

Assignment2_Final

by Shannon Boulting

FILE	16808_SHANNON_BOULTING_ASSIGNMENT2_FINAL_252504_137018310 3.DOCX (633.85K)		
TIME SUBMITTED	07-JUN-2020 08:17PM (UTC+0100)	WORD COUNT	1837
SUBMISSION ID	130265482	CHARACTER COUNT	10846

NETWORKED APPLICATIONS

Assignment 2



Figure 1 [1]

Shannon Boulting

Word count: 1080

Table of Contents

1. Introduction	3
1.1 Description of a Webserver	3
1.2 About HTTP	3
1.3 Socket Programming.....	3
1.4 HTTP Request/Response	3
1.5 HTTP Methods.....	4
2. Design.....	5
2.1 Prototype Design & Development	5
2.2 Webserver Design	5
2.3 Message Implementation	5
2.4 Server Logic Implementation	6
3. Adhering to Best Practice.....	8
3.1 Design Decisions	8
3.1.1 Select()	8
3.1.2 Select Pros & Cons	9
3.2 HTTP responses.....	9
4. Conclusion.....	12
5. Future Implementations	12
6. References.....	13

Table of Figures

Figure 1 [1]	0
Figure 2: HTTP sequence diagram	4
Figure 3: GET method implemented.....	4
Figure 4: State Machine Diagram.....	5
Figure 5: Client disconnection.....	6
Figure 6: Error message implementation	6
Figure 7: Closure of Winsock	7
Figure 8: Clearing of master and temp	8
Figure 9: adding listening socket.....	8
Figure 10: Setting up connections	8
Figure 11: Closure of Select()	9
Figure 12: HTTP Response - Request Successful.....	10
Figure 13: HTTP Response - Not Found	10
Figure 14: HTTP Response - Not Implemented.....	10
Figure 15: HTTP Response Format [3].....	11
Figure 16: HTTP Response code implementation.....	11
Figure 17: Handling Multiple Connections	12

Table of Tables

Table 1: Messages 5

1. Introduction

The aim of this documentation is to give an in-depth account of the design and development of a webserver.

1.1 Description of a Webserver

A web server is responsible for storing the contents of a website e.g. text, images and video which are encoded in HTML (Hypertext Markup Language), and also delivering this data to the client (in this case the browser) which requested it. A web server can communicate with a web browser via the HTTP (Hypertext Transfer Protocol) [2].

1.2 About HTTP

HTTP communications are carried out normally via TCP/IP connections with 80 being the default port, however other ports can be utilised [3].

The protocol used in HTTP is the application-level protocol. The HTTP protocol implemented within the webserver is known as HTTP/1.1 in line with RFC 2068 [4]. It provides a faster response and allows for multiple transactions to be undertaken on a single connection as well as providing cache support [5].

1.3 Socket Programming

For the implementation of TCP, socket programming is required. A socket allows for programmes to access the network so messages can be sent and received among applications on various networked machines [3].

The following process is followed in socket programming:

- 1) Creation of the socket [3]
- 2) Identification of socket
- 3) Waiting for incoming connection
- 4) Send and receive messages
- 5) Closing the socket [3].

1.4 HTTP Request/Response

Communication between the client and server is via requests and responses, as follows:

- 1) Client (browser) sends a HTTP request [6]
- 2) Request is received by the webserver
- 3) Request is processed by an application ran by the server
- 4) Server sends a HTTP response in return to the browser
- 5) Response is received by the browser [6].

The steps can be seen in the sequence diagram in Figure 2.

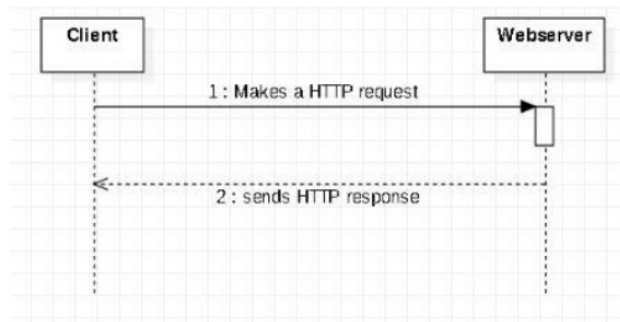


Figure 2: HTTP sequence diagram

1.5 HTTP Methods

HTTP identifies a set of request methods used to indicate an action which is required to be carried out. Within the webserver, the GET method is implemented to retrieve data (Figure 3) [7].

```
if (parsed.size() >= 3 && parsed[0] == "GET")
{
    //open the document in the local file system
    std::ifstream f("..\wwwroot" + parsed[1]);
```

Figure 3: GET method implemented

The file is retrieved through ifstream (input file stream class). The document is parsed out.

2. Design

This section will provide an insight into the design and development of the webserver.

