Cyrill Castro

CSC 138

Prof J Dai

2 May 2019

Project 4 Resubmission – C Implementation of Client-Server Communications

Source code of server.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <strings.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <netdb.h>

#include <unistd.h>

#define SERVER\_PORT 5432

#define MAX\_PENDING 5

#define MAX\_LINE 256

int main() {

struct sockaddr\_in sin;

char buf[MAX\_LINE];

int s, new\_s, len;

/\* build address data structure \*/

bzero((char \*)&sin, sizeof(sin));

sin.sin\_family = AF\_INET;

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

sin.sin\_port = htons(SERVER\_PORT);

/\* setup passive open \*/

s = socket (PF\_INET, SOCK\_STREAM, 0);

if (s < 0) {

perror("ERROR: Failed to create socket.");

exit(EXIT\_FAILURE);

} else {

/\* Bind the socket to data structure \*/

new\_s = bind(s,(struct sockaddr \*)&sin, sizeof(sin));

if (new\_s < 0) {

perror("ERROR: Failed to bind socket.");

exit(EXIT\_FAILURE);

} else {

/\* Make the socket to listen \*/

listen(s, MAX\_PENDING);

//printf("Server is listening to port: %d," SERVER\_PORT);

int new = accept(s, (struct sockaddr \*)&sin, &len);

/\* wait for connection, then recieve and print text \*/

if (new < 0) {

perror("ERROR: Failed to accept connection.");

exit(EXIT\_FAILURE);

} else {

while (1) {

len = sizeof(sin);

/\* Accept connections \*/

bzero(buf, MAX\_LINE);

/\* Receive and print text \*/

recv(new, buf, len, 0);

printf("%s", buf);;

}

close(new);

return EXIT\_SUCCESS;

} // end of accept check

} // end of bind check

} // end of socket check

}

Source code of client.c

#include <stdio.h>

#include <stdlib.h>

#include <strings.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <netdb.h>

#include <unistd.h>

#define SERVER\_PORT 5432

#define MAX\_LINE 256

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

{

struct hostent \*hp;

struct sockaddr\_in sin;

char \*host;

char buf[MAX\_LINE];

int s; /\* socket \*/

int len; /\* addr\_len \*/

host = argv[1];

if ((host == NULL) || (host == '\0')) {

perror("ERROR: Host argument is null.");

exit(EXIT\_FAILURE);

} else {

/\* translate host name into peer's IP address \*/

hp = gethostbyname(host);

if (hp == NULL) {

perror("ERROR: Null pointer.");

exit(EXIT\_FAILURE);

} else {

/\* build address data structure \*/

bzero((char\*)&sin, sizeof(sin));

sin.sin\_family = AF\_INET; /\*Internet Address\*/

bcopy(hp->h\_addr, (char \*)&sin.sin\_addr, hp->h\_length);

sin.sin\_port = htons(SERVER\_PORT);

/\* Create the socket \*/

s = socket(PF\_INET, SOCK\_STREAM, 0);

if (s < 0) {

perror("ERROR: Failed to create socket.");

exit(EXIT\_FAILURE);

} else {

/\* Connect the socket with server\*/

int c = connect(s, (const struct sockaddr \*)&sin, sizeof(sin));

if (c < 0) {

perror("ERROR: Failed to connect socket.");

exit(EXIT\_FAILURE);

} else {

/\* main loop: get and send lines of text \*/

while (1) {

bzero(&buf, MAX\_LINE);

fgets(buf, sizeof(buf), stdin);

buf[MAX\_LINE - 1] = '\0';

len = strlen(buf) + 1;

send(s, buf, len, 0);

}

close(s);

return EXIT\_SUCCESS;

} // end of connect check

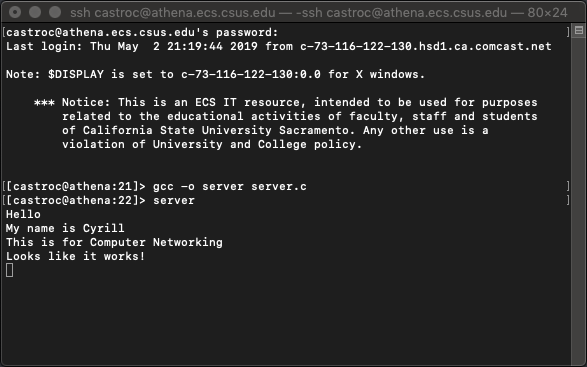
} // end of socket check

} // end of gethostbynamecheck

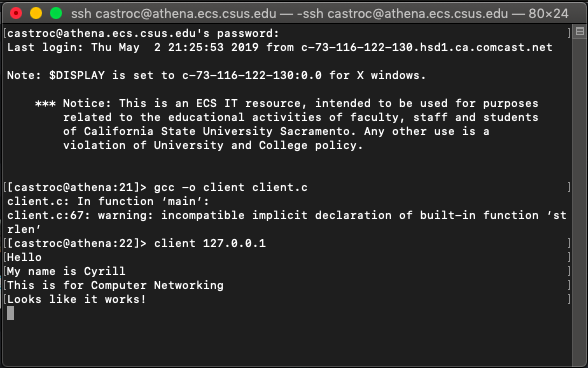
} // end of host check

}

Result (screenshot) of server.c commands and run



Result (screenshot) of client.c commands and run



Analysis

For this resubmission of project 4, I first improved the source code of both server.c and client.c so that when they both are compiled, there are syntax errors. After ensuring that both of the C files compiled smoothly, I corrected any errors in the logic of how the client and server worked to communicate with each other. Once I was able to successfully run both sides of the communication, I implemented the “protection” part of the client and the server, to ensure that they both do not crash if there were any errors made in the functions that were called. The final result for this project was a smooth, protected run on both ends.

The client side is responsible for creating the socket, then connecting the socket with the server. It receives the text and sends it to the server. Meanwhile, the server side is responsible for creating the socket, binding the socket to the address data structure (which is used to collect the text from the client), and making the socket listen. Once the connection is accepted, then the server receives the text from the client side and prints it. As for the robustness, I performed a check on the integers that were returned by each function that was used. If they returned a result that indicated a successful run, then it continued to the next portion and check until the run was successful. If there was an error in any of the checks, the code printed an error message and returned an exit failure. This method of security was implemented in both sides to ensure that no errors made either of them crash and ruin the connection/communication.

Through this project, I was able to fully understand how client and server communicate with each other through the socket, and I was able to learn about the true functions of each side, and how important each of those functions are to create the communication between the two parts of the socket.