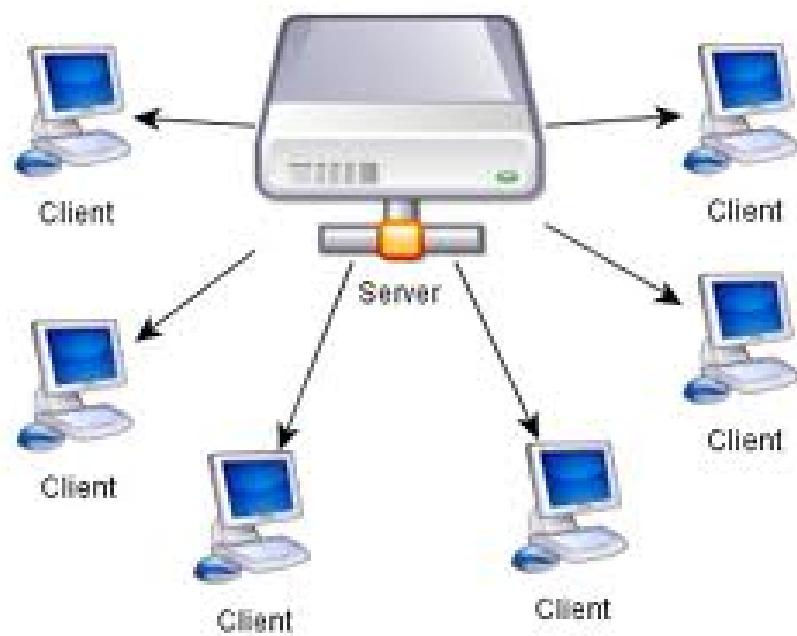


Operating Systems

Assignment - Phase 3: Multi Client - Server Shell



D&S Remote shell

Phase 1:-

Description:

For this assignment, we programmed our own local shell / CLI using C. It stimulates the services of the operating system using system calls. We used features such as process creation, interprocess communication, etc. to execute the same. 15 commands that have been listed below can be run, including pipes (upto three pipes).

How to Compile:

We have used makefile to organize the project's code compilation. To run a makefile code, run the following commands in the terminal.

First, run `$make clean`

Then run `$make` to compile the code

Lastly, run the executable. Run using `./shell`

List of commands:-

1. ls: Command to display the files in a directory
2. touch: Command to create a file
3. mkdir: Command to create a folder
4. pwd: Command to print the current directory
5. rm: Command to remove objects
6. cp: Command to copy
7. rmdir: Command to remove a directory
8. grep: Command to search for a string of characters in a file
9. wc: Command to find the word count
10. find: Command that locates an object
11. mv: Command that moves an object
12. cat: Command that outputs the contents of a file
13. ps: Command that displays the running processes
14. df: Command to display the available disk space
15. whoami: Command to print the name of the current user

Pipe Commands: Used to combine two or more commands. The output of one command is the input of the next command.

1. ls | grep c
2. ls -l | grep d | wc -c
3. cat xyz.txt | grep r | tee abc.txt | wc -l

Test Cases can be found under phase 2 below.

Challenges :-

Along the way, we faced a few challenges we were able to overcome.

1. The first challenge we faced was parsing the entered command(s). We overcame this by using strtok() which broke the string into a series of strings using our given delimiters. We were able to modify this in our pre-built functions to reuse them for extracting commands when pipes were used.
2. Figuring out the right nested forks for triple pipes proved to be a challenging task as well. We were able to overcome this by doing some research as well as using trial and error till we were able to have an error free case.

Phase 2:-

I. Description:

For this assignment, we programmed our own Client-Server remote shell using socket communication in C. It stimulates the services of the Client server network using sockets, ports and local IP addresses to create a remote connection between the client and server. In this phase, we were taking the input from the user client and sending the request to the server. For each input from the client which could either be a command from our listed menu or a pipe command with up to three pipes. Then the server will process the command and send the output or result back to the client which will be printed on the clients terminal screen

II. How to Compile and Run:

We have used makefile to organize the project's code compilation. To run a makefile code, run the following commands in the terminal.

First, run `$make clean` (this is only if you want to run the code again and clear all the .o files)
Then run `$make` to compile the code

Then, the server should be started first using `./server`

Finally start the client using `./client`

III. List of commands that the server:-

1. ls: Command to display the files in a directory
2. touch: Command to create a file
3. mkdir: Command to create a folder
4. pwd: Command to print the current directory
5. rm: Command to remove objects
6. cp: Command to copy
7. rmdir: Command to remove a directory
8. grep: Command to search for a string of characters in a file
9. wc: Command to find the word count
10. find: Command that locates an object
11. mv: Command that moves an object
12. cat: Command that outputs the contents of a file

13. ps: Command that displays the running processes
14. df: Command to display the available disk space
15. whoami: Command to print the name of the current user

Pipe Commands: Used to combine two or more commands. The output of one command is the input of the next command.

1. ls | wc
2. ls -l | grep d | wc -c
3. cat xyz.txt | grep r | tee abc.txt | wc -l

IV. File Contents:

1. client.c : Contains the main function for running the client side application.
2. server.c : Contains the main function for running the server side application.
3. checkpipe.c : Checks the number of pipes in the entered command
4. pipes.c : Performs execution of commands with pipes. There are separate functions for one, two and three pipes in this file.
5. pipesplitter.c : performs parsing and tokenizing of all entered commands. Also performs the task of splitting pipe commands into separate commands.
6. print.c : Prints the user prompt and current directory during each execution.
7. singlecommand.c : Runs execution of the commands which do not use pipes.

