**Client-Server I/O using Shared Sockets - Project 2**

**Team Members**

* Jimit Thobhani

UAlbany ID: 001427969

Email: jthobhani@albany.edu

* Dhruv Patel

UAlbany ID: 001457291

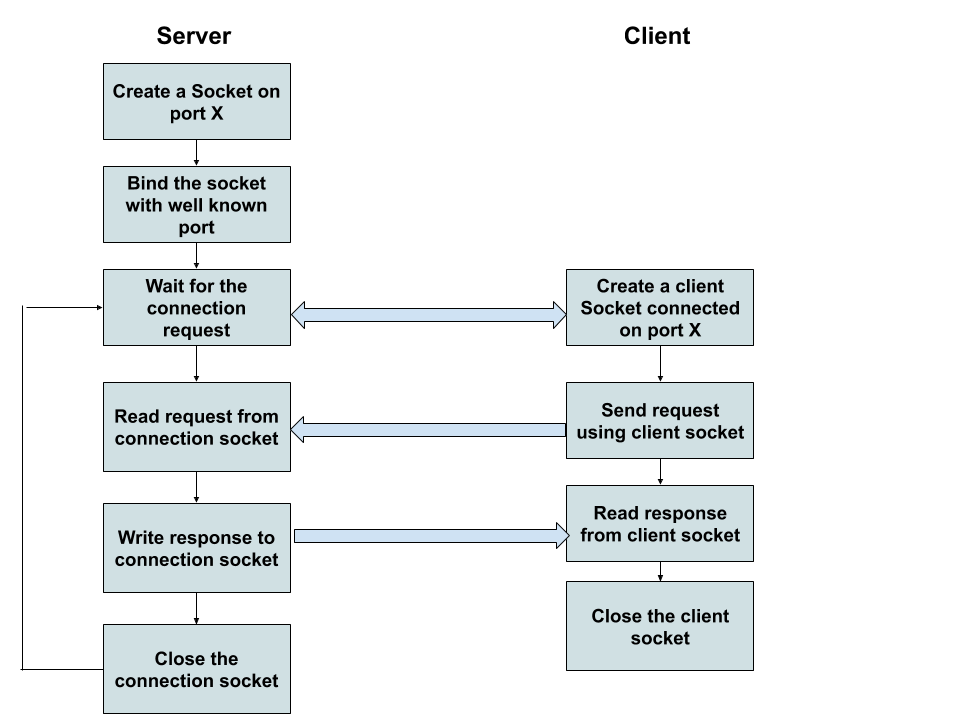
Email: dpatel23@albany.edu

**Table of Contents**

|  |  |
| --- | --- |
| **Title** | **Page number** |
| High-Level Data Flow Diagram | 3 |
| List of Routines and Description | 4 |
| Implementation Details such as Data Structures and Algorithms | 5 |
| Test Documentation | 6 |
| User Documentation | 11 |
| Source Code | 11 |

**System Documentation**

**High-Level Data Diagram**

****

**List of routines and their brief description**

* int socket(int domain, int type, int protocol)
  + Creates an unbounded socket in a communications domain.
  + Return a file descriptor that can be used in later functions that operate on sockets.
  + Domain - Specifies the communications domain in which a socket is to be created.
  + Type - Specifies the type of the socket to be created.
  + Protocol - Specifies a protocol to be used within the socket.
* bool socket\_bind(resource socket, string address, int port)
  + Binds the name given in address to the socket described by the socket, which must be a valid socket resource, created by socket\_create().
  + The address parameter is either an IP address if the socket is of AF\_INET family or the pathname of a Unix-domain socket if it is AF\_UNIX family.
  + The port parameter is only used when connecting using AF\_INET socket.
* bool socket\_listen(resource socket, int backlog)
  + Used for listening to the incoming connections on the socket.
  + A maximum of backlog incoming connections will be queued for processing.
  + If a connection request arrives with the queue full, the client may receive an error with an indication of ECONNREFUSED.
  + Only applicable to sockets of type SOCK\_STREAM or SOCK\_SEQPACKET.
* bool socket\_connect(resource socket, string address, int port)
  + Initiates a connection using a resource socket, which must be a valid resource socket created by socket\_create().
  + All input parameters are the same as socket\_bind.
* string socket\_read(resource socket, int length, int type)
  + Reads from the resource socket created by socket\_create() or socket\_accept() functions.
  + The maximum number of bytes read is specified by the length parameter.
  + Returns data as a string on success, or false on error.
* int socket\_write(resource socket, string buffer, int length)
  + Writes to the resource socket from the buffer.

**Implementation details such as Data Structures and Algorithms**

In our source code files, we have used the shared socket concept for the communication between client and server. Also, we used the structure cmds for storing the processes and executing them with approximate process ID.

This project also implements pipes that act as a queue data structure in a first in first out (FIFO) manner.

