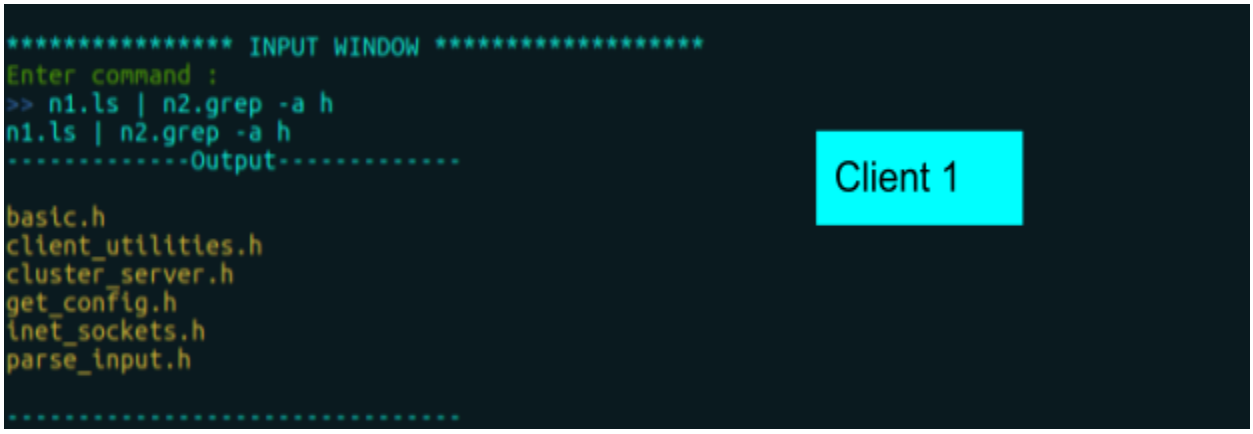
- *Server Side* :
    1. Server has a **passive parent** which accepts new clients on socket L_s. Server, upon accepting a client, gets a new socket C_s for that client, and then creates a child.
    2. **Parent does this concurrently** and can accept multiple connections from different nodes on socket L_s.
    3. The child then parses the input command (possibly chained with pipes) received on socket C_s, and handles each command in the chain iteratively.
    4. For each command in the chain:
        a. The **child creates an active socket** D_s and connects to another client node through socket D_s.
        b. It sends this individual command to another node and receives output from that node through socket D_s, stores output in a buffer, closes socket D_s and proceeds to the next command in the chain.
    5. For the next command in the chain, it repeats step 4, with one small change. In step 4b), it sends the previous output stored in buffer + current command in chain, instead of just sending the command to another node.
    6. Step 5 is repeated until the last command in the chain. After receiving final output from socket D_s, child writes this output to socket C_s so that the requesting node can display final output.