V. Functions and Functionalities:

1. Pipe Checking:

Pipe checking is done by the file checkpipe.c which checks if an entered string has pipes. If pipes are present, each section of the pipe is parsed and tokenized separately and sent to the appropriate pipe execution function (single pipe, double pipe, etc...).

```
int checkPipe(char* str1){
    int count = 0;
    // int pipeFlag = 0;
    for(int i=0;i<=(strlen(str1));i++){
        if(str1[i]=='|'){

    
```

```

        ++count;
    }
}

return count;
}

```

2. Parsing and Tokenizing:

We handle parsing and tokenizing of commands in the splitter.c file. Both lineSplitter() and pipeSplitter() functions work in a similar manner. Token sizes are dynamically allocated and the tokens are separated using predefined delimiters for each function. The tokens are returned as an array which is further used by other functions.

```

char **pipeSplitter(char *line, int pipeflag){
    int bufferSize = BUFFER_SIZE, indexPosition = 0;
    char **tokens = malloc(bufferSize * sizeof(char*)); // Array of tokens dynamically allocated
    char *token; // Each token (string between pipes)

    if (!tokens) { // If tokens memory was allocated
        fprintf(stderr, "Token memory allocation error\n");
        exit(EXIT_FAILURE);
    }

    token = strtok(line, PIPE_DELIM); // Strtok breaks string line into a series of tokens using the token delimiters
    provided as PIPE_DELIM Macro
    while (token != NULL) {
        tokens[indexPosition] = token; // place the token into the array of tokens
        indexPosition++; // Increment the index position for the token to be saved in the array
        if (indexPosition >= bufferSize) {
            bufferSize += BUFFER_SIZE;
            tokens = realloc(tokens, bufferSize * sizeof(char*));
            if (!tokens) {
                fprintf(stderr, "Token memory allocation error\n");
                exit(EXIT_FAILURE);
            }
        }
        token = strtok(NULL, PIPE_DELIM); // Strtok NULL tells the function to continue tokenizing the string that
        was passed first
    }
    tokens[indexPosition] = NULL;
    return tokens; // return the array of tokens (arguments pipe separated )
}

```

3. Piped Commands Execution:

Commands which pipes are directed to the functions in the pipes.c file. The pipes make use of the concept of process communication to execute multiple commands using multiple child processes. The number of commands determines the number of pipes used. For instance, a single pipe command requires two separate commands to be run, where the output of the first command becomes the input of the second command. This is done using two child processes (forks), which communicate using a file descriptor. The standard output of the first child is redirected to the second child using the dup2() function. The final output is then sent back to the output stream. Double and Triple pipe functions work similarly, except that the number of file descriptors, child processes, output redirections, etc... increase proportionately with the number of pipes.

```
dup2(socket, STDOUT_FILENO); // reading redirected output of ls through pipe 1  
dup2(socket, STDERR_FILENO);
```

4. Sockets for Server to Client Communication:

This function is the core of our program. It demonstrated communication between two nodes (here the Server and Client) over a network. Various functions like listen(), bind(), accept() and connect() are used to enable a successful connection between the server and client. Messages are sent back and forth between the two nodes using send() and recv().

```
int sock1, sock2, valread;  
struct sockaddr_in address; // structure for storing address  
int addrlen = sizeof(address); // Getting the size off the address  
if ((sock1 = socket(AF_INET, SOCK_STREAM, 0)) == 0)  
{  
    perror("Socket creation failed");  
    exit(EXIT_FAILURE);  
}  
// Setting and defining the address for the socket to bind on  
address.sin_family = AF_INET;  
address.sin_addr.s_addr = htonl(INADDR_ANY);  
address.sin_port = htons(PORT);  
  
if (bind(sock1, (struct sockaddr *)&address, sizeof(address)) < 0)  
{  
    perror("Binding failed");  
    exit(EXIT_FAILURE);  
}
```

```

while(1) // to keep server alive forever
{
    if (listen(sock1, 10) < 0) // defining for number of clients in queue (not necessary here since its single client connection)
    {
        perror("Listen Failed");
        exit(EXIT_FAILURE);
    }
    if (connectFlag == 0)
    {
        printf("Waiting for Client to connect\n"); // Server is listening and
        if ((sock2 = accept(sock1, (struct sockaddr *)&address,
                           (socklen_t *)&addrLen)) < 0)
        {
            perror("accept");
            exit(EXIT_FAILURE);
        }
    }
}

```

5. Client Disconnection and Reconnection

The server waits for the client to connect and accepts commands as long as it is connected to the. When the client leaves using ‘exit’, the connection is disrupted and the server now goes back to waiting for a client to connect. This functions similar to how an actual server and client system would work. This feature is implemented by using a flag which checks which changes to true if the client sends exit. This makes the client exit and sends the server to waiting for a client.

The screenshot shows two terminal windows side-by-side. The left window is titled "Phase2 — server — 66x24" and the right window is titled "Phase2 — client — 82x24".

Server Terminal (Left):

```

Welcome to D&S Remote Shell
-----
Waiting for Client to connect
Successfully Connected: Listening to client.
Server Received: ls
Server Received: pwd
Client has disconnected.
Server Received: exit
Waiting for Client to connect
Successfully Connected: Listening to client.

```

Client Terminal (Right):

```

Successfully connected to Remote shell server
WWelcome to D&S Remote Shell: Client
-----
Enter your linux shell command to be sent to the server
Enter <help> for the list of commands
Enter <exit> to leave the shell

devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase2$>>

```

VI. Test Cases:

Here are our commands with screenshots of both our server and client end.

1. ls:

The screenshot shows two terminal windows side-by-side. The left window is titled "Phase2 — server — 66x24" and the right window is titled "Phase2 — client — 82x24". Both windows display a "Welcome to D&S Remote Shell" message. The server window shows the message "Waiting for Client to connect" and "Successfully Connected: Listening to client.". The client window shows the message "Enter your linux shell command to be sent to the server" and "Enter <help> for the list of commands". Below these messages, both windows display the output of the "ls" command, which lists various files and directories including "Makefile", "Phase 1 Documentation.pdf", "abc.txt", "asdc.s.t", "checkpipe.c", "checkpipe.h", "checkpipe.o", "client", "client.c", "client.h", "client.o", "hooyerah.txt", "main.c", "medaddy.cpp", "mnd.txt", and "pipe.c".

2. touch:

The screenshot shows two terminal windows side-by-side. The left window is titled "Phase2 — server — 66x24" and the right window is titled "Phase2 — client — 82x24". Both windows display a "Welcome to D&S Remote Shell" message. The server window shows the message "Waiting for Client to connect" and "Successfully Connected: Listening to client.". The client window shows the message "Enter your linux shell command to be sent to the server" and "Enter <help> for the list of commands". Below these messages, both windows display the output of the "touch try1.txt" command. The client window also shows the command "devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase2\$>>touch try1.txt". In the bottom half of the image, there is a code editor window showing the source code for the "server.c" file. The code is written in C and includes functions for handling commands like "exit", "singlecommand", "splitter", and "test.op". A specific line of code is highlighted with a yellow box: "if (strcmp(command, exitString) == 0)". The code editor also shows other files in the project structure on the left, including "server.h", "server.o", "singlecommand.h", "singlecommand.o", "splitter.h", "splitter.o", "test.op", and "try1.txt".

