
# TCP Project

Data Communication and Network Programming

Shrijanand Chintapatla 10/18/18 Data Comm & Network Prog

# TABLE OF CONTENTS:

# TCP Project – Shrijanand Chintapatla

## 1) GitHub Repository Link:

*Answer:*

The link is given below:

https://github.com/shrijanand/TCP_Programming_Project

README.md

# README - TCP_Programming_Project

This README exists for the TCP Programming Project for CSCI 450: Data Communication and Network Programming, Fall 2018.

### Who are the people responsible for the TCP Programming Project?

1 - Shrijanand Chintapatla

### Project Design and Sprint 1 Plan

- Create Repo and initialize README.md
- ---- First README.md added to TCP_Programming_Project at October 2 at 18:30
- Create Server.c and Client.c Source Code Files
- ---- Added Server.c and Client.c Source Code Files to Repo.

### Sprint 2 Plan and Sprint 1 Refinement

- Included necessary Libraries to Server.c and Client.c
- ---- Added editted Server.c and Client.c Source Code Files to Repo.
- Editted README.md for Libraries and Main Function
- ---- Editted README.md for TCP_Programming_Project.

### Sprint 3 Plan and Sprint 2 Refinement

- Editted Server.c and completed Implementation for V1 of Server.
- ---- Editted README.md for TCP_Programming_Project pertaining to Server.
- Editted Client.c and completed Implementation for V1 of Client.
- ---- Editted README.md for TCP_Programming_Project pertaining to Client.

### Final Sprint - Sprint 4 Plan

- Tested Server.c and Client.c to communicate w/ each other.
- ---- Ran simple tests to have base-level Minimum Viable Product (MVP) Project.
- Editted Server.c and completed Implementation for V2 of Server.
- ---- Editted README.md for TCP_Programming_Project pertaining to Server.
- Editted Client.c and completed Implementation for V2 of Client.
- ---- Editted README.md for TCP_Programming_Project pertaining to Client.

## 2) Application Protocol:

*Answer:*

Type of messages:

- Request: Client sends a request to the Server.
- Response: Server sends a response after the request to the Client.

  Specifically, Server translates to the format type specified in the request of the Client.
- Success Message: Simple text "Success!" that indicated successful operation in expected behavior.
- Error Message: Numerous error messages starting in structure with stderr to indicate unsuccessful operation in, however, expected behavior.

Syntax and Semantics:

Format:

```
typedef struct Message_Struct
{
    uint8_t Transmission_Type;
    uint16_t Length_of_Name;
    uint64_t Size_of_File;
    unsigned char* File;
    unsigned char* Name_of_File;
}Message_Struct;
```

Meaning:

- Transmission_Type stores the 'to format' value – 0,1,2,3.

- Length_of_Name stores the size of the File Name.
- Size_of_File stores the File specified.
- File stores the Output File.
- Name_of_File stores the size of the Output File Name.

Range:

Transmission_Type: 0, 1, 2, 3

Length_of_Name: 0 to $2^{16}$

Size_of_File: 0 to $2^{64}$

Rules:

- The Server needs to be initialized first before the Client can send requests.
- Client sends the request to the Server.
- Server responds to the request to the Client.
- Server translates among 4 various formats.
- Server and Client must depict errors with stderr to indicate unsuccessful operation in, however, expected behavior.

## 3) Test Cases:

*Answer:*

The following table depicts all the Test Cases that Server.c and Client.c were tested against:

| Test Cases: | Expected Output: | Actual Output: | Errors: |
|---|---|---|---|
| **File does not exist.** | Error: Kindly check that the file exists. | Error: Kindly check that the file exists. | File does not exist. |
| **File is empty.** | Error: Kindly check that the file is not empty. | Error: Kindly check that the file is not empty. | File is empty. |
| **Format is out of range.** | Error: Kindly check that Format is not out of range. | Error: Kindly check that Format is not out of range. | Format is not 0, 1, 2, or 3. |
| **[Abbreviated due to redundancy] File with specific type, to be translated to a specified type.** | Normal working and expected behavior and output. | Normal working and expected behavior and output. | No errors. |

| | | | |
|---|---|---|---|
| **[Abbreviated due to redundancy] File with two specific types, to be translated to a specified type.** | Normal working and expected behavior and output. | Normal working and expected behavior and output. | No errors. |
| **[Abbreviated due to redundancy] One specific type, w/ errors.** | Format Error. | Format Error. | Format Error. |
| **[Abbreviated due to redundancy] Two specific types, w/ errors in one specific type.** | Format Error. | Format Error. | Format Error. |
| **[Abbreviated due to redundancy] N clients send files one-by-one.** | Success! | Success! | Success! |