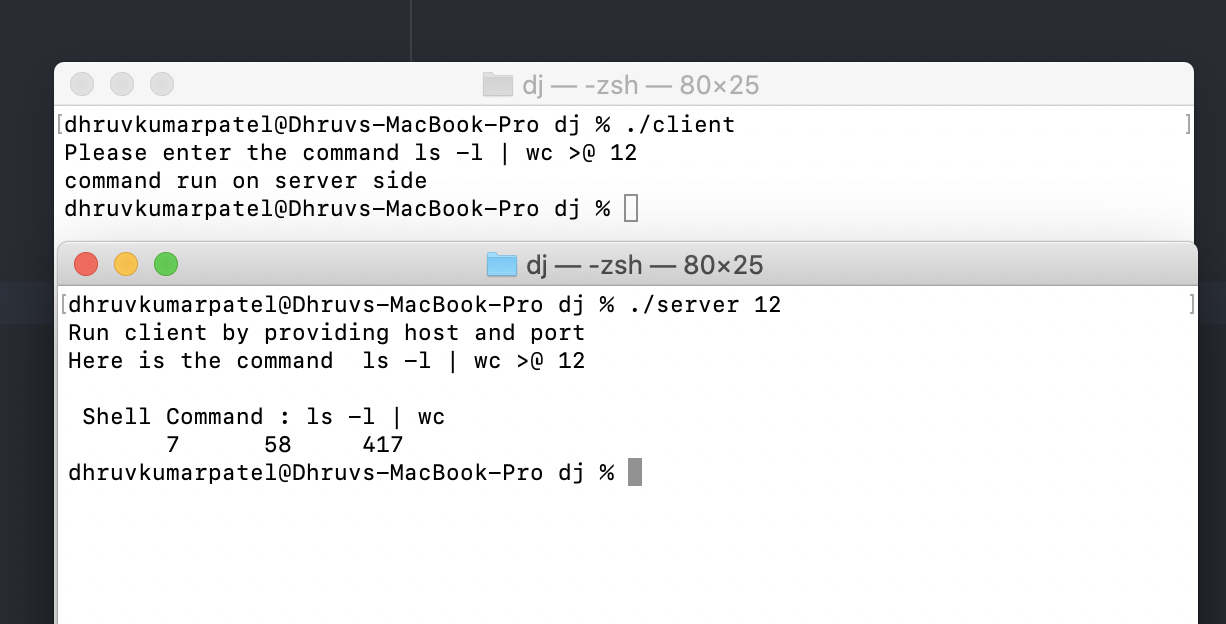
**Test Documentation**

We tested the program using the terminal. Firstly, we started the server on port 4000 and then executing the client at the same port. Secondly, we executed many shell commands. If >@ is included in the buffer, then the command gets executed in the server terminal and also the output gets displayed in it. If <@ is included in the buffer, then the command execution takes place at the server side then it sends the output of the command as an input to the client and it gets displayed in the client’s terminal.

