UCD School of Electrical and Electronic Engineering

EEEN20060 Communication Systems

Internet & TCP Report – Option 1

Name: Ruairi Hogan **Student Number: 21432816**

Name: Alannah Mehigan **Student Number: 21343546**

Brightspace Group: 52

By submitting this report through Brightspace, we certify that ALL of the following are true:

1. We have read the *UCD Plagiarism Policy* (available on Brightspace). We understand the definition of plagiarism and the consequences of plagiarism.
2. We recognise that any work that has been plagiarised (in whole or in part) may be subject to penalties, as outlined in the document above.
3. We have not plagiarised any part of this report. The work described was done by the team, and this report is all our own original work, except where otherwise acknowledged in the report.

Introduction

The aim of this assignment was to write a program that sent a HTTP 1.1 request, for a file, to a server at given URL and to process the response and save it to a file.

To achieve that, we had to write a C code to get a user inputted URL separate the host name from the path, find the IP address of the host of the server and build a HTTP 1.1 request to get the file from the server. We first opened a socket, connected the socket to the given IP address and sent a request via TCP/IP protocol. Then we got the response and processed it, created a file and saved the file data to this file.

Although for the majority of time we worked together to write the code, Alannah focused on handling user input and building the request, whereas Ruairi focused on processing the response from the server. We also wrote those respective sections of this report. Both of us conducted the testing together and worked together to finish the report.

HTTP version 1.1 (Alannah)

HTTP allows for several types of requests, but we only used the GET request.

Request Structure

The first line of the GET request consist of the path to the file to be retrieved (e.g., /images/picture.jpg). The next line consists of the host name of the server (e.g., www.myserver.ie). The end of each line is marked by "\r\n".

The end of the request is marked by "\r\n\r\n"

Response Structure

The response should begin with "HTTP/1.1".

If the request has been successful, that marker will be followed by a number from 200-299 for successful requests and then an OK (e.g., "200 OK").

The end of the header lines is marked by "\r\n"

After the last line in the header, two consecutive end of line markers ("CR LF CR LF" or in C: "\r\n\r\n"). This called the end of header marker and is used to find when to start reading the data from.

Client Program

The program that we wrote first creates a socket to connect to a web server. It then prompts the user to enter a URL of a website that uses the HTTP protocol. Then it finds the IP address of the server of this website. The program then has to send a HTTP 1.1 request. To do this we need to find the host name and the path to the file we want. The program finds the host name by splitting the URL up until the first forward slash. It reads in the URL, taking for granted that there will be a "HTTP://" in front of the part of the URL we want, so it doesn't read it in. The program then creates a HTTP 1.1 request. This consists of using the GET request in HTTP 1.1. It uses the previously obtained host name and path and makes a string to send on TCP. It then waits for a response from the server. If the response is valid, the program prints the header up until the end of header marker in the terminal. It then writes any data bytes after the end of header marker into a file of file name given in the last section of the path. Then the code uses a while loop to repeat for another URL if the user wants to.

Handling User Input (Alannah)

Before getting the user input we use the TCPcreateSocket() function to create a socket which will connect to the server the user wants.

We first prompted the user to input a URL and gave an example format using the printf() statement. The `scanf("http://%s", URLstr);` function is then used to put the user input into a string array called URLstr. We then have to process this URL so that we can get the host name of the server. We use the strtok() function along with a "/" delimiter to create a token of the host name.

We then copy this token to a string array called hoststr. It then uses the function getIPaddress() that was given to us to find the IPaddress of the server.

We then use the TCPclientConnect() function using the socket we have created. This function creates to the server with the IP address we have found on port 80 (TCP always uses port 80). It will stop the code if an error is found.