3. mkdir:

The screenshot shows two terminal windows side-by-side. The left window, titled "Phase2 — server — 66x24", displays a remote shell session. The right window, titled "Phase2 — client — 82x24", shows the file system structure and the execution of the "mkdir" command.

Server Terminal (Left):

```
Welcome to D&S Remote Shell
-----
Waiting for Client to connect
Successfully Connected: Listening to client.
Server Received: ls
Server Received: touch try1.txt
Server Received: mkdir sampleFolder
[]
```

Client Terminal (Right):

```
read.o
server
server.c
server.h
server.o
singlecommand.c
singlecommand.h
singlecommand.o
splitter.c
splitter.h
splitter.o
test.op
test.txt
testaa.txt
todo.md
ttet.txt
uygbuyb
words.txt

devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase2$>>touch try1.txt
devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase2$>>mkdir sampleFolder
devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase2$>>
```

File Explorer (Bottom Left):

- todo.md
- C checkpipe.c
- C client.c
- C pipe.c
- C singlecommand...
- C splitter.c
- PHASE2
- sampleFolder (highlighted)
- abc.txt
- asdcst

4. pwd:

The screenshot shows two terminal windows side-by-side. The left window, titled "Phase2 — server — 66x24", displays a remote shell session. The right window, titled "Phase2 — client — 82x24", shows the file system structure and the execution of the "pwd" command.

Server Terminal (Left):

```
Welcome to D&S Remote Shell
-----
Waiting for Client to connect
Successfully Connected: Listening to client.
Server Received: ls
Server Received: touch try1.txt
Server Received: mkdir sampleFolder
Server Received: pwd
[]
```

Client Terminal (Right):

```
server.h
server.o
singlecommand.c
singlecommand.h
singlecommand.o
splitter.c
splitter.h
splitter.o
test.op
test.txt
testaa.txt
todo.md
ttet.txt
uygbuyb
words.txt

devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase2$>>touch try1.txt
devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase2$>>mkdir sampleFolder
devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase2$>>pwd
/Users/devkalavadiya/Desktop/OS/Phase2
devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase2$>>
```

5. rm:

The screenshot shows two terminal windows side-by-side. The left window, titled "Phase2 — server — 66x24", displays a remote shell session. The right window, titled "Phase2 — client — 82x24", shows the file system structure and the execution of the "rm" command.

Server Terminal (Left):

```
Welcome to D&S Remote Shell
-----
Waiting for Client to connect
Successfully Connected: Listening to client.
Server Received: ls
Server Received: touch try1.txt
Server Received: mkdir sampleFolder
Server Received: pwd
Server Received: rm abc.txt
Server Received: ls
[]
```

Client Terminal (Right):

```
devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase2$>>rm abc.txt
devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase2$>>ls
Makefile
Phase 1 Documentation.pdf
asdcst
checkpipe.c
checkpipe.h
checkpipe.o
client
client.c
client.h
client.o
hooyerah.txt
main.c
medaddy.cpp
mnd.txt
pipe.c
pipe.h
pipe.o
print.c
print.h
print.o
```

Page Number: 10

6. cp :

Phase2 — server — 66x24

```
Welcome to D&S Remote Shell
-----
Waiting for Client to connect
Successfully Connected: Listening to client.
Server Received: ls
Server Received: touch try1.txt
Server Received: mkdir sampleFolder
Server Received: pwd
Server Received: rm abc.txt
Server Received: ls
Server Received: cp abc.txt abcd.txt
Server Received: touch abc.txt
Server Received: cp abc.txt abcd.txt
[]
```

Phase2 — client — 82x24

```
server.h
server.o
singlecommand.c
singlecommand.h
singlecommand.o
splitter.c
splitter.h
splitter.o
test.op
test.txt
testaa.txt
todo.md
try1.txt
ttet.txt
words.txt

devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase2$>>cp abc.txt abcd.txt
cp: abc.txt: No such file or directory

devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase2$>>touch abc.txt

devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase2$>>cp abc.txt abcd.txt

devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase2$>>
```

File Explorer:

- todo.md
- checkpipe.c
- client.c
- pipe.c
- singlecommand.c
- splitter.c
- PHASE2
- sampleFolder
 - abc.txt
 - abcd.txt
- asdcst.c

abcd.txt is highlighted with a yellow box.

7. rmdir

Phase2 — server — 66x24

```
Welcome to D&S Remote Shell
-----
Waiting for Client to connect
Successfully Connected: Listening to client.
Server Received: ls
Server Received: touch try1.txt
Server Received: mkdir sampleFolder
Server Received: pwd
Server Received: rm abc.txt
Server Received: ls
Server Received: cp abc.txt abcd.txt
Server Received: touch abc.txt
Server Received: cp abc.txt abcd.txt
Server Received: rmdir sampleFolder
[]
```

Phase2 — client — 82x24

```
singlecommand.c
singlecommand.h
singlecommand.o
splitter.c
splitter.h
splitter.o
test.op
test.txt
testaa.txt
todo.md
try1.txt
ttet.txt
words.txt

devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase2$>>cp abc.txt abcd.txt
cp: abc.txt: No such file or directory

devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase2$>>touch abc.txt

devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase2$>>cp abc.txt abcd.txt

devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase2$>>rmdir sampleFolder

devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase2$>>
```

File Explorer:

- todo.md
- checkpipe.c
- client.c
- pipe.c
- singlecommand.c
- splitter.c
- PHASE2
- abcd.txt
- asdcst.c

abcd.txt is highlighted with a yellow box.