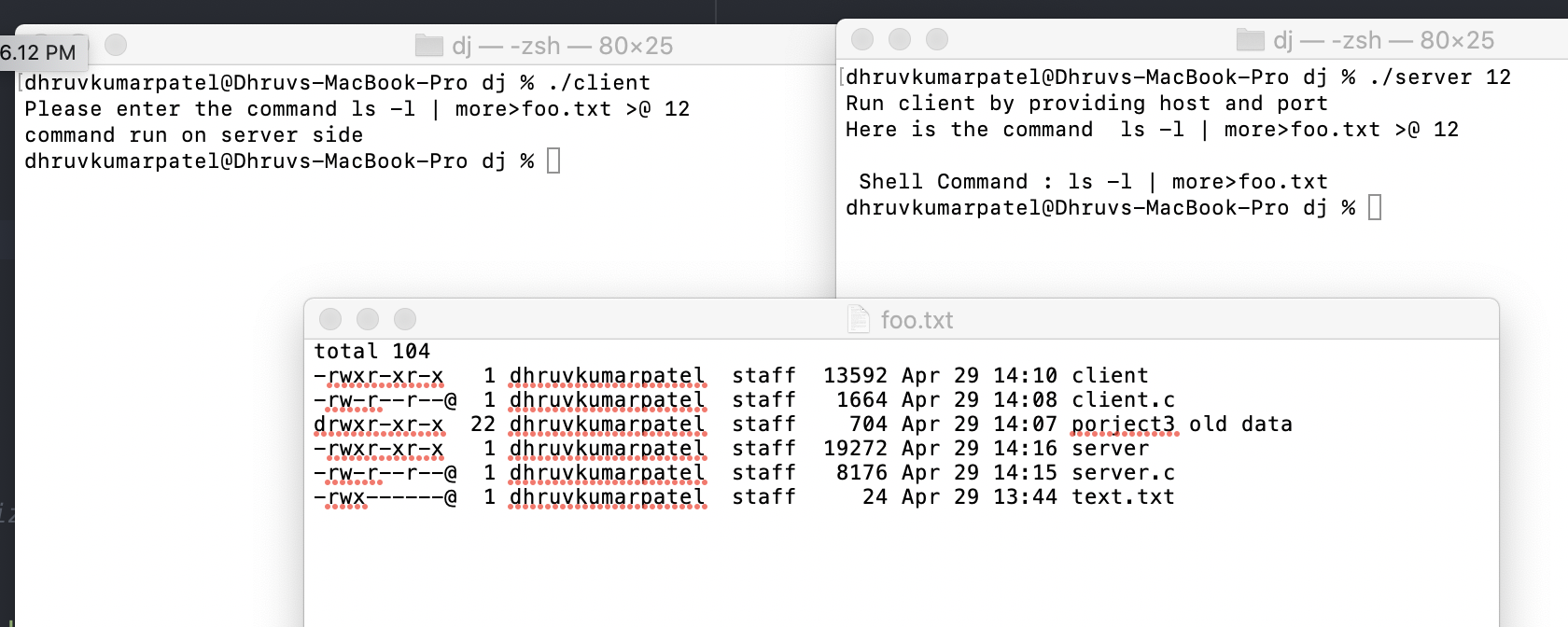
**Testing Outputs**

* Screenshots of a server acting as an output

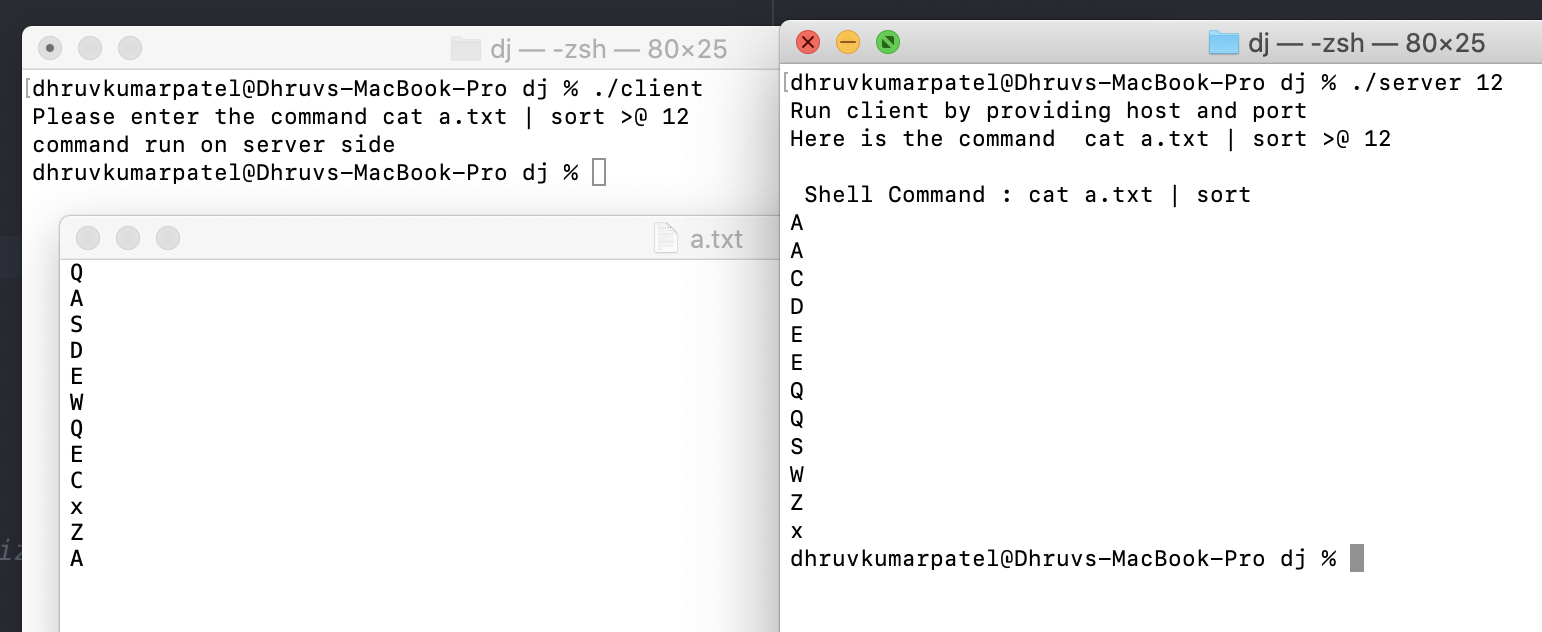
1. ls -l | wc >@ 12