2.1 Prototype Design & Development

The key design decisions implemented were:

- 1) How it will respond to requests a web browser will make
- 2) How multiple connections will be handled
- 3) How a HTML file will be transferred

2.2 Webserver Design

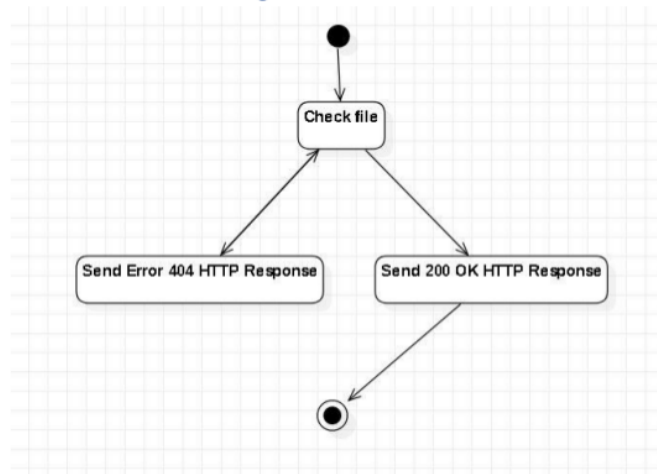


Figure 4: State Machine Diagram

Figure 4 displays a State Machine diagram of the webserver. The webserver checks the file and sends either an error message, if it is unable to read the file, or it will display the webpage content back to the client.

2.3 Message Implementation

Table 1 shows the messages which will be implemented within the webserver:

Message	Response(s)	Notes
File good	Successful response	HTTP response 200
File not good	Not found	HTTP response 404
No file	Not Implemented	HTTP response 501
No messages received	Client Disconnect	

Table 1: Messages

2.4 Server Logic Implementation

Upon implementing the server logic, the following were considered:

If no messages are received the client is disconnected and the socket is closed and cleared (Figure 5).

```
else //inbound message
{
    char buf[4096];
    ZeroMemory(buf, 4096);

    //receive message
    int bytesRec = recv(newfd, buf, 4096, 0);
    if (bytesRec <= 0)
    {
        //disconnect client
        ClientDisconnected(i);
        closesocket(newfd);
        FD_CLR(newfd, &master);
    }
}
```

Figure 5: Client disconnection

Error messages are implemented for creation of a socket, binding and listening so the user is continuously notified about errors when connections are occurring (Figure 6).

```
if (wsOk != 0)
{
    printf("Startup failed");
    exit;
}

//creating a socket
if ((s_socket = socket(AF_INET, SOCK_STREAM, 0)) == 0)
{
    printf("Error in socket");
    exit;
}

//bind the IP Address and port to a socket
sockaddr_in hint;
hint.sin_family = AF_INET;
hint.sin_port = htons(8080);
inet_pton(AF_INET, server_IP, &hint.sin_addr);

if (bind(s_socket, (sockaddr*)&hint, sizeof(hint)) < 0)
{
    printf("Error in bind");
    exit;
}

//notify winsock the socket is for listening
if (listen(s_socket, SOMAXCONN) == SOCKET_ERROR)
{
    printf("Error in listen");
    exit;
}
```

Figure 6: Error message implementation

As data is received the FD_CLR is implemented to clear the file descriptor within the set. WSACleanup () terminates the ws2_32.dll. The closesocket() function closes the socket identified (Figure 7).

```
//listening socket is removed from master file descriptor and closed to prevent further connections
FD_CLR(s_socket, &master);
closesocket(s_socket);
WSACleanup();
return 0;
```

Figure 7: Closure of Winsock

3. Adhering to Best Practice

This section demonstrates how the development of the webserver has adhered to best practice.

3.1 Design Decisions

Select() function was implemented to handle the multiple connections the webserver will receive. Moreover, it can be implemented to identify the status of a socket(s) which are waiting to carry out synchronous I/O [8].

3.1.1 Select()

Within select there are the following functions: FD_ZERO which is utilised to clear the fd_set (Figure 8) and FD_SET which is implemented to add a descriptor to the fd_set (Figure 9).

```
//clearance of master & temp
FD_ZERO(&master);
FD_ZERO(&read_fds);
```

Figure 8: Clearing of master and temp

```
//add the listening socket so that server is notified on incoming connections
FD_SET(s_socket, &master);
```