The folder can was deleted and can not be seen on the highlighted box area. (Previously present in the image for cp)

8. wc:

The screenshot shows two terminal windows side-by-side. The left window is titled "Phase2 — server — 66x24" and the right window is titled "Phase2 — client — 82x24". Both windows show a "Welcome to D&S Remote Shell" message. The client window has a highlighted box around the directory listing and the command entry area.

```
Welcome to D&S Remote Shell
-----
Waiting for Client to connect
Successfully Connected: Listening to client.
Server Received: ls
Server Received: touch try1.txt
Server Received: mkdir sampleFolder
Server Received: pwd
Server Received: rm abc.txt
Server Received: ls
Server Received: cp abc.txt abcd.txt
Server Received: touch abc.txt
Server Received: cp abc.txt abcd.txt
Server Received: rmdir sampleFolder
Server Received: wc pipe.c
[]

splitter.c
splitter.h
splitter.o
test.op
test.txt
testaa.txt
todo.md
try1.txt
ttet.txt
words.txt

devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase2$>>cp abc.txt abcd.txt
cp: abc.txt: No such file or directory

devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase2$>>touch abc.txt

devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase2$>>cp abc.txt abcd.txt

devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase2$>>rmdir sampleFolder

devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase2$>>wc pipe.c
      227      827     8684 pipe.c

devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase2$>>[
```

9. find:

The screenshot shows two terminal windows side-by-side. The left window is titled "Phase2 — server — 66x24" and the right window is titled "Phase2 — client — 82x24". Both windows show a "Welcome to D&S Remote Shell" message. The client window has a highlighted box around the directory listing and the command entry area.

```
Welcome to D&S Remote Shell
-----
Waiting for Client to connect
Successfully Connected: Listening to client.
Server Received: ls
Server Received: touch try1.txt
Server Received: mkdir sampleFolder
Server Received: pwd
Server Received: rm abc.txt
Server Received: ls
Server Received: cp abc.txt abcd.txt
Server Received: touch abc.txt
Server Received: cp abc.txt abcd.txt
Server Received: rmdir sampleFolder
Server Received: wc pipe.c
Server Received: find abcd.txt
[]

test.op
test.txt
testaa.txt
todo.md
try1.txt
ttet.txt
words.txt

devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase2$>>cp abc.txt abcd.txt
cp: abc.txt: No such file or directory

devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase2$>>touch abc.txt

devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase2$>>cp abc.txt abcd.txt

devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase2$>>rmdir sampleFolder

devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase2$>>wc pipe.c
      227      827     8684 pipe.c

devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase2$>>find abcd.txt
abcd.txt

devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase2$>>[
```

10. mv :

The screenshot shows two terminal windows and a file browser. The left terminal (server) logs the movement of files from 'sampleFolder' to 'Dest'. The right terminal (client) shows the commands being issued. A file browser window at the bottom left displays a directory structure with a folder named 'Dest/sampleFolder' highlighted.

```
Welcome to D&S Remote Shell
-----
Waiting for Client to connect
Successfully Connected: Listening to client.
Server Received: ls
Server Received: touch try1.txt
Server Received: mkdir sampleFolder
Server Received: pwd
Server Received: rm abc.txt
Server Received: ls
Server Received: cp abc.txt abcd.txt
Server Received: touch abc.txt
Server Received: cp abc.txt abcd.txt
Server Received: rmdir sampleFolder
Server Received: wc pipe.c
Server Received: find abcd.txt
Server Received: mkdir sampleFolder
Server Received: mkdir Dest
Server Received: mv sampleFolder Dest
[]

words.txt
devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase2$>>cp abc.txt abcd.txt
cp: abc.txt: No such file or directory
devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase2$>>touch abc.txt
devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase2$>>cp abc.txt abcd.txt
devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase2$>>rmdir sampleFolder
devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase2$>>wc pipe.c
227 827 8684 pipe.c
devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase2$>>find abcd.txt
abcd.txt
devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase2$>>mkdir sampleFolder
devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase2$>>mkdir Dest
devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase2$>>mv sampleFolder Dest
devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase2$>>
```

```
(socklen_t *)&addrLen)) < 0)
{
    perror("accept");
    exit(EXIT_FAILURE);
}
connectFlag = 1; // Setting connection flag to be true
printf(">Successfully Connected: Listening to client.\n");
}
else
{
    // bzero(command, BUFFER_SIZE);
```

As it can be seen the sample folder was moved inside the Dest directory

11. cat :

The screenshot shows two terminal windows and a file browser. The left terminal (server) logs the execution of 'cat checkpipe.c'. The right terminal (client) shows the command being issued. A file browser window at the bottom left displays a directory structure with a file named 'checkpipe.c' highlighted.

```
Welcome to D&S Remote Shell
-----
Waiting for Client to connect
Successfully Connected: Listening to client.
Server Received: ls
Server Received: touch try1.txt
Server Received: mkdir sampleFolder
Server Received: pwd
Server Received: rm abc.txt
Server Received: ls
Server Received: cp abc.txt abcd.txt
Server Received: touch abc.txt
Server Received: cp abc.txt abcd.txt
Server Received: rmdir sampleFolder
Server Received: wc pipe.c
Server Received: find abcd.txt
Server Received: mkdir sampleFolder
Server Received: mkdir Dest
Server Received: mv sampleFolder Dest
Server Received: cat checkpipe.c
[]

devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase2$>>mv sampleFolder Dest
devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase2$>>cat checkpipe.c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <string.h>
#include <sys/wait.h>

int checkPipe(char* str1){
    int count = 0;
    // int pipeFlag = 0;
    for(int i=0;i<=(strlen(str1));i++){
        if(str1[i]=='|'){
            ++count;
        }
    }
    return count;
}
return 0;
}
```

12. ps:

The image shows two terminal windows side-by-side. The left window, titled "Phase2 — server — 66x24", displays a "D&S Remote Shell" session. The right window, titled "Phase2 — client — 82x24", shows a C program named "checkpipe.c".

```
Welcome to D&S Remote Shell
-----
Waiting for Client to connect
Successfully Connected: Listening to client.
Server Received: ls
Server Received: touch try1.txt
Server Received: mkdir sampleFolder
Server Received: pwd
Server Received: rm abc.txt
Server Received: ls
Server Received: cp abc.txt abcd.txt
Server Received: touch abc.txt
Server Received: cp abc.txt abcd.txt
Server Received: rmdir sampleFolder
Server Received: wc pipe.c
Server Received: find abcd.txt
Server Received: mkdir sampleFolder
Server Received: mkdir Dest
Server Received: mv sampleFolder Dest
Server Received: cat checkpipe.c
[]

devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase2$>>mv sampleFolder Dest
devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase2$>>cat checkpipe.c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <string.h>
#include <sys/wait.h>

int checkPipe(char* str1){
    int count = 0;
    // int pipeFlag = 0;
    for(int i=0;i<=(strlen(str1));i++){
        if(str1[i]=='|'){
            ++count;
        }
    }
    return count;
    // return 0;
}
devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase2$>>[
```

13. df:

The image shows two terminal windows side-by-side. The left window, titled "Phase2 — server — 66x24", displays a "D&S Remote Shell" session. The right window, titled "Phase2 — client — 132x24", shows disk usage information.

```
Welcome to D&S Remote Shell
-----
Waiting for Client to connect
Successfully Connected: Listening to client.
Server Received: ls
Server Received: touch try1.txt
Server Received: mkdir sampleFolder
Server Received: pwd
Server Received: rm abc.txt
Server Received: ls
Server Received: cp abc.txt abcd.txt
Server Received: touch abc.txt
Server Received: cp abc.txt abcd.txt
Server Received: rmdir sampleFolder
Server Received: wc pipe.c
Server Received: find abcd.txt
Server Received: mkdir sampleFolder
Server Received: mkdir Dest
Server Received: mv sampleFolder Dest
Server Received: cat checkpipe.c
Server Received: ps
Server Received: df
[]

32597 ttys001 0:00.00 ./client
70476 ttys001 0:00.02 ./client
1217 ttys002 0:00.01 ./server
28140 ttys002 0:02.36 zsh
32468 ttys002 0:00.01 ./server
70475 ttys002 0:00.02 ./server
51790 ttys004 0:00.24 /bin/zsh -l
51795 ttys006 0:00.12 /bin/zsh -l

devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase2$>>df
Filesystem      512-blocks   Used Available Capacity  iused   ifree %iused Mounted on
/dev/disk3s1s1      965595304 30767744 657294744   5%  577263 3286473720   0%   /
devfs                  411       411       0  100%    712       0 100%  /dev
/dev/disk3s6      965595304 8388664 657294744   2%     4 3286473720   0%  /System/Volumes/VM
/dev/disk3s2      965595304 856848 657294744   1%   1586 3286473720   0%  /System/Volumes/Preboot
/dev/disk3s4      965595304 547080 657294744   1%   138 3286473720   0%  /System/Volumes/Update
/dev/disk1s2      1024000 12328 986344   2%     3 4931720   0%  /System/Volumes/xarts
/dev/disk1s1      1024000 14928 986344   2%     28 4931720   0%  /System/Volumes/iSCSPreboot
/dev/disk1s3      1024000 680 986344   1%     35 4931720   0%  /System/Volumes/Hardware
/dev/disk3s5      965595304 265775880 657294744   29% 1601251 3286473720   0%  /System/Volumes/Data
map auto_home          0       0       0  100%     0       0 100%  /System/Volumes/Data/home
//DRIVE@localhost:59858/Google%20Drive 965595304 341165304 624430000   36% 42645661 78053750   35%  /Volumes/GoogleDrive

devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase2$>>[
```

14

14. who am i:

The image shows two terminal windows side-by-side. The left window, titled 'Phase2 — server — 66x24', displays a session log from a D&S Remote Shell. It shows the server waiting for a client connection, receiving various commands like 'ls', 'touch try1.txt', 'mkdir sampleFolder', etc., and responding with file operations like 'cp abc.txt abcd.txt'. The right window, titled 'Phase2 — client — 84x24', shows the client's perspective. It lists system volumes and their details, then executes the 'whoami' command, which returns 'devkalavadiya'. Both windows have a dark background with light-colored text.

```
Welcome to D&S Remote Shell
-----
>Waiting for Client to connect
>Successfully Connected: Listening to client.
>Server Received: ls
>Server Received: touch try1.txt
>Server Received: mkdir sampleFolder
>Server Received: pwd
>Server Received: rm abc.txt
>Server Received: ls
>Server Received: cp abc.txt abcd.txt
>Server Received: touch abc.txt
>Server Received: cp abc.txt abcd.txt
>Server Received: rmdir sampleFolder
>Server Received: wc pipe.c
>Server Received: find abcd.txt
>Server Received: mkdir sampleFolder
>Server Received: mkdir Dest
>Server Received: mv sampleFolder Dest
>Server Received: cat checkpipe.c
>Server Received: ps
>Server Received: df
>Server Received: whoami
[]
```

```
Phase2 — client — 84x24
2      0 100% /dev
/dev/disk3s6          965595304  8388664 6572
4 3286473720    0% /System/Volumes/VM
/dev/disk3s2          965595304  856848 6572
6 3286473720    0% /System/Volumes/Preboot
/dev/disk3s4          965595304  547080 6572
8 3286473720    0% /System/Volumes/Update
/dev/disk1s2          1024000   12328  9
3 4931720     0% /System/Volumes/xarts
8 4931720     0% /System/Volumes/iSCPReboot
/dev/disk1s3          1024000   1024000 680 9
5 4931720     0% /System/Volumes/Hardware
/dev/disk3s5          965595304  265775880 6572
1 3286473720    0% /System/Volumes/Data
map auto_home          0       0
0      0 100% /System/Volumes/Data/home
//DRIVE@localhost:59858/Google%20Drive  965595304 341165304 6244
1 78053750    35% /Volumes/GoogleDrive
devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase2$>whoami
devkalavadiya
devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase2$>[]
```

15. Single pipe execution:

The image shows two terminal windows. The left window, 'Phase2 — wc - server — 66x24', shows the server receiving a command and executing it. The right window, 'Phase2 — client — 84x24', shows the client sending the command and receiving the output. The command executed is 'ls | wc', which outputs file statistics. Both windows have a dark background with light-colored text.

```
Welcome to D&S Remote Shell
-----
>Waiting for Client to connect
>Successfully Connected: Listening to client.
>Server Received: ls | wc
[]
```

```
Phase2 — client — 84x24
WWelcome to D&S Remote Shell: Client
-----
Enter your linux shell command to be sent to the server
Enter <help> for the list of commands
Enter <exit> to leave the shell

Successfully connected to Remote shell server
devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase2$>ls | wc
      43      45      420
devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase2$>[]
```