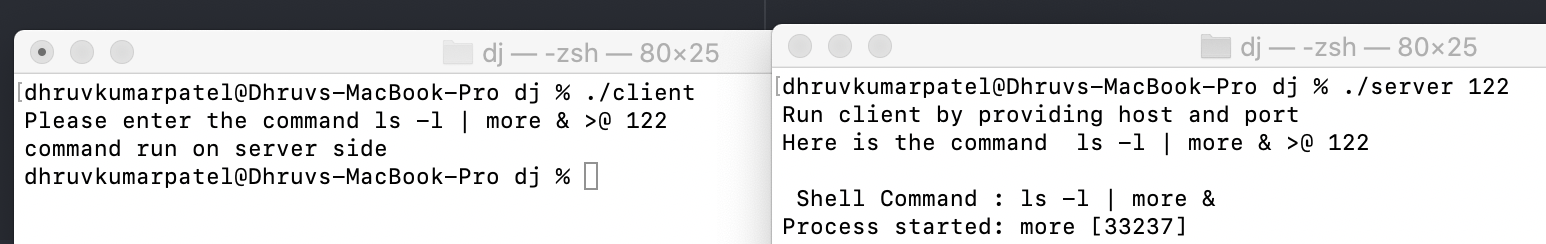
1. ls -l | more > foo.txt >@ 12



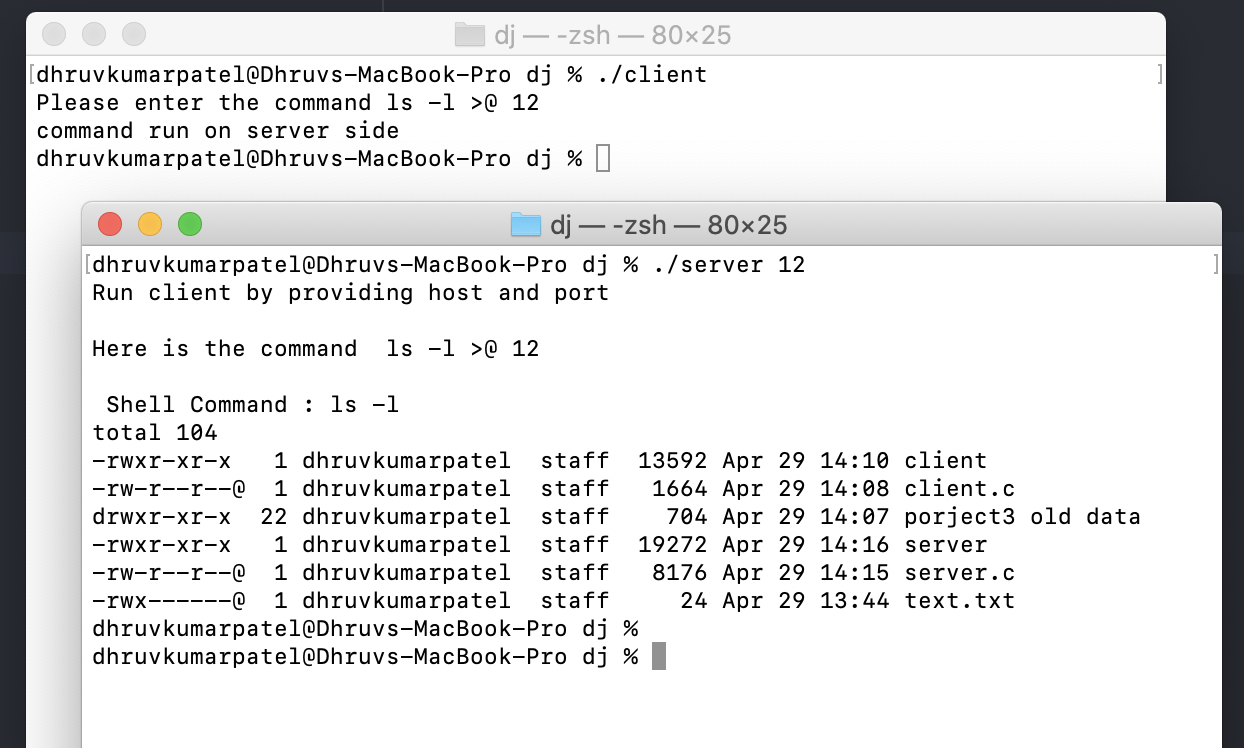
1. cat a.txt | sort >@ 12



1. ls -l | more & >@ 122

