TCP/IP Server
COE768

Meher Bhagat

November 04, 2019

Instructor: Bobby Ma

Section: 06

Contents

List of Figures

List of Tables

1 Introduction

1.1 TCP Protocol

One of the primary protocols used on the Internet protocol suite is the Transmission Control Protocol (TCP). The main advantages of this protocol is that the data transmitted between two nodes is the following:

- Ordered and reliable
- Provides error detection
- Provides connection establishment and connection detection (Three-Way Handshaking)

TCP is one of the most widely used protocols and designed in the following way. All devices in the network are assigned a unique Internet Protocol (IP) address. Each devices can connect to up to $(2^{16} - 1)$ or 65536 ports. Each port can be used to send and receive data from another electronic device. TCP ensures a private connection between two devices.

In TCP, there is always a client and a server. The server is always listening for connections and the client is always attempting to connect to a server. Connection establishment is performed by TCP by using a three-way handshake. The three messages and the purposes of these messages are the following:

1. SYN (Synchronize)
    - The purpose of the SYN message is to initiate a connection from the client to the server.
2. SYN-ACK (Synchronize-Acknowledgement)
    - The purpose of the SYN-ACK message is for the server to respond to the client and acknowledge the SYN message was received.
3. ACK (Acknowledgement)
    - The purpose of the ACK message is for the client to respond to the server and acknowledge the SYN-ACK message was received by the client.

Once all three messages are transmitted successfully, both the client and server can successfully communicate over the channel.

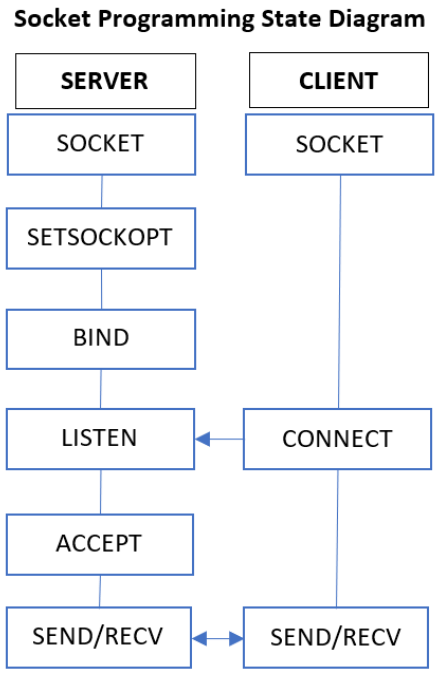


Figure 1: Socket Programming State Diagram

TCP can be implemented in C using the Linux environment, To successfully implement any TCP application, the use of socket programming is required.

1.2 Socket Programming

Socket Programming is a method by which two devices can communicate with one another. The Server socket listens on a port with an assigned IP address. The client reaches out to the server with it's own IP address on the server broadcasting port. This concept is illustrated in Figure 1.

1.3 Purpose

The purpose of this project is to implement a command-line file transfer application based on TCP/IP. The server is a concurrent server which is able to handle multiple clients at the same time. The client has to ability to upload and download files, and change the directory and list the files in the directory from the server.

The client and the server use the following Protocol Data Unit (PDU):

Figure 2: Protocol Data Unit Format

The type specifies the PDU Type as shown in Table 1. The length field is used to state the length of the Data field. The data is contained within the data field.

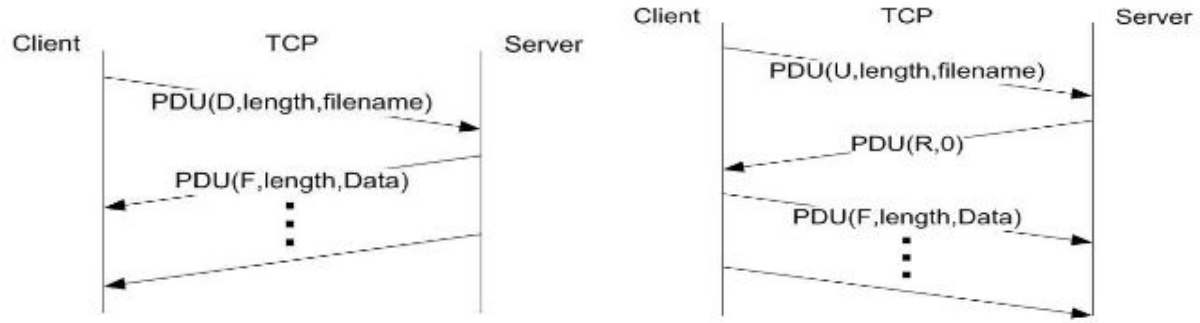
The eight PDUs listed in the table below are used during the file transfer:

PDU Type	**Function**
D	This PDU is sent by the client. The purpose of this PDU is to carry the filename of the file to be transferred from the server to the client.
U	This PDU is sent by the client. The purpose of this PDU is to carry the filename of the file to be transferred from the client to the server.
R	This PDU is sent by the server. The purpose of this PDU is to inform the client that the server is ready to receive messages including file uploads.
F	This PDU is sent by the client and server. The purpose of this PDU is to carry the file data that is being transferred between the client and the server.
E	This PDU is sent by the client and server. The purpose of this PDU is to report any error between the client and the server.
P	This PDU is sent by the client. The purpose of the PDU is to inform the server to change the working directory in which the client can download files
L	This PDU is sent by the client to request the names of the files in the directory. The Data field contains the directory path name
I	This PDU is sent by the server. The Data field contains the names of the file in the directory.

Table 1: PDU Types and their Respective Functions

1.3.1 PDU Exchange Diagrams

The file upload and download PDUs are shown in Figure 3. In the case of file download, the client sends the file name to be downloaded to the server along with a PDU type of D. If the file exists, the server responds with the F PDU type and the file data until the data transfer is complete. In the case of file upload, the client sends the file name to be uploaded to the server along with a PDU type of U. If the server replies with the R PDU type, then the client with the file data (with a F PDU type).



(a) File Download PDU

(b) File Upload PDU

Figure 3: File Download and Upload PDUs

The error PDU is used to report any errors that occur when the file download or file upload is requested by he client.

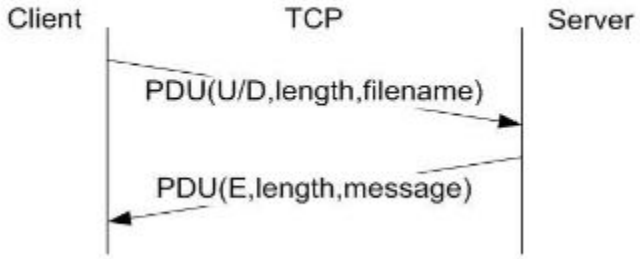
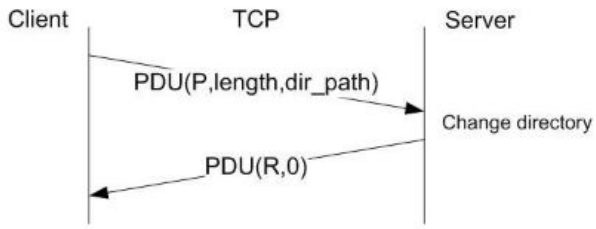
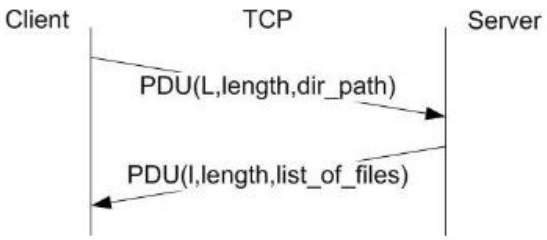


Figure 4: Error Reporting of File Download/Upload PDUs

The P PDU is used to change the current working directory. If successful, then the server sends the client a R PDU. If not, then the E PDU is returned as shown in Figure 4. The L PDU is used is the client requests the list of current working directories. The server responds with I and a list of files or it returns an E PDU if the files cannot be listed.



(a) Change Directory PDU



(b) List Directory PDU

Figure 5: Change and List Directory PDUs

2 Description of Client and Server

The client is able to perform the following functions (the command for the client is provided in brackets):

- File Download ("download")
- File Upload ("upload")
- Change Directory ("cd")
- List Directory ("ls")

All commands except "ls" requires at least 1 command line argument ("ls" takes no arguments). The application detects if an incorrect amount of parameters are entered and will not perform any function if this occurs.

File Download: If the file is not available in the directory that the client wants to download, then the server will respond by saying it is ready and that no file data will be downloaded as shown in Figure 3.

File Upload: The client will send the filename that is entered by the user to server. Once the server is ready it will respond as described in Figure 3. If the file is not available in the working directory of the client, then no file data will be uploaded.