16. Double pipes execution:

The image shows two terminal windows. The left window, 'Phase2 — grep < server — 66x24', shows the server receiving a command and executing it. The right window, 'Phase2 — client — 84x24', shows the client sending the command and receiving the output. The command executed is 'ls -l | grep d | wc -c', which counts the number of files matching 'd'. Both windows have a dark background with light-colored text.

```
Welcome to D&S Remote Shell
-----
-->Waiting for Client to connect
>Successfully Connected: Listening to client.
>Server Received: ls -l | grep d | wc -c
[]
```

```
Phase2 — client — 84x24
WWelcome to D&S Remote Shell: Client
-----
Enter your linux shell command to be sent to the server
Enter <help> for the list of commands
Enter <exit> to leave the shell

Successfully connected to Remote shell server
devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase2$>ls -l | grep d | wc -c
      2828
devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase2$>[]
```

17. Triple pipes execution:

```
Phase2 - tee < server - 66x24
Welcome to D&S Remote Shell
-----
--Waiting for Client to connect
>Successfully Connected: Listening to client.
>Server Received: ls -l | grep d | wc -c
>Server Received: cat checkpipe.c | grep i | tee abc.txt | wc -l
[]

Phase2 - client - 100x24
Welcome to D&S Remote Shell: Client
-----
Enter your linux shell command to be sent to the server
Enter <help> for the list of commands
Enter <exit> to leave the shell
Successfully connected to Remote shell server
devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase2$>ls -l | grep d | wc -c
2828
devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase2$>cat checkpipe.c | grep i | tee abc.txt | wc -l
11
devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase2$>[
```

18. Error Handling Test Cases:

- When creating same file name error message is relayed back to client

```
Phase2 - mkdir < server - 83x24
Welcome to D&S Remote Shell: Server
-----
--Waiting for Client to connect
>Successfully Connected: Listening to client.
>Server Received: ls
>Server Received: ls -l
>Server Received: pwd
>Server Received: ls | wc
>Server Received: ls | wc | wc
>Server Received: mkdir test
>Server Received: mkdir test

Phase2 - client - 70x24
-rw-r--r-- 1 devkalavadiya staff 55 Apr 6 18:52 server.h
-rw-r--r-- 2 devkalavadiya staff 4536 Apr 7 23:14 server.o
-rw-r--r--@ 1 devkalavadiya staff 663 Apr 7 22:59 singlecommand.h
-rw-r--r-- 1 devkalavadiya staff 188 Apr 5 01:56 singlecommand.o
-rw-r--r-- 2 devkalavadiya staff 1112 Apr 7 23:14 splitter.c
-rw-r--r-- 1 devkalavadiya staff 3123 Apr 7 22:58 splitter.h
-rw-r--r-- 2 devkalavadiya staff 128 Apr 7 12:55 splitter.o
-rw-r--r-- 1888 Apr 7 23:14 splitter.o

devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase2$>>pwd
/Users/devkalavadiya/Desktop/OS/Phase2

devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase2$>>ls | wc
27 27 260

devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase2$>>ls | wc | wc
1 3 25

devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase2$>>mkdir test
devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase2$>>mkdir test
mkdir: test: File exists

devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase2$>[
```

- Null argument in a piped commands

```
Phase2 - server - 83x24
Welcome to D&S Remote Shell: Server
-----
--Waiting for Client to connect
>Successfully Connected: Listening to client.
>Server Received: |
>Server Received: ||
>Server Received: |||
[]

Phase2 - client - 70x24
Welcome to D&S Remote Shell: Client
-----
Enter your linux shell command to be sent to the server
Enter <help> for the list of commands
Enter <exit> to leave the shell
Successfully connected to Remote shell server
devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase2$>|
>Null Argument was detected. Please try again
devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase2$>| |
>Null Argument was detected. Please try again
devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase2$>|||
>Null Argument was detected. Please try again
devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase2$>[
```

c. Piped command without all arguments

The screenshot shows two terminal windows. The left window (server) has the title "Phase2 — ls < server — 83x24". It displays the following text:

```
Welcome to D&S Remote Shell: Server
-----
>Waiting for Client to connect
>Successfully Connected: Listening to client.
>Server Received: ls
>Server Received: ls -l
>Server Received: pwd
>Server Received: ls | wc
>Server Received: ls | wc | wc
>Server Received: mkdir test
>Server Received: mkdir test
```

The right window (client) has the title "Phase2 — client — 70x24". It displays the following text:

```
devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase2$>>pwd
/Users/devkalavadiya/Desktop/OS/Phase2
devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase2$>>ls | wc
    27      27     260
devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase2$>>ls | wc | wc
        1      3     25
devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase2$>>mkdir test
devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase2$>>mkdir test
mkdir: test: File exists

devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase2$>>ls | grep
usage: grep [-abcdDEFGhIiJlMmnOopqRSsUVvwXxZz] [-A num] [-B num] [-C
[num]]
          [-e pattern] [-f file] [--binary-files=value] [--color=when]
          [--context[=num]] [--directories=action] [--label] [--line-bu
fered]
          [--null] [pattern] [file ...]

devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase2$>>
```

d. Enter without any command doesn't break code :)

The screenshot shows two terminal windows. The left window (server) has the title "Phase2 — server — 83x24". It displays the following text:

```
Welcome to D&S Remote Shell: Server
-----
>Waiting for Client to connect
>Successfully Connected: Listening to client.
>Server Received: |
>Server Received: ||
>Server Received: |||
[]
```

The right window (client) has the title "Phase2 — client — 70x24". It displays the following text:

```
devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase2$>|
>Null Argument was detected. Please try again
devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase2$>| |
>Null Argument was detected. Please try again
devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase2$>|||
>Null Argument was detected. Please try again
devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase2$>
devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase2$>
devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase2$>
devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase2$>
devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase2$>
devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase2$>
devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase2$>
devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase2$>
```

VII. Challenges :-

We ran into a lot of bugs for this Phase. Most were due to the complex working of the c programming language. A few of them are detailed below.