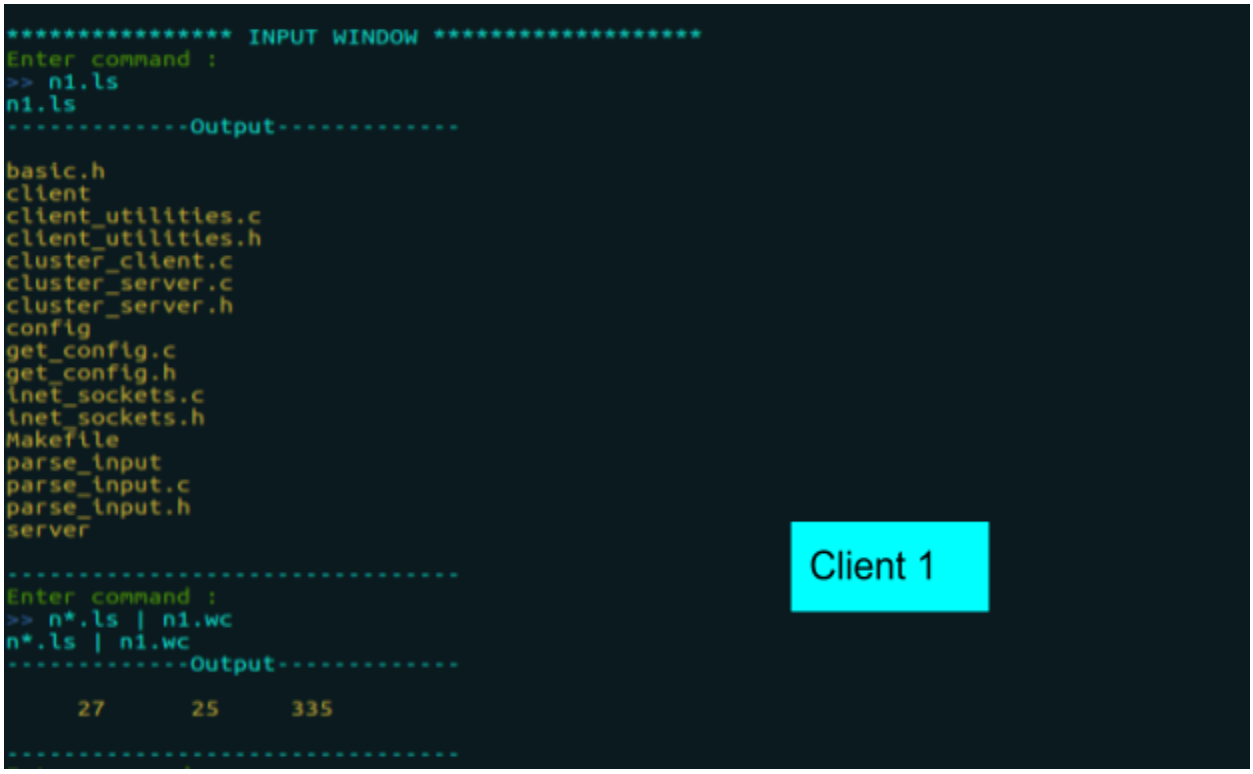
- *Client Side* :
    1. **Client has a passive parent**, which accepts requests from the server on socket L_c to execute commands given by some other node.
    2. Client, upon accepting a server request, gets a new socket D_c for that request. **Parent does this iteratively**, because it needs to handle cd command which requires state preservation across parent and child.
    3. Parent executes the server request by forking a child, using 2 pipes and then writes the output to socket D_c, and closes the socket. The Server Child reads this output from socket D_s as described in step 4b of server side.
    4. **Client also has an active child** process which loops forever reading input commands. It establishes connection to the server through socket C_c. Server accepts this connection on socket L_s as described in step 1a of server side.
    5. The child reads input from command line, sends command to the server, waits for response from server, prints it to the stdout, if possible and then proceeds to next input from stdin.