Change Directory: The client will send the new working directory for the server. If the directory is valid, the server will change the directory as shown in Figure 5. If the directory is invalid, the server will not change the directory. The ls command can be run once the directory is changed to validate if the directory has been change successfully.

List Directory: The client will send the server this command and the server will return all files and folders in the currently specified directory as shown in Figure 5.

3 Observations, Analysis and Results

The following sequence of commands was entered on the TCP file transfer client:

1. cd /home/mb
2. ls
3. download program.txt
4. upload program2.txt
5. ls

These sequence of events show the application can perform all intended functions described in the project requirements.

The outputs of these events can be seen in the following screenshots (Note: the client window is on the left and the server window is on the right):

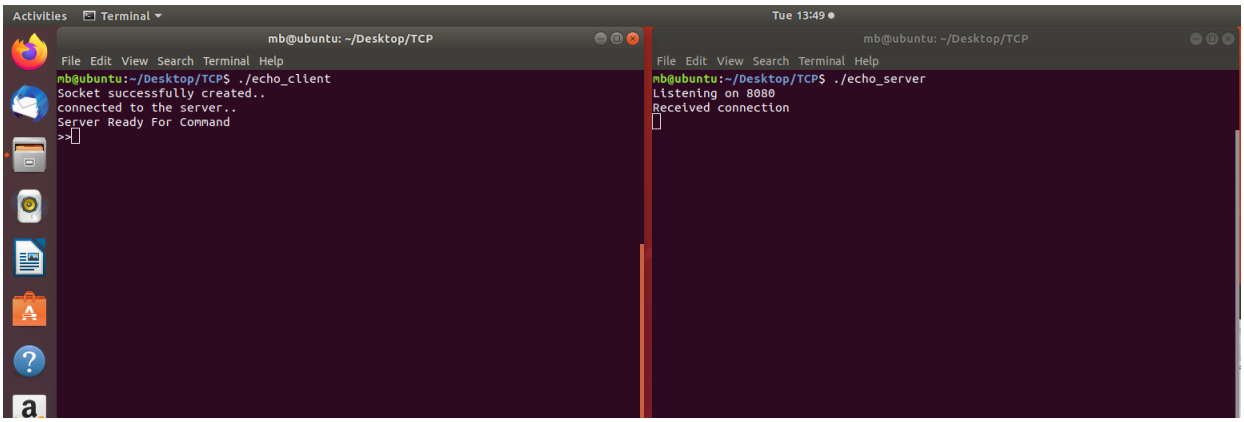


Figure 6: Initial Startup of Client (Left) and Server(Right)