1. The first issue after we connected successfully was the lagging of buffer outputs between the client and server. For instance, when we entered ls, the first command worked as expected and for the next command onwards the buffer was lagging because everything from the server was being output into the client.

We realized this issue was created by an incorrect placement of the dup2() function which. Placing it in the parent process was creating a problem. We moved it to the child area and fixed the issue.

2. Another challenge we faced was the buffer size problem. The commands output would queue up into the buffer but would not print correctly and commands would be sent in chunks when printing on the client. In order to fix this we started by clearing the buffer and the input before sending and receiving from server and client. Additionally, to fix the problem of large output getting sent as chunks, we added a recv() at the start with a `MSG_DONTWAIT` argument which didn't block the client from running.
3. We faced a bug where the client kept quitting after entering piped commands. Upon careful debugging we realized that the file descriptors and sockets were not closed properly and in the right area. With a little trial and error we figured this part out successfully.

PHASE 3:-

I. File Contents:

- client.c : Contains the main function for running the client side application.
- server.c : Contains the main function for running the server side application.
- checkpipe.c : Checks the number of pipes in the entered command
- pipes.c : Performs execution of commands with pipes. There are separate functions for one, two and three pipes in this file.
- pipesplitter.c : performs parsing and tokenizing of all entered commands. Also performs the task of splitting pipes commands into separate commands.
- print.c : Prints the user prompt and current directory during each execution.
- singlecommand.c : Runs execution of the commands which do not use pipes.

II. How to Compile and Run:

We have used makefile to organize the project's code compilation. To run a makefile code, run the following commands in the terminal.

First, run `$make clean` (this is only if you want to run the code again and clear all the .o files)

Then run `$make` to compile the code

Then, the server should be started first using `$./server`

Finally start clients using `$./client`

III. Functions and Functionalities:

1. Multithreading for multiple Clients

```
while (1) // to keep server alive forever
{
    printf(">Waiting for Client to connect\n"); // Server is listening and waiting for
connection
    if ((sock2 = accept(sock1, (struct sockaddr *)&address,
                        (socklen_t *)&addrlen)) < 0)
    {
        perror("accept");
        exit(EXIT_FAILURE);
    }
    // connectFlag = 1; // Setting connection flag to be true
    printf(">Successfully Connected: Listening to client.\n");
    int rc;                                // return value from pthread_create
to check if new thread is created successfully                         */
    pthread_t thread_id;                    // thread's ID (just an integer,
typedef unsigned long int) to identify new thread
    int *new_socket = (int *)malloc(sizeof(int)); // for passing safely the integer
socket to the thread
    if (new_socket == NULL)
    {
        fprintf(stderr, "Couldn't allocate memory for thread new socket argument.\n");
        free(new_socket);
        exit(EXIT_FAILURE);
    }
    *new_socket = sock2;
    printf("New Client has Connected!\n");
    // create a new thread that will handle the communication with the newly accepted
client
    rc = pthread_create(&thread_id, NULL, HandleClient, new_socket);
    if (rc) // if rc is > 0 imply could not create new thread
    {
        printf("\n ERROR: return code from pthread_create is %d \n", rc);
        free(new_socket);
        exit(EXIT_FAILURE);

    }
}
```

2. Exiting Client using “exit” and ^C:

```
void clientExitHandler(int sig_num)
{
    char *exitString = "exit";
    // printf("\nExiting shell successfully \n");
    send(sock, exitString, strlen(exitString) + 1, 0); // sending exit message to
server
    close(sock); // close the socket/end the connection
    printf("\n Exiting client. \n");
    fflush(stdout); // force to flush any data in buffers to the file descriptor
of standard output,, a pretty convinent function
    exit(0);
}
```

```
int main() // main function
{
    printf("\t\t Welcome to D&S Remote Shell: Multi Server\n");
    printf("\t -----\n");
    signal(SIGINT, serverExitHandler);
}
```

IV. Test Cases:

Here are our commands with screenshots of both our server and client end.

1. Single-client connect:

```
Phase3 — server — 80x45
-----
Welcome to D&S Remote Shell: Multi Server
-----
>Waiting for Client to connect
>Successfully Connected: Listening to client.
New Client has Connected!
>Waiting for Client to connect
Handling new client in a thread using socket: 4
Listening to client..
```

```
Phase3 — client — 80x22
-----
Welcome to D&S Remote Shell: Client
-----
Enter your linux shell command to be sent to the server
Enter <help> for the list of commands
Enter <exit> to leave the shell
-----
Successfully connected to Remote shell server
devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase3$>
```

```
Phase3 — -zsh — 80x20
-----
Last login: Thu Apr 28 17:38:59 on ttys008
devkalavadiya@Devs-MacBook-Pro-2 ~ % cd Desktop/OS/Phase3
devkalavadiya@Devs-MacBook-Pro-2 Phase3 %
```

2. Multi-client connect:

The screenshot shows two terminal windows titled "Phase3 - client" and "Phase3 - client" side-by-side. Both windows are connected to a single "Phase3 - server" window. The server window displays log messages indicating new client connections and socket handling. The clients show a standard remote shell prompt with help commands.

```
Welcome to D&S Remote Shell: Multi Server
-----
>Waiting for Client to connect
>Successfully Connected: Listening to client.
New Client has Connected!
>Waiting for Client to connect
Handling new client in a thread using socket: 4
Listening to client..
>Successfully Connected: Listening to client.
New Client has Connected!
>Waiting for Client to connect
Handling new client in a thread using socket: 5
Listening to client..
```

```
Welcome to D&S Remote Shell: Client
-----
Enter your linux shell command to be sent to the server
Enter <help> for the list of commands
Enter <exit> to leave the shell

Successfully connected to Remote shell server
devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase3$>[]
```

```
Welcome to D&S Remote Shell: Client
-----
Enter your linux shell command to be sent to the server
Enter <help> for the list of commands
Enter <exit> to leave the shell

Successfully connected to Remote shell server
devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase3$>[]
```