Building the Request (Alannah)

The program then builds the request to send to this server. We have already found the host name to connect the socket to the server. We now make another token of the URL but this time we don't give it a delimiter, so it gives us the rest of the URL (the path) and copy that to a string array called pathstr. We then build the string for the request using the snprintf() function, the first line is the GET request and the path of the file we want to retrieve. This is then followed by a "\r\n" and then the host name is given, followed by "\r\n\r\n". This is given to a string array called request. We now send the request using a TCP function called send() and have an if statement to detect an error.

Processing the Response (Ruairi)

This program receives the data by iterating through a while loop for each block of data that comes in with the max response size to be 4000 bytes this is to ensure that the full header is always going to be in the first block. When we receive the response, we let the recv() function return to a variable called numRx which is the number of bytes received in that specific block. If numRx is not greater than zero, we know an error was found and we stop the code, and it runs to the end of the while loop and asks the user if they want to repeat for a different URL.

If numRx is greater than zero, the program adds numRx to numBytes and then enters an if statement. The if statement is to decide if the program is processing the header or the actual data from the file. This is determined by a variable called filebody which is set to 0 by default and gets changed to 1 after processing the header of the response.

When in the if statement for processing the header. The basic algorithm of this part of the code is to put the block received into a string called response, then check for the end of header marker ("\r\n\r\n"), and since the max response is 4000 bytes, the end of header marker is going to be in the first block. Sometimes servers can send the first block as the header only but that will work too. If the header is somehow more than 4000 bytes, the program will check the second block and so on until it finds the end of header marker. It uses the strstr() function in C to find the end of header marker. If the program found it, it created a pointer called loc to the first position of the end of header marker in the response.

```
// Check if the end of header marker is in the response
          response[numBytes] = '\0';
          loc = strstr(response, "\r\n\r\n");
          if(loc != NULL)
         {
```

Then there's an if statement that checks if the variable loc has changed or is it NULL, if it's NULL it means that the end of header marker hasn't been put into the totalResponse variable yet.

If it has found the end of header marker, it enters the if statement. We then wrote a code to find the name to call the file. This was done by getting the pathstr that we had for the request and making a token of it until the last "/" because the file name is the last part of the path. We used strtok() to do this.

```
// Finding the Name to call the file

            token = strtok(pathstr, "/");
            while (token != NULL) {
                name = token;
                token = strtok(NULL, "/");
            }
```

The program then changes the loc pointer 3 addresses forward, so it's at the end of the "\r\n\r\n" then it changes the last \n to a '\0' so it terminates the response string at the end of the header. Then the pointer loc goes forward 1 address so it's at the start of the file data.

```
/*Terminating the response string after the end of
header marker and moving loc to point to the remaining data*/
          loc += 3;
          *loc = '\0';
          loc += 1;
```

The program then opens a file with the filename that we've already found and then writes the rest of the data in the block that isn't the header, to the file. Then a printf statement prints the header in the terminal (the response array).

```
     val += fwrite(loc,1,(numRx - strlen(response)-1),fp);
```

Then it finds the content length of the response. This was given to us in the "Content-Length:" line in the header. The program first searches for the substring "Content-Length:" then it creates a pointer to where it finds the sub string. It then creates a token of the data until the next line, using delimiter "\n". It then uses the atoi function to get the integers from the string.

```
// Finding the Content Length
          content = strstr(response, "Content-Length:");
          content = strtok(content,"\n");
          content_length = atoi(&content[16]);
```

Then the filebody gets changed to 1 so the next time the code iterates through the while loop, it writes everything to a file, as only the data is left.