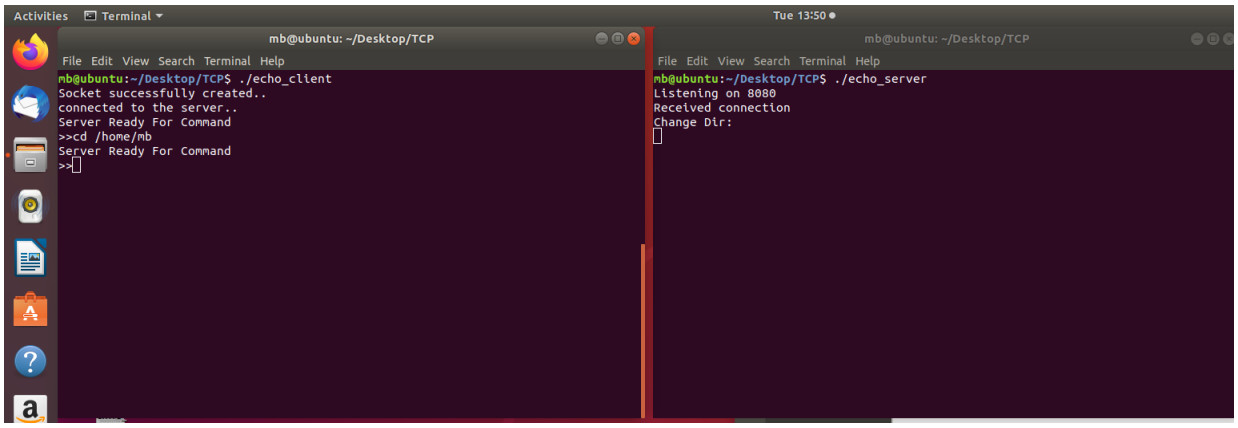


Figure 7: Command 1: Change Directory

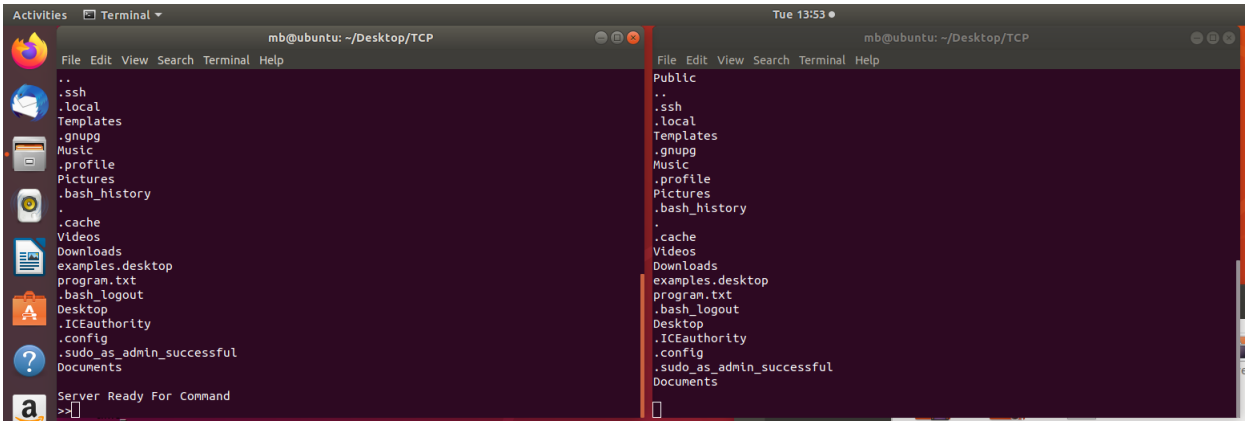


Figure 8: Command 2: List

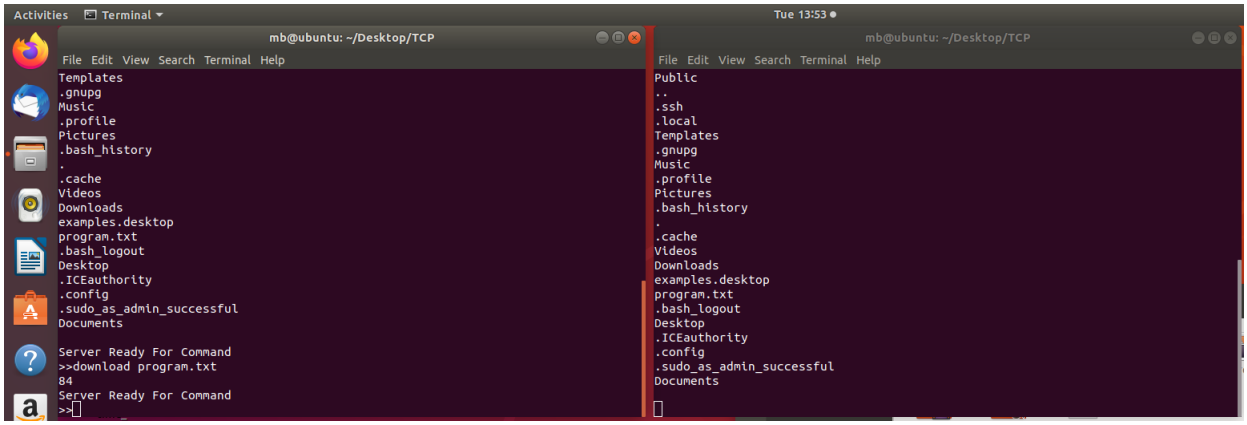


Figure 9: Command 3: Download

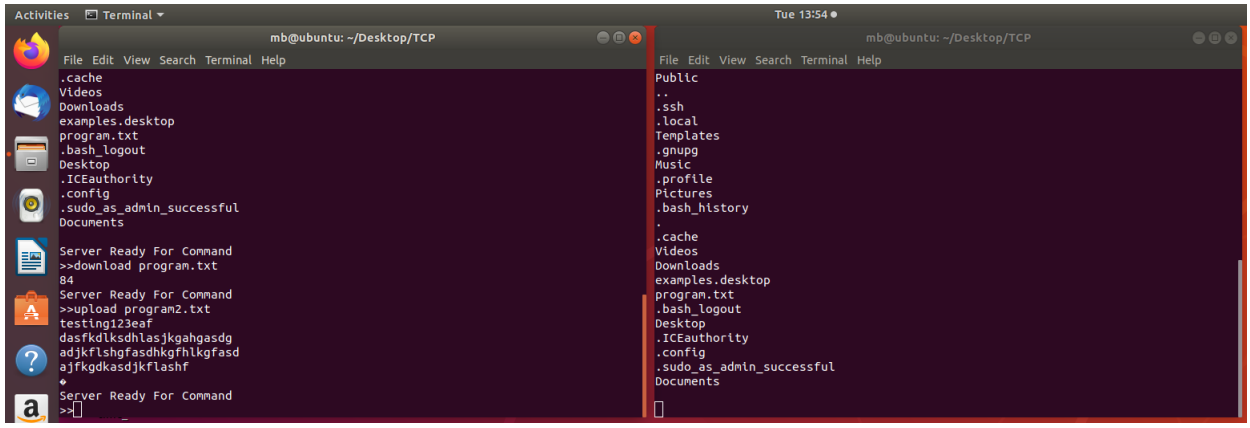


Figure 10: Command 4: Upload

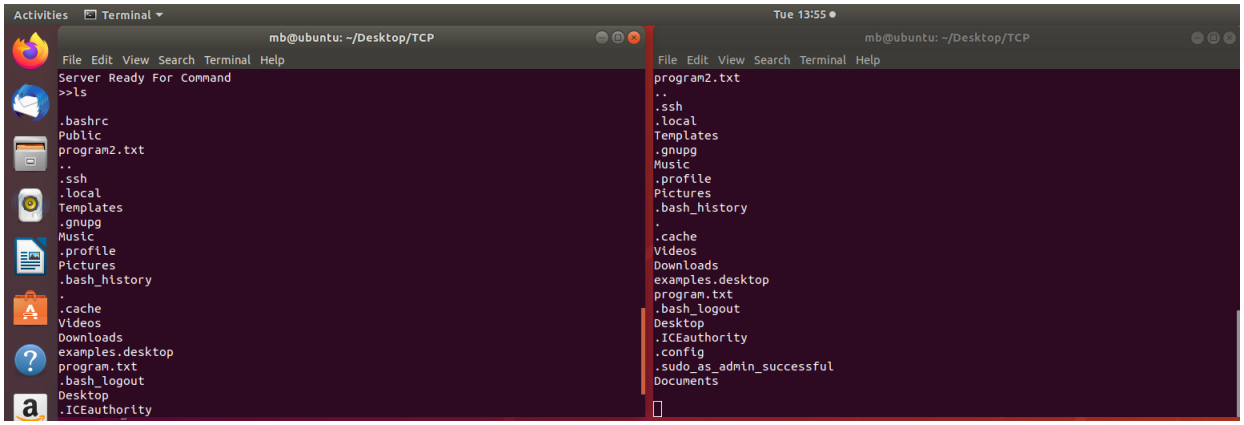


Figure 11: Command 6: List

A Client Code

```c
#include <dirent.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <stdlib.h>
#include <netinet/in.h>
#include <string.h>
#include <sys/stat.h>


#define USER_INPUT_SIZE 4101
#define ARG_SIZE 4097
#define SERVER_TCP_PORT 8080

struct pdu
{
 char type;
 int length;
 char data[ARG_SIZE];
} rpdu, tpdu;


void init_serv(int serv_sock) {
  char user_input[USER_INPUT_SIZE];
  char user_argument[ARG_SIZE];
  char command[3];
  int  i = 0, size = 0;
  char* token;
  FILE* fp;
  struct stat filestat;


  while(rpdu.type != 'R')
    read(serv_sock, &rpdu,sizeof(rpdu));
  printf("Server Ready, %c, recieved\n", rpdu.type);


  while(1){
    printf("\nPlease enter the command (cd 'dir_name', ls 'dir_name', U 'file_name', D '
    file_name'). Enter Q to quit: \n>>");
    fgets(user_input, sizeof(user_input), stdin);

    if(strcmp("ls\n",user_input) == 0){
      strcat(user_input, "current_dir");
    }

    token = strtok(user_input, " \n");

    if(strcmp("Q",token) == 0){
      printf("Exiting Application\n");
      close(serv_sock);
      break;
    }

    if(token == NULL){
      printf("Ensure command has 1 argument\n\n");
    }else{
      //split the command and the user argument:
      while ((token != NULL)) {
        if(i==0){
          strcpy(command,token);
          printf("%s\n",command);
        }else if (i==1){
          strcpy(user_argument,token);
```

```
65            printf("%s\n",user_argument);
66          }
67          i++;
68                token= strtok(NULL, "\n");
69        }
70        i=0;
71
72
73        //prepare message
74        tpdu.length = strlen(user_argument);
75        memcpy(tpdu.data,user_argument,tpdu.length);
76
77
78
79
80
81        if(strcmp("cd",command)==0){
82
