UCD School of Electrical and Electronic Engineering

EEEN20060 Communication Systems

Internet & TCP Report

Author: Weiheng Xia **Student Number: 17204875**

Team Number: **Select:** Option 2 client

Working with: Andrew Moore

Collaborating with: Marc Gannon , Beibei Zhu

Introduction

The aim of this assignment was to write software to transfer a file over the Internet. We chose option 2 so we aimed to design our own application layer protocol and implemented it with both client and server program. I worked on the client code with Andrew Moore and we designed our protocol with Beibei Zhu and Marc Gannon, who implemented the server side. We tested our program and debugged it together.

TCP

In our assignment, we used TCP (transmission control protocol) to set up connections between devices, and to transmit files. The services we used were listed below:

- Firstly, we used *TCPcreateSocket ()* function to initialize and create a socket.
- Secondly, we used *TCPclientConnect()* function to request a connection with the server. It needs 3 arguments: the created socket, the IP address, and the port number.
- Thirdly, *send()* function was used to send the client's request to the server, which requires an identifier for a connected socket, a pointer to an array of bytes to send, and the number of bytes to send.
- Fourthly, recv() function was used to receive the response from the server. We also put it into our own function called "recieve file", to make our code clearer.
- At last, we used TCPcloseSocket() function to shut down the connection.

Our Protocol (for option 2 only)

In our protocol, the client would send requests to the server and then get responses from it.

To identify the different parts of requests and responses, the protocol used question mark "?" to separate them. And an end marker "\n" was added at the end of requests and responses to indicate that this is the end of requests or responses.

For example, in the client's request, the first part is "U" or "D", representing Uploading mode or Downloading mode. Then there would be a "?" mark. The second part is the name of the file. Then there would be an end mark "\n".

In downloading mode, the server would send header like this:

i.	"Y"/ "N" (Representing whether or not it has the file)	-- FOLLOWED BY "?"
ii.	The file name (If the server has the file and says "Y")	-- FOLLOWED BY "?"
iii.	The size of the file	-- FOLLOWED BY "?"
iv.	"\n" (The end marker, followed by the file to send)	

In uploading mode, the client would send header like this:

i.	"U/D"(The mode name)	-- FOLLOWED BY "?"
ii.	The file name	-- FOLLOWED BY "?"

iii. The size of the file -- FOLLOWED BY “?”

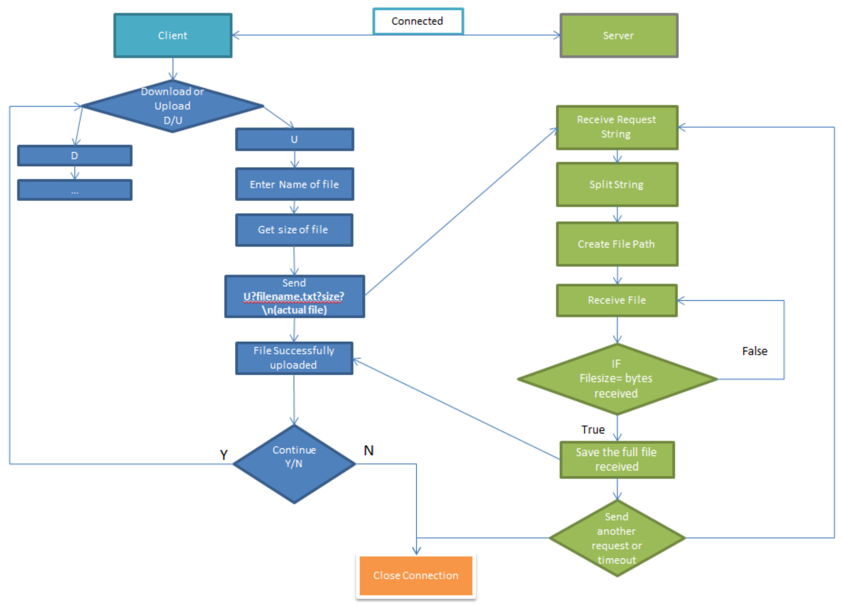
iv. “\n” (The end marker, followed by the file to upload) -- FOLLOWED BY “?”