An if statement stops the retrieval of data blocks by saying that when the number of bytes written to the file is greater than or equal to (the greater than is for errors, if they do occur) the content-length, it stops the code. The number of bytes written so far is gotten from an integer variable that adds the return value of the fwrite statements every time they're called. The file is then closed.

```
if(content_length <= val)
```

The program then closes the socket and asks the user if they want to repeat the code again for a different URL.

```
// Initialise winsock, version 2.2, giving pointer to data structure
retVal = WSAStartup(MAKEWORD(2,2), &wsaData);
if (retVal != 0)  // check for error
{
   printf("*** WSAStartup failed: %d\n", retVal);
   printError();
   return 1;
}
 printf("WSAStartup succeeded\n" );
```

Testing (Alannah and Ruairi)

- The first test we performed was getting different files from different web servers. We tried .html files, .jpg, .gif and .pdf and all of them worked correctly. These are some of the files we teste with 2 console window snippets:
    - http://faraday1.ucd.ie/modules/archive/systems/synchronous.pdf
    - http://faraday1.ucd.ie/pscc/images/photospscc18/thursday/IMG_1086_web.jpg

```
HTTP/1.1 200 OK
Date: Wed, 03 May 2023 17:18:58 GMT
Server: Apache/2.4.54 (Unix)
Last-Modified: Fri, 22 Jun 2018 15:05:48 GMT
ETag: "40cb3-56f3c5debab00"
Accept-Ranges: bytes
Content-Length: 265395
Content-Type: image/jpeg

This is the number of bytes received: 265628, This is the number of bytes actually written 265395
Client is closing the connection...
CloseSocket: Socket shutting down...
CloseSocket: Socket closed, 0 remaining
CloseSocket: WSACleanup returned 0
```

    - http://www.columbia.edu/~fdc/sample.html
    - http://faraday1.ucd.ie/images/logoUCDsmall.gif

```
HTTP/1.1 200 OK
Date: Wed, 03 May 2023 17:12:35 GMT
Server: Apache/2.4.54 (Unix)
Last-Modified: Wed, 22 Jul 2020 06:52:27 GMT
ETag: "1264-5ab022ecca0c0"
Accept-Ranges: bytes
Content-Length: 4708
Content-Type: image/gif

This is the number of bytes received: 4937, This is the number of bytes actually written 4708
Client is closing the connection...
CloseSocket: Socket shutting down...
CloseSocket: Socket closed, 0 remaining
CloseSocket: WSACleanup returned 0
```

  The program worked perfectly and retrieved all files with the correct size and formatting.

- We tested to see if problems arose with if there was a mistake using the scanf, invalid IP address, or if the file name was invalid. When we got errors on these we used if statements to detect them:

```
retVal = scanf("http://%s",
URLstr);

if( retVal < 0)
{
printf("*** Problem using
scanf\n");
stop = 1;       }
```

```
retVal =
getIPaddress(hoststr,
NULL, serverIPstr);

if( retVal < 0)
{
printf("*** Problem
finding IP address\n");
stop = 1;
```

```
if(fp == NULL){
perror("ERROR OPENING FILE");
stop = 1;
}
```

- We also tested it by decreasing the MAXRESPONSE size to 400 and increasing it to 40000 and the code still worked.
- The program ended with a request asking the user if they wanted to get another piece of information. When the user inputs ‘y’ the program runs again and asks the user for another input. If ‘n’ is entered the program closes and if any other character is entered the question is asked again, rejecting any characters other than ‘y’ and ‘n’.
- We tried to find an .MP3 to test with but could only find ones with URLs containing HTTPS.

Problems Faced

- One problem that we faced was that every time the code would iterate for another URL, the program would give an error code. This was because the buffer still had old user inputs in it, to fix this problem, we use fflush(stdin) to clear the buffer before the next URL was inputted.
- Another problem was not resetting the variables on every iteration for different URLs so some of the variables kept their values from the last iteration.

Conclusion

We succeeded in developing a program which can send a HTTP 1.1 request to a server at a given URL and processing the response to save the file to a specified location.

The program can handle various responses, including file formats and server responses. We were able to write this program through our learned knowledge about HTTP 1.1 protocol and how to use the CodeBlocks built-in libraries.

The program meets the requirement of the assignment but could be further improved by additional features such as the ability to download multiple files at once.

Every file evaluated in the testing was saved correctly and no errors were encountered.