83          //prepare type and send
84          tpdu.type='P';
85            write(serv_sock, &tpdu, sizeof(tpdu));//1B for type, 4B for length ,5B overhead
86          //read server response
87          read(serv_sock, &rpdu, sizeof(rpdu));
88          if(rpdu.type =='E')
89            printf("Directory Change Unsuccessful\nError message recieved: %s\n\n",rpdu.data);
90          else if(rpdu.type =='R')
91            printf("\nDirectory Changed Successfully\n\n");
92        }else if(strcmp("ls",command)==0){
93
94          //prepare type and send
95          tpdu.type='L';
96            write(serv_sock, &tpdu,sizeof(tpdu));//1B for type, 4B for length ,5B overhead
97          read(serv_sock, &rpdu, sizeof(rpdu));
98
99          if(rpdu.type =='E')
100           printf("Directory Change Unsuccessful\nError message recieved: %s\n\n",rpdu.data);
101         else if(rpdu.type =='I')
102           printf("\nList:\n%s\nListing successful, Server Ready\n\n",rpdu.data);
103
104       }else if(strcmp("U",command)==0){
105
106         fp = fopen(user_argument,"r");
107
108         if(fp < 0)
109           printf("\nFile does not exist\n");
110         else{
111           tpdu.type='U';
112             write(serv_sock,&tpdu,sizeof(tpdu));//1B for type, 4B for length ,5B overhead
113           read(serv_sock, &rpdu, sizeof(rpdu));
114
115           if(rpdu.type='R'){
116             stat(user_argument, &filestat);
117                 size = filestat.st_size;
118             tpdu.type='F';
119             while(size>ARG_SIZE){
120               fread(&tpdu.data, ARG_SIZE,1,fp);
121               tpdu.length=strlen(tpdu.data);
122               write(serv_sock, &tpdu, sizeof(tpdu));
123               size = (size - (ARG_SIZE));
124             }
125             memset(&tpdu, 0, sizeof(tpdu));
126             tpdu.type='F';
127             fread(&tpdu.data, size,1,fp);
128             tpdu.length=strlen(tpdu.data);
129             write(serv_sock, &tpdu,sizeof(tpdu));
130             fclose(fp);
131           }else
```

```
132             printf("\nServer not Ready, upload failed\n");
133         }
134 