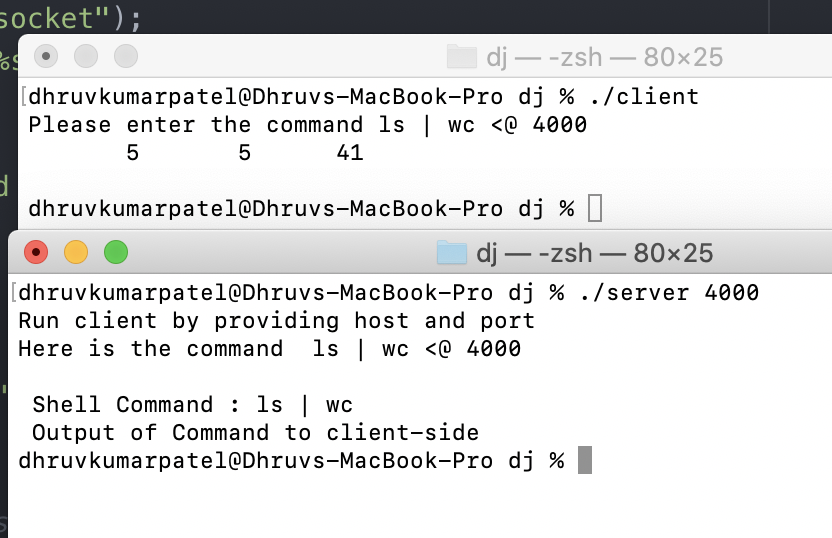


5) ls -l >@ 12

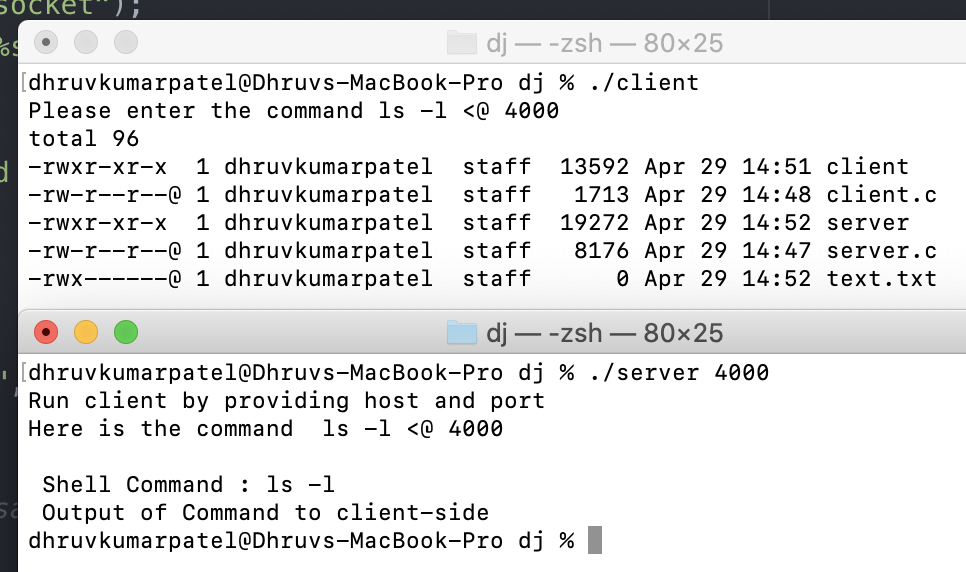


* Screenshots of server executing the command and providing the input to the client