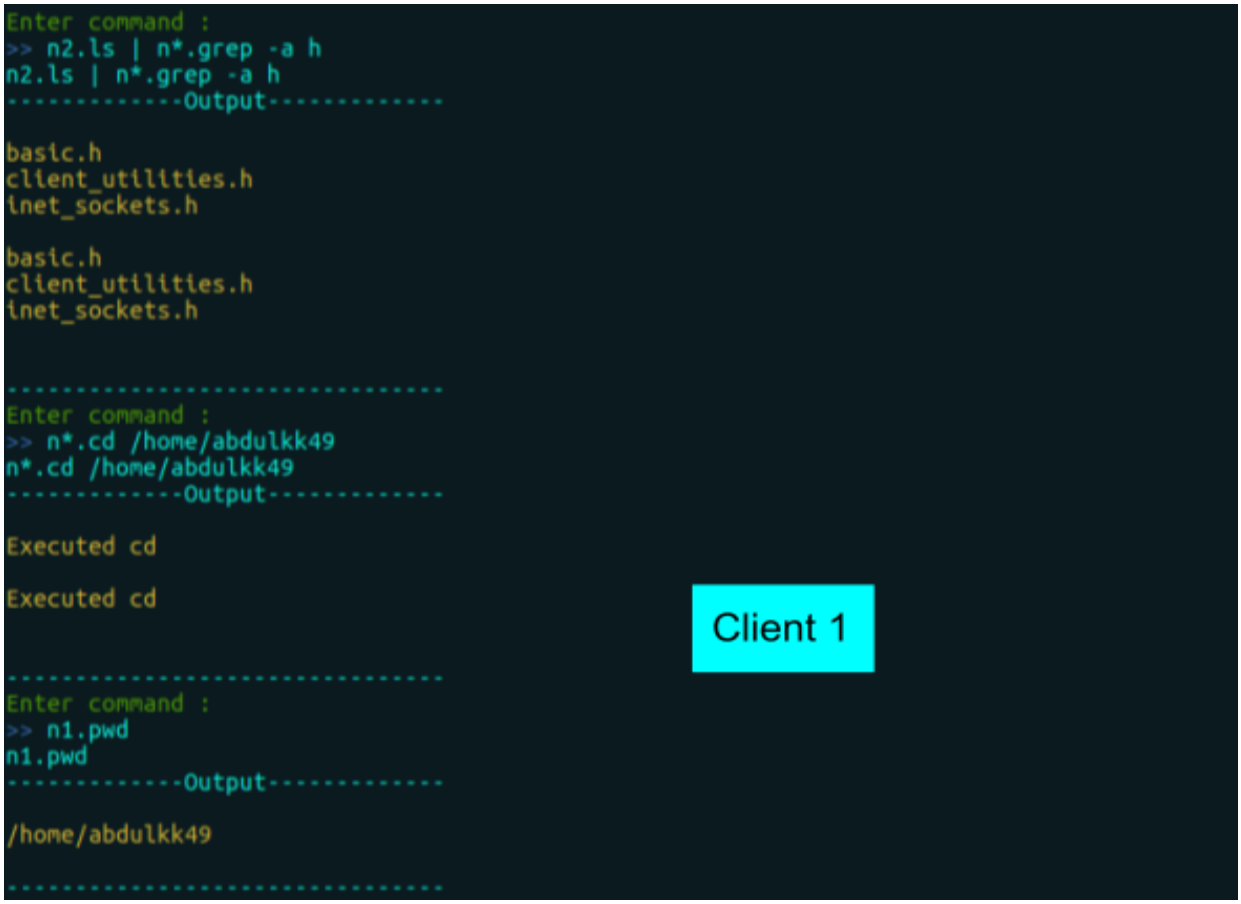
Testing :

Since port forwarding is required for hosts not in the local network, we avoid testing using remote hosts. For testing purposes on the same local machine, as the Client IP address is the same for all clients, We define CLIENT_PORT1(40001) and CLIENT_PORT2(40002) in

inet_sockets.h. We create two copies of client code in two different folders, Client 1 parent runs on port 40001 and Client 2 parent runs on port 40002. Server parent runs on port 50000. Server child connects to port 40001 if command is to be executed on Client 1 and on port 40002 if it is to be executed on Client 2.
For non-testing purposes(clients on different machines), code has specified appropriate lines to be commented in cluster_client.c and cluster_server.c. We can use CLIENT_PORT(40005) defined in inet_sockets.h.






Enter command :
>> n2.ls | n*.grep -a h
n2.ls | n*.grep -a h
------------Output------------
basic.h
client_utilities.h
inet_sockets.h
basic.h
client_utilities.h
inet_sockets.h
Enter command :
>> n*.cd /home/abdulkk49
n*.cd /home/abdulkk49
------------Output------------
Executed cd
Executed cd
Client 1
Enter command :
>> n1.pwd
n1.pwd
------------Output------------
/home/abdulkk49


Enter command :
>> n2.pwd
n2.pwd
------------Output------------
/home/abdulkk49
Client 2