## 4) Usage of Client and Server:

*Answer:*

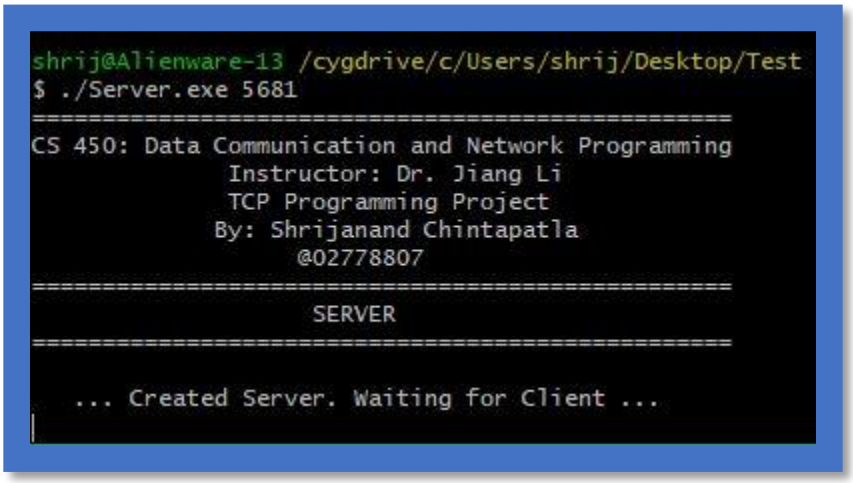
Usage of Server.c

We can instantiate this server by running the following command in your preferred *nix environment/terminal:

*<Server><Port_Number>*

Ex: ./Server.exe 5681

*Figure:*




*Description:*

Server.c essentially creates a server socket from a given Port Number and then starts the pertaining server.

After creating this listening socket, relevant data members are now inputted into the socket address structure. Then, socket address is bind to the socket that continues to listen. The listening socket continues to listen to requests from a hypothetical client.

When the client sends data from a file specified by its file path, the server checks the received data. The server then translates different formats of data to others' by specified format type. Then, the server saves this translation to a specified file name in the directory.

*Basic Error Handling:*

If enough arguments are not passed, or if a port number has not been provided, Server.c would print a stderror and terminate.

```
if (argc < 2)
{
    fprintf(stderr, "Error: Kindly check that a port is provided. \n");
    exit(1);
}
```

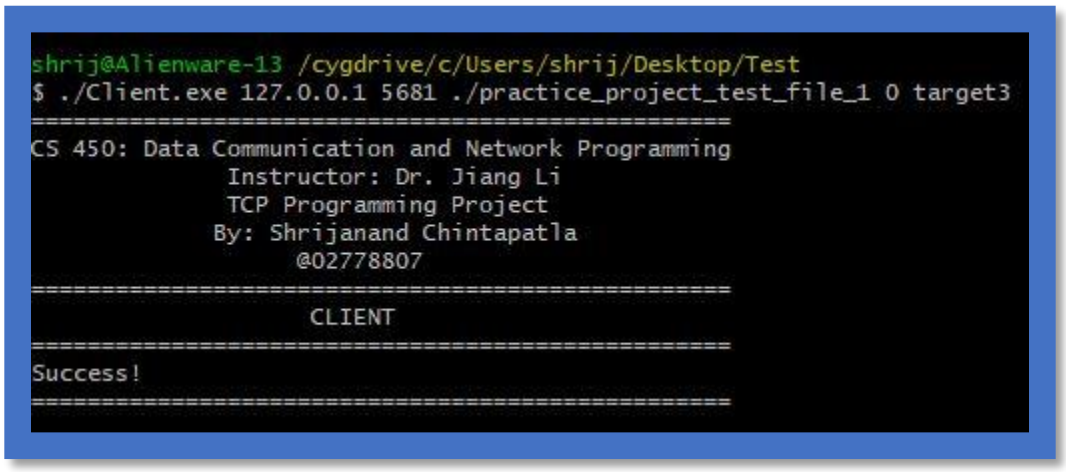
## Usage of Client.c

We can instantiate this Client by running the following command in your preferred *nix environment/terminal:

<Client><Server_IP_Address><Server_Port><File_Path><Format_Type><File_Name>

Ex: ./Client.exe 127.0.0.1 5681 ./practice_project_test_file_1 0 target3

*Figure:*




*Description:*

After creating the socket for the Client, the Client calculates the Message size that requires transmission. The message is then created and written to the socked. Finally, if the response is transmitted in an expected behavior, “Success!” is printed to the screen.

*Basic Error Handling:*

If there aren't enough arguments passed to the Client via the command-line, Client.c would print a stderror and terminate.

```
if (argc < 6)
{
  fprintf(stderr, "Error: Kindly check the number of arguments.\n");
  exit(1);
}
```

If Format is not defined as one of the four translation formats (0, 1, 2, 3), Client.c would print a stderror and terminate.

```
if (Format < 0 || Format > 3)
{
  fprintf(stderr, "Error: Kindly check that Format is not out of range.\n");
  exit(1);
}
```

If the declared File Path is invalid, Client.c would print a stderror and terminate.

```
if (!Valid_File_Path(File_Path))
{
    fprintf(stderr, "Error: Kindly check that the file exists.\n");
    exit(1);
}
```

If File specified is empty, Client.c would print a stderror and terminate.

```
if (File_Empty(File_Path))
{
    fprintf(stderr, "Error: Kindly check that the file is not empty.\n");
    exit(1);
}
```

## 5) Compilation Instructions – Server.c and Client.c:

*Answer:*

Compilation Instructions - Server.c

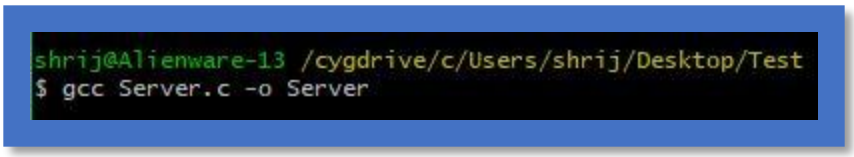
We can compile Server.c by running the following command in your preferred

*nix environment/terminal:

gcc <File_Name.c> -o <File_Name>

Ex: gcc Server.c -o Server

*Figure:*




Compilation Instructions - Client.c

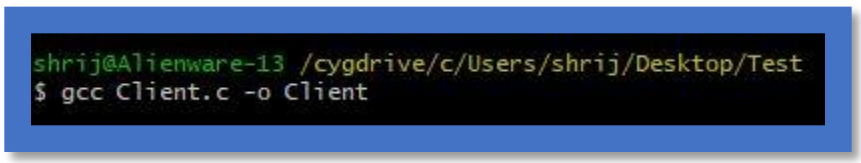
We can compile Client.c by running the following command in your preferred

*nix environment/terminal:

gcc <File_Name.c> -o <File_Name>

Ex: gcc Client.c -o Client

*Figure:*



## 6) Significant References:

*Answer:*

The following Internet resources were instrumentally useful and informational (in no particular order):

- https://www.binarytides.com/socket-programming-c-linux-tutorial/
- http://www.paulgriffiths.net/program/c/sockets.php
- https://www.geeksforgeeks.org/socket-programming-cc/

The following persons helped, in some limited form, with the formulation of the project (in alphabetic order):

- Aarya BC
- Brandon Cole
- David Hill
- Elijah Carter
- Keenah Mays

## 7) Any Known Problems:

*Answer:*

No issues with the Server.c and Client.c noticed.

However, one plausible 'issue' is the lack of closing ports even with interrupting Server.c. For more description, if the process is terminated without closing the instance of Cygwin used to run the Server, the port number might not be able to be used again!

**O ========== O ========== O**