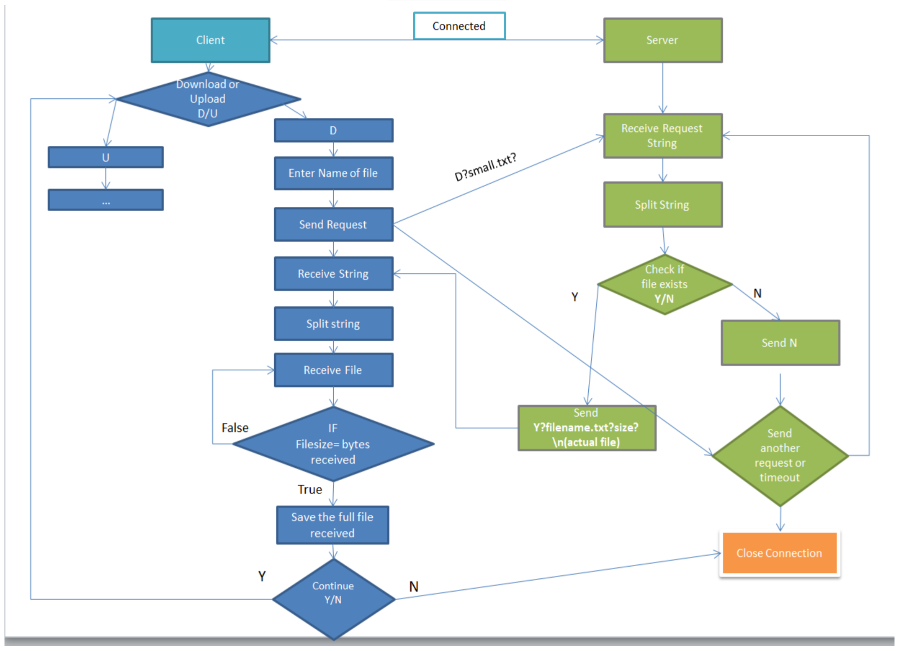
The end mark “\n” at the end of header showed that this was the end of header and the beginning of the file.

Then the receiver would check the size of the file, it had received, with the ideal file size sent in the header. If they were the same, then it meant the end of the file.

Our Program

Our program has two modes to choose, one is Downloading and the other is Uploading. To demonstrate our code, there are two flow charts made by my teammate Andrew Moore below:





The structure of our code was to get requests from the user first, then send the requests to the server, and finally get response from the server. And after each transmission, the client would ask the user again to continue process or to stop it.

The interaction between the user and the client happened mainly when client asked the user to type in the "Mode+ Filename". The code for request input is shown below:

```
//get the U/D,file name
printf("Please enter request mode(U/D):");
scanf(" %c", &mode);
printf("Please enter file name:");
scanf("%s", name);
```

After each transmission, the client would ask the user again to continue process or to stop it:

```
printf("Would you like to continue the process ?\n");
 scanf(" %c", &close);
```

We also created several functions to help organize our code :

Function Stringsplit (To extract the header)

```
char *stringsplit(char response[]){
const char s[2] = "\n";
char *token;

/* get the first token */
token = strtok(response, s); // split the header and the file by "\n"
if(token == NULL){
   printf("Header cannot be found! \n");
   return 0;
}
return token;                //extract to get header
}
```

This "stringsplit" function mainly deals to separate header and the file. It uses the function strtok() , and uses "\n" as a symbol to differentiate the header and the file. Therefore the receiver can get the header, and extract the information using the next following function "status_information"

Function status_information (To get the file name and file size from the header)

```
int status_information(char header[]) {
   const char s[2] = "?";
   char *token;
   int count;
   count = 0;
    int index;

   /* get the first token */
   token = strtok(header, s);
   count++;
   printf("Is the file there(Y/N)? |%s|\n", token);
   if(strcmp(token,"n") == 0 )
     printf("no file exists");
   else{

  /* walk through other tokens */
   while( count<3 ) {
      token = strtok(NULL, s);
      count++;
      if(count == 2){
         for(index=0;index<200;index++){
            file_name[index] = token[index];
//when counts is equal to 2, we can get the file name
         }
         printf("file_name is : |%s|\n", token);
         }
      else if(count == 3){
         size_of_file = atoi(token);
//when counts is equal to 3, we can get the file size
         printf("%d", size_of_file);
         printf("The size of file is : %d\n", size_of_file);
         return size_of_file;
         }
      }
   }
   return -1;
}
```

This “status_information” function uses token “?” to extract the different parts of the header, and to get the “Y/N”, file name, and file size. It still mainly uses “Strtok()” function, and uses “?” as the symbol.

Function recieveFile

```
int recieveFile(char *fullFileName, int downloadSize, SOCKET clientSocket){
    int numRx = 0;

    char fileData[BLOCKSIZE+1];
    FILE *fp = fopen(fullFileName, "w+b");

    while(bytesRead < downloadSize){
        numRx = recv(clientSocket, fileData, BLOCKSIZE, 0);
        // numRx will be number of bytes received, or an error indicator
(negative value)

        if( numRx < 0)  // check for error
        {
         printf("Problem receiving, maybe connection closed by client?\n");
         printError();   // give details of the error
         fclose(fp);
         return 1;   // set the flag to end the loop
        }
        else if (numRx == 0)  // indicates connection closing (but not an
        error)
        {
            printf("Connection closed by server\n");
            fclose(fp);
            return 1;   // set the flag to end the loop
        }
        else// numRx > 0 => we got some data from the client
        {
            bytesRead += numRx;
            fwrite(fileData, numRx, 1, fp);

        }
    }
    fclose(fp);
    return 0;
}
```

We shared this function “recieveFile” with our collaborating team. This function uses “recv()” function to receive files. And it used numRX as a counter to count the number of bytes received and compared it with the ideal file size. When the numRx was equal to the file size, then it would stop receiving and close the file.