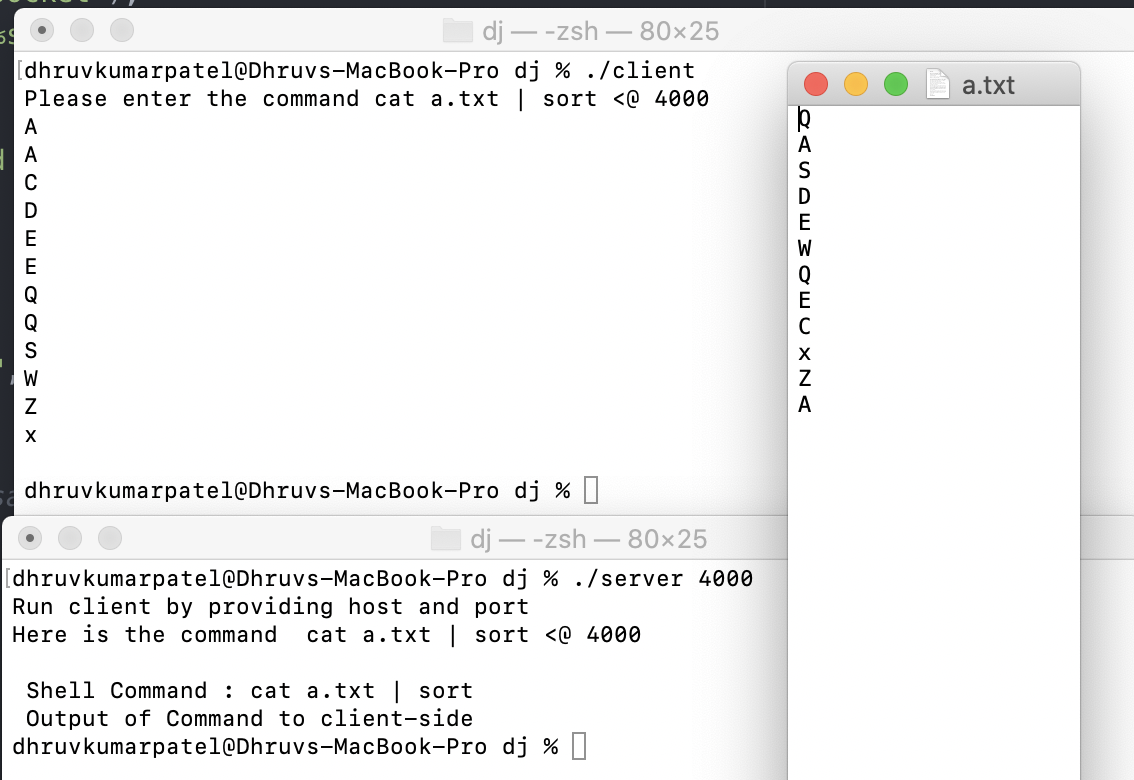
1. ls | wc <@ 4000



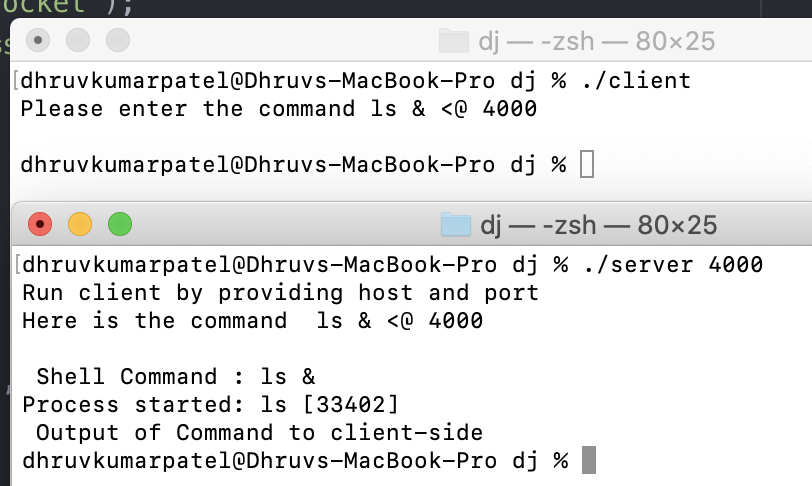
1. ls -l <@ 12



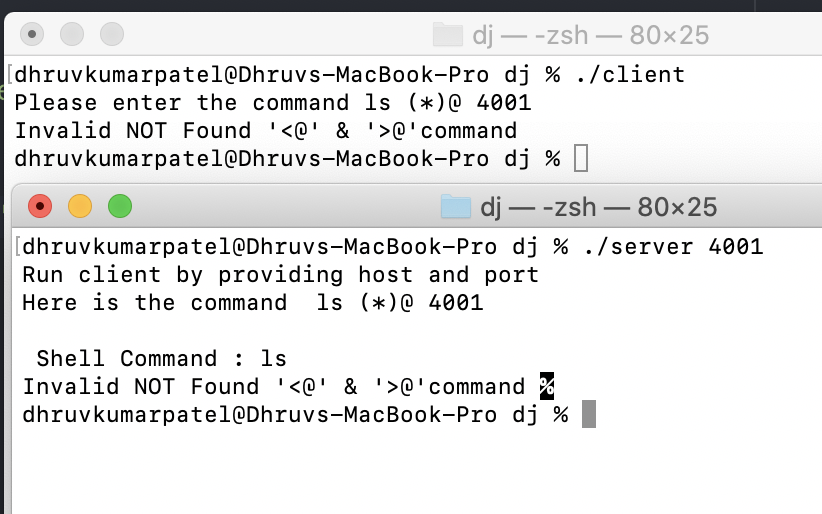
1. cat a.txt | sort | sort <@ 4000



1. ls & <@ 4000



1. ls (\*) @ 4001



**User Documentation**

**How to run the program:-**

1. Open the terminal
2. Compile both the server and client files using gcc which will create executable file shell.
3. Run ./server 4000 on the server-side and on the client-side ./client.
4. Then on the client side enter the command with <@ or <@.
5. Test the commands using the following way:-
   1. Please enter the command: ls >@ 4000

**Source code**

**client.c**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <netdb.h>

#define MAX 1000

void error(const char \*msg)

{

perror(msg);

exit(0);

}

int main(int argc, char \*argv[])