135       }else if(strcmp("D",command)==0){
136 
137 
138         tpdu.type='D';
139           write(serv_sock,&tpdu,sizeof(tpdu));//1B for type, 4B for length ,5B overhead
140         read(serv_sock, &rpdu, sizeof(rpdu));
141         if(rpdu.type == 'E')
142           printf("File not recieved!!! \nError message recieved: %s\n",rpdu.data);
143         else if (rpdu.type == 'F') {
144           fp = fopen(user_argument,"w");
145 
146           while (rpdu.type !='R'){
147             fwrite(rpdu.data , 1 , rpdu.length, fp);
148             read(serv_sock, &rpdu, sizeof(rpdu));
149           }
150           fclose(fp);
151         }
152 
153       }else
154         printf("Enter valid command!!\n\n");
155 
156       memset(&tpdu, 0, sizeof(tpdu));
157 
158     }
159   }
160 
161 
162 
163 }
164 
165 int main (int argc, char *argv[]) {
166 
167     int sd;
168     struct sockaddr_in servaddr, cli;
169   int port;
170 
171     switch(argc){
172   case 1:
173     port = SERVER_TCP_PORT;
174     break;
175   case 2:
176     port = atoi(argv[1]);
177     break;
178   default:
179     fprintf(stderr, "Usage: %s [port]\n", argv[0]);
180     exit(1);
181   }
182 
183 
184     // socket create and varification
185     sd = socket(AF_INET, SOCK_STREAM, 0);
186     if (sd == -1) {
187         printf("socket creation failed...\n");
188         exit(0);
189     }
190     else
191         printf("Socket successfully created..\n");
192     bzero(&servaddr, sizeof(servaddr));
193 
194     // assign IP, PORT
195     servaddr.sin_family = AF_INET;
196     //servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
197     servaddr.sin_addr.s_addr = inet_addr("127.0.0.1");
198     servaddr.sin_port = htons(port);
199 
```

```
200     // connect the client socket to server socket
201     if (connect(sd, (struct sockaddr*)&servaddr, sizeof(servaddr)) != 0) {
202         printf("connection with the server failed...\n");
203         exit(1);
204     }
205     else
206         printf("connected to the server..\n");
207 
208     // function for chat
209     init_serv(sd);
210 
211     // close the socket
212     close(sd);
213 }
```