Function findFileSize

```
int findFileSize(char *fName){
    FILE *fp;
    long size;    //file size
    int ret;      //return code from functions

    fp = fopen(fName, "rb"); //open for binary read
    ret = fseek(fp, 0, SEEK_END); // seek to end of file
    if (fp == NULL){
                    //say NO to client
       return -1;
    }
    if ( ret !=0){
      perror("Error in fseek"); // can't seek for the end, error
      fclose(fp);
      return -1;
    }
    return size = ftell(fp); // get current file position from the start
}
```

We shared this function “findFileSize” with our collaborating team. The goal of this function was to find out the file size.

Function addFilePath

```
char *addFilePath(char *fullFileName, char *fileName){

    //Adding folder name in front
    char *filePath = "./TCPstorage/";
    int fullFileSize = strlen(filePath) + strlen(fileName) + 1;

    if((fullFileName = (char *)calloc(fullFileSize, sizeof(char))) == 0){
        perror("sendfile:");
        return NULL;
    }

    strcat(fullFileName, filePath);
    strcat(fullFileName, fileName);

    return fullFileName;
}
```

We shared this function “addFilePath” with our collaborating team.

Testing

We have tested the program both on a laptop and on lab computers. We encountered a lot of failures in the process and debugged for a lot of times to figure out the problems.

At first, the program couldn’t recognize the header and cannot extract the header information. As we debugged the code, we found that the return value type of function “stringsplit” was wrong. Then we changed the type of return value of “stringsplit” function to be a pointer, which fixed this problem.

What’s more, we discovered that when downloading, the size of the file was always 1KB. Then we tried to debugged by printing the size of the file we received, and I found out that the size we were using was the size of the header, instead of the size of the file. So we changed the variable to the number of bytes we received as the file, and it went successfully.

In order to make our program more flexible, we also added a function for the client so that it could send multiple requests, without closing the connect socket. It also caused some trouble, but we figured it out by adding a loop in the client code, to make the client keep running the process until it wants to stop.

Attached below are the outputs of our program in client and server:

Client:

```
Communication Systems client program



WSAStartup succeeded
Socket created
Enter the IP address of the server: 127.0.0.1
Enter the port number on which the server is listening: 32980
Trying to connect to 127.0.0.1 on port 32980
Connected!
Please enter request mode(U/D):D
Please enter file name:crest.jpg
Sent 12 bytes, waiting for reply...
This is the whole header received: |y?crest.jpg?5190|
Is the file there(Y/N)? |y|
file_name is : |crest.jpg|
The size of file is : 5190
receive file size = 5190
The transmission is successful
Would you like to continue the process ?
y
Please enter request mode(U/D):U
Please enter file name:boat.jpg
Sent 18 bytes, waiting for reply...
File sent successfully.
Would you like to continue the process ?
n

Client is closing the connection...
Socket shutting down...
Socket closed, 0 remaining
WSACleanup returned 0

Press enter key to end:
```

Server:

```
/******************

Communication Systems server program

WSAStartup succeeded
Socket created
Socket bound to port 32980, server is ready

Listening for connection requests...

Accepted connection from port 63995 on 127.0.0.1
Request contains end marker

Request received, 12 bytes: 'D' 'crest.jpg' '0'
```

```
Sent response of 17 bytes: y?crest.jpg?5190

File sent successfully.
Request contains end marker

Request received, 18 bytes: 'U' 'boat.jpg' '224884'
Sent response of 18 bytes: y?boat.jpg?224884

Connection closed by client

Server is closing the connection...
Connection already closed by the other side
Socket closed, 1 remaining
Closing an idle socket
Socket closed, 0 remaining
WSACleanup returned 0

Press enter key to end:
```

Conclusion

The program that we coded perfectly implemented our protocol. And we also added some more features, such as the waiting loop for more requests.

We collaborated efficiently as a big group and worked together every week having a 2 or 3 hours' group meeting. The coding work pushed me to become more skilled in C language, and I really enjoyed the process of finding problems and solving problems by ourselves. I am glad that we applied the knowledge gained in class to practical, so that I can understand the use of the knowledge and pushed me to study more by myself. It was really an interesting course and an interesting lab.