{

int sockfd, portno, n;

struct sockaddr\_in serv\_addr;

struct hostent \*server;

char buffer[256];

char b[256],in[MAX];

char str[50];

char\* token;

int k,i = 0;

char \*p[50];

printf("Please enter the command ");

bzero(buffer,256);

bzero(b,256);

bzero(in,MAX);

fgets(buffer,256,stdin);

strcpy(b,buffer);

token = strtok(buffer," ");

while (token != NULL) {

p[i++] = token;

token = strtok(NULL, "\n,' ' ");

}

//printf("%s : Buffer\n",x);

portno = atoi(p[i-1]);

sockfd = socket(AF\_INET, SOCK\_STREAM, 0);

if (sockfd < 0)

error("ERROR opening socket");

server = gethostbyname("127.0.0.1");

if (server == NULL) {

fprintf(stderr,"ERROR, no such host\n");

exit(0);

}

bzero((char \*) &serv\_addr, sizeof(serv\_addr));

serv\_addr.sin\_family = AF\_INET;

bcopy((char \*)server->h\_addr,

(char \*)&serv\_addr.sin\_addr.s\_addr,

server->h\_length);

serv\_addr.sin\_port = htons(portno);

if (connect(sockfd,(struct sockaddr \*)&serv\_addr,sizeof(serv\_addr)) < 0)

error("ERROR connecting");

n = write(sockfd,b,strlen(b));

if (n < 0)

error("ERROR writing to socket");

bzero(buffer,256);

n = read(sockfd,in,MAX);

if (n < 0)

error("ERROR reading from socket");

printf("%s\n",in);

close(sockfd);

return 0;

}

**server.c**

#include <arpa/inet.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <ctype.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <signal.h>

#define MAX 1000

//Variable for custom shell

char host[MAX];

char user[MAX];

char line[MAX];

char execpath[MAX];

char execdir[MAX];

int F\_IN=0;

int F\_OUT=1;

int CMDCTR=1;

int PIPEIN=0;

pid\_t SHELLPID;

//functions prototype for custom shell

void tokenize( char \*line , char \*\*tokens , char\* delimiter);

void deletecmd(int cmdpid);

void checkINoperator(char \*tokens);

void checkOUToperator(char \*tokens);

int check\_bg(char \*\* tokens);

void commandExe();

int check\_space();

typedef struct cmds

{

char name[100];

pid\_t pid;

}cmds;

cmds cmd[MAX];

void error(const char \*msg)

{

perror(msg);

exit(1);

}

int getdata(char\* b) {

char \*p[50];

char\* token;

int i,j = 0;

int k;

char command[256];

bzero(command,256);

token = strtok(b," "); // tokenizing string

while (token != NULL) {

p[i++] = token;

token = strtok(NULL, "\n,' ' ");

}

for( j=0; j<i-2; j++){ // seperating command form line

strcat(command,p[j]);

strcat(command," ");

}

if(strcmp(p[i-2],">@")==0){ // for command run on server side and output displayed on server side

k=1;

}

else if(strcmp(p[i-2],"<@")==0){ // for command run on server side and output displayed on client side

k=0;

}

else{ // for invalid command

k=2;

}

printf(" Shell Command : %s \n",command);

if(k==1){ // for command run on server side and output displayed on server side

SHELLPID = getpid();

getcwd(execdir,MAX);

line[0] = '\0';

strcpy(line,command);

commandExe(); //executing commands

}

else if(k==0){ // for command run on server side and output displayed on client side

strcat(command," > text.txt");

SHELLPID = getpid();

getcwd(execdir,MAX);

line[0] = '\0';

strcpy(line,command);

commandExe();

printf(" Output of Command to client-side \n");

}

else{ // for invalid command

printf("Invalid NOT Found '<@' & '>@'command ");

}

return k;

}

int main(int argc, char \*argv[])

{

int sockfd, newsockfd, portno;

socklen\_t clilen;

char buffer[256];

char b[256];

struct sockaddr\_in serv\_addr, cli\_addr;

int n;

if (argc < 2) {

fprintf(stderr,"ERROR, no port provided\n");

exit(1);

}

fprintf(stdout, "Run client by providing host and port\n");

sockfd = socket(AF\_INET, SOCK\_STREAM, 0);

if (sockfd < 0)

error("ERROR opening socket");

bzero((char \*) &serv\_addr, sizeof(serv\_addr));

portno = atoi(argv[1]);

serv\_addr.sin\_family = AF\_INET;

serv\_addr.sin\_addr.s\_addr = INADDR\_ANY;

serv\_addr.sin\_port = htons(portno);

if (bind(sockfd, (struct sockaddr \*) &serv\_addr,

sizeof(serv\_addr)) < 0)

error("ERROR on binding");

listen(sockfd,5);

clilen = sizeof(cli\_addr);

newsockfd = accept(sockfd,(struct sockaddr \*) &cli\_addr,&clilen);

if (newsockfd < 0)

error("ERROR on accept");

bzero(b,256);

n = read(newsockfd,b,255);

if (n < 0)

error("ERROR reading from socket");

printf("Here is the command %s\n",b);

int k = getdata(b);

if(k==1){

n = write(newsockfd,"command run on server side",256);