B Server Code

```c
#include <dirent.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/socket.h>
#include <stdlib.h>
#include <netinet/in.h>
#include <string.h>
#include <dirent.h>
#include <sys/stat.h>

#define FILENAME_SIZE 255
#define ARG_SIZE 4097
#define SERVER_TCP_PORT 8080

struct pdu
{
 char type;
 int length;
 char data[ARG_SIZE];
} rpdu, tpdu;


int init_client(int cli_sock){

  char type_recv;
    DIR *folder;
  char current_folder[ARG_SIZE];
  char filename [ARG_SIZE+FILENAME_SIZE];
  struct dirent *dir;
  int i=0,size=0,flag =0;
  FILE* fp;
  struct stat filestat;

  strcpy(current_folder,".");

  tpdu.type='R';
    write(cli_sock, &tpdu, 1);//1B

  while(1){

    if(flag==0)
      read(cli_sock, &rpdu, sizeof(rpdu));
    else{
      flag=0;
      printf("flag removed\n");
      }
    type_recv = rpdu.type;

    switch(type_recv){

    case 'P':
      folder = opendir(rpdu.data);

      if(folder == NULL){
        printf("cd: Unable to read directory\n");
        tpdu.type = 'E';
        strcpy(tpdu.data,"Cannot change to specfied directory");
        tpdu.length=strlen(tpdu.data);
        write(cli_sock, &tpdu,sizeof(tpdu));

            }else{
        printf("cd: Current directory has been changed!\n");
        strcpy(current_folder,rpdu.data);
        tpdu.type='R';
        write(cli_sock, &tpdu,sizeof(tpdu));
```

```
66                  closedir(folder);
67          }
68          break;
69      case 'L':
70          if(strcmp("current_dir",rpdu.data)==0)
71              folder = opendir(current_folder);
72          else
73              folder =opendir(rpdu.data);
74
75          if(folder == NULL){
76
77                  printf("ls: Cannot list specified directory\n");
78              tpdu.type = 'E';
79              strcpy(tpdu.data,"Cannot list specified directory");
80              tpdu.length=strlen(tpdu.data);
81              write(cli_sock, &tpdu,sizeof(tpdu));
82
83                  }else{
84                    printf("ls: Compiling and sending list:\n");
85                i=0;
86                tpdu.type='I';
87                 while ((dir = readdir(folder)) != NULL){
88                           //printf("%s\n", dir->d_name);
89
90                //if the directory file names is longer than the data size , then send the
    compiled file and folder names and create new pdu to be sent
91                if((strlen(dir->d_name)+i+1)>ARG_SIZE){
92                  tpdu.length=strlen(tpdu.data);
93                  write(cli_sock, &tpdu,sizeof(tpdu));
94                  memset(&tpdu, 0, sizeof(tpdu));
95                  tpdu.type='I';
96                  i=0;
97                }
98                //copy the data to tpdu and append new line after each entry
99                memcpy(tpdu.data+i,dir->d_name,strlen(dir->d_name));
100               memcpy(tpdu.data+strlen(dir->d_name)+i,"\n",1);
101               i=i+strlen(dir->d_name)+1;
102                   }
103
104         tpdu.length=strlen(tpdu.data);
105         write(cli_sock, &tpdu,sizeof(tpdu));
106             closedir(folder);
107           }
108
109     break;
110 case 'D':
111   memcpy(filename,current_folder,strlen(current_folder));
112   strcat(filename, "/");
113   strcat(filename, rpdu.data);
114   fp = fopen(filename,"r");
115
116     if(fp < 0){
117       printf("\nFile does not exist\n");
118       tpdu.type = 'E';
119       strcpy(tpdu.data,"File not found!!!");
120       tpdu.length=strlen(tpdu.data);
121       write(cli_sock, &tpdu,sizeof(tpdu));
122     }else{
123       printf("\nSending File, %s, to Client\n",rpdu.data);
124       stat(filename, &filestat);
125             size = filestat.st_size;
126
127
128       tpdu.type='F';
129       while(size>ARG_SIZE){
130         fread(&tpdu.data, ARG_SIZE,1,fp);
131         tpdu.length=strlen(tpdu.data);
132         write(cli_sock, &tpdu, sizeof(tpdu));
```

```
133             size = (size - (ARG_SIZE));
134
135           }
136           //send final data packet