3. Client disconnect:

The screenshot shows two terminal windows titled "Phase3 - server" and "Phase3 - zsh" side-by-side. The "Phase3 - server" window logs client connections and disconnections. The "Phase3 - zsh" window shows a client disconnecting by entering "exit". The client window shows a standard remote shell prompt with help commands.

```
Welcome to D&S Remote Shell: Multi Server
-----
>Waiting for Client to connect
>Successfully Connected: Listening to client.
New Client has Connected!
>Waiting for Client to connect
Handling new client in a thread using socket: 4
Listening to client..
>Successfully Connected: Listening to client.
New Client has Connected!
>Waiting for Client to connect
Handling new client in a thread using socket: 5
Listening to client..
>Client on socket 4 has disconnected.
>Client on socket 5 has disconnected.
```

```
Welcome to D&S Remote Shell: Client
-----
Enter your linux shell command to be sent to the server
Enter <help> for the list of commands
Enter <exit> to leave the shell

Successfully connected to Remote shell server
devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase3$>exit
Exiting shell successfully
devkalavadiya@Devs-MacBook-Pro-2 Phase3 % []
```

```
Welcome to D&S Remote Shell: Client
-----
Enter your linux shell command to be sent to the server
Enter <help> for the list of commands
Enter <exit> to leave the shell

Successfully connected to Remote shell server
devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase3$>^C
Exiting client.
devkalavadiya@Devs-MacBook-Pro-2 Phase3 % []
```

4. Client reconnect:

The image shows three terminal windows. The top-left window is titled "Phase3 - server - 82x45" and displays the server's log output. It shows the server waiting for clients, successfully connecting them, and handling new clients in threads using sockets 4 and 5. The top-right window is titled "Phase3 - client - 80x22" and shows a client connected to the server, with the command "pwd" being entered. The bottom-right window is also titled "Phase3 - client - 80x22" and shows the client's prompt after the command was sent.

```
Welcome to D&S Remote Shell: Multi Server
-----
>Waiting for Client to connect
>Successfully Connected: Listening to client.
New Client has Connected!
>Waiting for Client to connect
Handling new client in a thread using socket: 4
Listening to client..
>Successfully Connected: Listening to client.
New Client has Connected!
>Waiting for Client to connect
Handling new client in a thread using socket: 5
Listening to client..
>Server Received using socket 4: pwd
>Client on socket 4 has disconnected.
>Successfully Connected: Listening to client.
New Client has Connected!
>Waiting for Client to connect
Handling new client in a thread using socket: 4
Listening to client..
>Server Received using socket 5: pwd
```

```
Welcome to D&S Remote Shell: Client
-----
Enter your linux shell command to be sent to the server
Enter <help> for the list of commands
Enter <exit> to leave the shell
Successfully connected to Remote shell server
devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase3$>
```

```
Welcome to D&S Remote Shell: Client
-----
Enter your linux shell command to be sent to the server
Enter <help> for the list of commands
Enter <exit> to leave the shell
Successfully connected to Remote shell server
devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase3$>pwd
/Users/devkalavadiya/Desktop/OS/Phase3
devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase3$>
```

5. Simultaneous code execution:

The image shows three terminal windows. The top-left window is titled "Phase3 - server - 80x45" and displays the server's log output. The top-right window is titled "Phase3 - client - 80x22" and shows a client connected to the server, with the command "ls" being entered. The bottom-right window is also titled "Phase3 - client - 80x20" and shows the client's prompt after the command was sent.

```
Welcome to D&S Remote Shell: Multi Server
-----
>Waiting for Client to connect
>Successfully Connected: Listening to client.
New Client has Connected!
>Waiting for Client to connect
Handling new client in a thread using socket: 4
Listening to client..
>Successfully Connected: Listening to client.
New Client has Connected!
>Waiting for Client to connect
Handling new client in a thread using socket: 5
Listening to client..
>Server Received using socket 4: ls
>Server Received using socket 5: pwd
```

```
Welcome to D&S Remote Shell: Client
-----
Enter your linux shell command to be sent to the server
Enter <help> for the list of commands
Enter <exit> to leave the shell
Successfully connected to Remote shell server
devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase3$>ls
Makefile
checkpipe.c
checkpipe.h
checkpipe.o
client.c
client.h
client.o
pipe.c
pipe.h
pipe.o
print.c
print.h
print.o
read.c
read.h
read.o
server.c
server.h
server.o
```

```
Welcome to D&S Remote Shell: Client
-----
Enter your linux shell command to be sent to the server
Enter <help> for the list of commands
Enter <exit> to leave the shell
Successfully connected to Remote shell server
devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase3$>pwd
/Users/devkalavadiya/Desktop/OS/Phase3
devkalavadiya:/Users/devkalavadiya/Desktop/OS/Phase3$>
```

6. Specific Error Case : Running Client in client

The screenshot shows three terminal windows titled "Phase3 — server — 80x45", "Phase3 — client — 80x22", and "Phase3 — client — 80x20".

- Server Terminal:** Displays log messages from the server process. It shows the server waiting for clients, successfully connecting to them, and handling multiple clients simultaneously. It also shows the server receiving file operations (ls and pwd) from clients.
- Client Terminal 1 (80x22):** Displays the client's perspective. It shows the client sending commands (ls and pwd) to the server and receiving directory listings back. It also shows the client attempting to run its own client command, which fails with an error message: "Cannot open client in client!"
- Client Terminal 2 (80x20):** Displays another client's perspective. It shows the client successfully connecting to the server and running commands (pwd and server). It also shows the client attempting to run its own client command, which fails with an error message: "Cannot open server in client!"

V. Challenges :-

The challenges faced in this phase were easier to analyze and debug.

1. The first error we came across after merging our Phase 2 code with a multi-client code was while exiting our client. It was causing the server to spam a null character. We fixed this by closing the sockets in the right places and in the right order. The incorrect termination of the threads might have also contributed to the error which we handled correctly.
2. Another thing we struggled with was to get the thread to terminate and have the closing of the socket at the end so that the particular client will successfully quit and disconnect. Once we had this working we realized that the client could reconnect to the server and was assigned the same thread number.
3. The exiting of the client using two different methods, ^c and “exit” gave us different results, which was an error. We figured this problem was caused by using the clientExitHandler() function that worked differently from our exit, and debugged it appropriately.