}

else if(k==0) {

char buff[MAX];

bzero(buff,MAX );

FILE \*f = fopen("./text.txt", "rb");

fseek(f, 0, SEEK\_END);

long fsize = ftell(f);

fseek(f, 0, SEEK\_SET); /\* same as rewind(f); \*/

char \*string = malloc(fsize + 1);

fread(string, 1, fsize, f);

fclose(f);

string[fsize] = 0;

strcat(buff, string);

n = write(newsockfd,buff,MAX);

}

else{

n = write(newsockfd,"Invalid NOT Found '<@' & '>@'command ",256);

}

if (n < 0)

error("ERROR writing to socket");

close(newsockfd);

close(sockfd);

return 0;

}

//Functions for custom shell

// Used to tokenize lines into commands

void tokenize( char \*line , char \*\*tokens , char\* delimiter)

{

char \*temp;

int i = 0;

//Splitting line by delimiter and saving it into temp

temp = strtok( line , delimiter );

while( temp != NULL )

{

tokens[i++] = temp;

temp = strtok( NULL , delimiter );

}

}

void deletecmd(int cmdpid)

{

int i;

int flag=0;

for(i=1;i<CMDCTR;i++)

{

if( cmd[i].pid == cmdpid )

flag=1;

if(flag==1)

cmd[i]=cmd[i+1];

}

}

//Function to call shell

void checkINoperator(char \*tokens)

{

int i=0;

char \*temp[100]={NULL} ;

tokenize( tokens , temp , "<" );

if(temp[1]!=NULL)

{

char \*temp1[100]={NULL};

tokenize( temp[1] , temp1 , " " );

F\_IN = open(temp1[0],O\_RDONLY);

}

}

void checkOUToperator(char \*tokens)

{

int i=0;

char \*temp[100]={NULL} ;

tokenize( tokens , temp , ">" );

if(temp[1]!=NULL)

{

char \*temp1[100]={NULL};

tokenize( temp[1] , temp1 , " " );

F\_OUT = open(temp1[0],O\_TRUNC | O\_WRONLY | O\_CREAT, S\_IRWXU);

}

else

F\_OUT=1;

}

int check\_bg(char \*\* tokens)

{

int i=0;

while(tokens[i]!=NULL)

{

if(strcmp(tokens[i],"&")==0)

{

tokens[i]=NULL;

return 1;

}

i++;

}

return 0;

}

//Executes all the commands in a single line

void commandExe()

{

char \*temp[100]={NULL};

char \*temppipe[100]={NULL};

// Checking if there are multiple commands seperated by ;

tokenize( line , temppipe , ";" );

int j=0;

while(temppipe[j]!=NULL)

{

tokenize( temppipe[j] , temp , "|" );

int i = 0;

int fd[2];

while( temp[i] != NULL )

{

char \*tokens[100] ={NULL};

char \*temp1[100]={NULL};

char \*temp2[100]={NULL};

checkOUToperator(temp[i]);

checkINoperator(temp[i]);

// Seperating commands and various arguments

tokenize( temp[i] , temp1 , "<" );

tokenize( temp1[0] , temp2 , ">" );

tokenize( temp2[0] , tokens , " " );

if(tokens[0]==NULL)

return;

int bg = check\_bg(tokens);

if ( strcmp(tokens[0],"exit") == 0)

\_exit(0);

else

{

pipe(fd);

if(bg)

strcpy(cmd[CMDCTR].name,tokens[0]);

pid\_t pid;

int flag;

pid = fork();

if( pid < 0 )

{

perror("Fork\_error ");

}

else if(pid == 0 )

{

if(F\_IN != 0)

{

dup2(F\_IN,STDIN\_FILENO);

close(F\_IN);

}

if(F\_OUT != 1)

{

dup2(F\_OUT,STDOUT\_FILENO);

close(F\_OUT);

}

if(temp[i+1]!=NULL)

{

dup2(fd[1],STDOUT\_FILENO);

close(fd[1]);

}

// executing command

if(execvp(\*tokens,tokens) < 0)

{

perror("Error ");

exit(0);

}

}

else

{

//waiting for child process to end

if(!bg)

wait(&flag);

else

{

cmd[CMDCTR++].pid = pid;

printf("Process started: %s [%d]\n",tokens[0],pid);

}

F\_IN=fd[0];

close(fd[1]);

}

}

i++;

}

F\_IN=0;

F\_OUT=1;

j++;

}

}

int check\_space()

{

int i=0;

while( line[i] != '\0' )

{

if(line[i]!=' ' && line[i]!='\t')

return 0;

i++;

}

return 1;

}