137           memset(&tpdu, 0, sizeof(tpdu));
138           tpdu.type='F';
139           fread(&tpdu.data, size,1,fp);
140           tpdu.length=strlen(tpdu.data);
141           write(cli_sock, &tpdu,sizeof(tpdu));
142           //send ready data packet
143           memset(&tpdu, 0, sizeof(tpdu));
144           tpdu.type='R';
145           write(cli_sock, &tpdu,sizeof(tpdu));
146         }
147       break;
148     case 'U':
149
150       memcpy(filename,current_folder,strlen(current_folder));
151       strcat(filename, "/");
152       strcat(filename, rpdu.data);
153
154       fp = fopen(filename,"w");
155
156         if(fp < 0){
157           printf("\nError writing to file\n");
158           tpdu.type = 'E';
159           strcpy(tpdu.data,"Error writing to file");
160           tpdu.length=strlen(tpdu.data);
161           write(cli_sock, &tpdu,(tpdu.length+1+4+5));
162         }else{
163           printf("\nRecieving File, %s, from Client\n",rpdu.data);
164           tpdu.type='R';
165             write(cli_sock, &tpdu,1);//1B for type, 4B for length, 5B overhead
166           read(cli_sock, &rpdu, sizeof(rpdu));
167
168           while (rpdu.type =='F'){
169             if(rpdu.length<ARG_SIZE){
170               fwrite(rpdu.data , 1 , rpdu.length, fp);
171               break;
172             }else{
173               fwrite(rpdu.data , 1 , rpdu.length, fp);
174               read(cli_sock, &rpdu, sizeof(rpdu));
175             }
176           }
177           if(rpdu.type !='F'){
178             flag=1;
179           }
180           fclose(fp);
181         }
182       break;
183
184     default:
185       printf("Client closed connection!\n\n");
186       exit(0);
187       break;
188     }
189     memset(&tpdu, 0, sizeof(tpdu));
190   }
191
192 }
193
194 int main (int argc, char *argv[]) {
195
196     int server_fd, sd, valread;
197     struct sockaddr_in address;
198     int opt = 1;
199     int addrlen;
200     int port;
```

```
201
202     switch(argc){
203   case 1:
204     port = SERVER_TCP_PORT;
205     break;
206   case 2:
207     port = atoi(argv[1]);
208     break;
209   default:
210     fprintf(stderr, "Usage: %s [port]\n", argv[0]);
211     exit(1);
212   }
213
214
215     // Creating socket file descriptor
216     if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0)
217     {
218         fprintf(stderr, "socket failed\n");
219         exit(EXIT_FAILURE);
220     }
221
222     // Forcefully attaching socket to the port 8080
223     if (setsockopt(server_fd, SOL_SOCKET, SO_REUSEADDR | SO_REUSEPORT,
224                                                 &opt, sizeof(opt)))
225     {
226         fprintf(stderr,"setsockopt\n");
227         exit(EXIT_FAILURE);
228     }
229     address.sin_family = AF_INET;
230     address.sin_addr.s_addr = INADDR_ANY;
231     address.sin_port = htons( port );
232
233     // Forcefully attaching socket to the port 8080
234     if (bind(server_fd, (struct sockaddr *)&address,
235                                 sizeof(address))<0)
236     {
237         fprintf(stderr,"bind failed\n");
238         exit(EXIT_FAILURE);
239     }
240     if (listen(server_fd, 5) < 0)
241     {
242         fprintf(stderr,"Listen failed\n");
243         exit(EXIT_FAILURE);
244     }
245     printf("Listening on %d\n", port);
246     while(1) {
247         if ((sd = accept(server_fd, (struct sockaddr *)&address,
248                         (socklen_t*)&addrlen))<0)
249         {
250             fprintf(stderr,"Accept Error");
251             exit(EXIT_FAILURE);
252         }
253         switch(fork()) {
254             case 0:
255                 close(server_fd);
256                 printf("Received connection\n");
257                 exit(init_client(sd));
258       break;
259             default:
260                 close(sd);
261                 break;
262             case -1:
263                 fprintf(stderr,"Fork Error\n");
264                 break;
265         }
266     }
267     return 0;
268 }
```