Figure 9: adding listening socket

Figure 10 demonstrates how a copy of the master file was created where the copy will contain sockets which are accepting the inbound connection requests/messages.

```
for(;;)
{
    //copy of master file
    read_fds = master;

    //seeing incoming connections
    int fdmax = select(0, &read_fds, NULL, NULL, NULL);

    //loop connections
    for (int i = 0; i < fdmax; i++)
    {
        //if(FD_ISSET(newfd, &read_fds))
        //{
            //if((newfd == s_socket)
            //{
                SOCKET newfd = read_fds.fd_array[i];

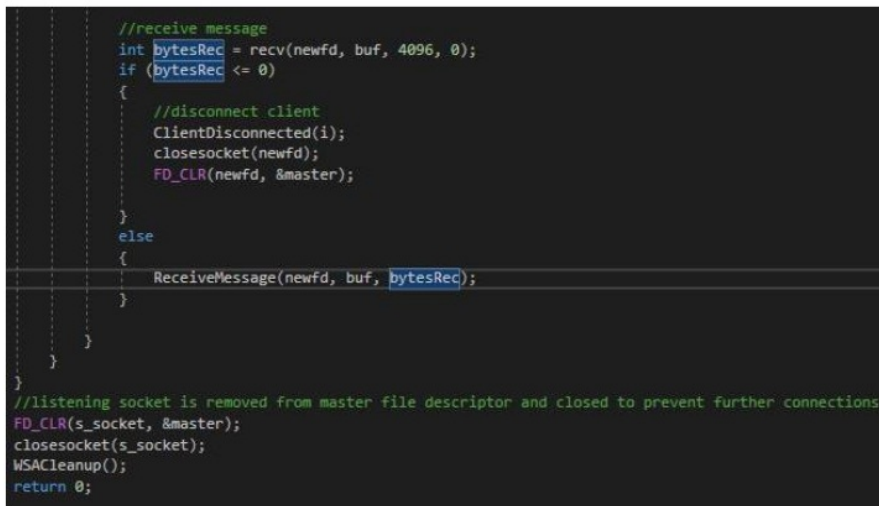
                //check inbound communication
                if (newfd == s_socket)
                {
                    //accept new connection
                    SOCKET client = accept(s_socket, nullptr, nullptr);

                    //add to master set
                    FD_SET(client, &master);
                    ClientConnected(client);
                }
            }
        }

        else //inbound message
        {
            char buf[4096];
            ZeroMemory(buf, 4096);
        }
    }
}
```

Figure 10: Setting up connections

In the final implementation, the listening socket is removed from the file descriptor set and closed. This is to prevent further connections. The socket number is retrieved and removed from the master file list and closed (Figure 11).



```
//receive message
int bytesRec = recv(newFd, buf, 4096, 0);
if (bytesRec <= 0)
{
    //disconnect client
    ClientDisconnected(i);
    closesocket(newFd);
    FD_CLR(newFd, &master);
}
else
{
    ReceiveMessage(newFd, buf, bytesRec);
}
}
}

//listening socket is removed from master file descriptor and closed to prevent further connections
FD_CLR(s_socket, &master);
closesocket(s_socket);
WSACleanup();
return 0;
```

Figure 11: Closure of Select()

3.1.2 Select Pros & Cons

The select function allows the file descriptors to be monitored which are waiting for an event. The server socket(listening) notifies pending connections available which can be accepted(accept) [9].

Multithreading could have been implanted for the webserver. However, it can be difficult to debug, is said not to be scalable for many clients and there is the possibility of deadlocks occurring. In this case, select() is believed to be a more efficient way of handling multiple clients and is believed to be better for functionality and performance [10] [11].

3.2 HTTP responses

Upon a HTTP request being received the server will respond with a HTTP response. There are 5 different classes: informational, successful, redirect, client and server errors [12].

The webserver will have the following responses (Figures 12, 13 and 14) implemented:

- 1) The request has succeeded (200) displaying a successful response [13]

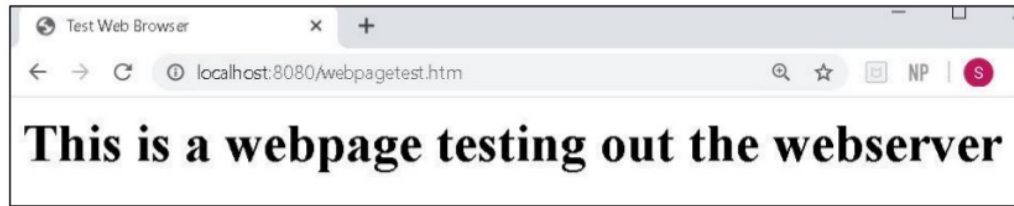


Figure 12: HTTP Response - Request Successful

- 2) Not Found (404) showing a client error [13]



Figure 13: HTTP Response - Not Found

- 3) Not implemented (501) showing server errors [13]

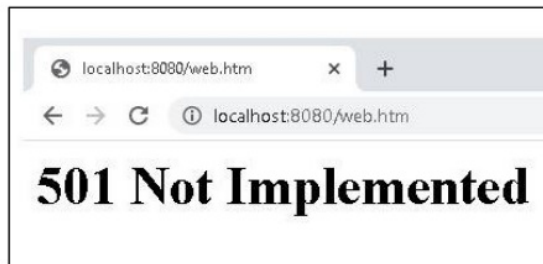


Figure 14: HTTP Response - Not Implemented

Figure 15 shows the format of a HTTP response.



Figure 15: HTTP Response Format [3]

The below is an example of a minimal HTTP Header requiring 3 parts:

- 1) HTTP/1.1 200 OK which indicates what HTTP version is being used and the Status code and message [3]
- 2) Content-Type: text/plain which indicates if the text being sent is plain. Images can be utilised as well
- 3) Content-Length: which indicates how many bytes is being sent by the server [3].

Figure 16 shows an example of the code implementation.

```
void sendOKresponse(int clientSocket, const char* msg, int length)
{
    std::istringstream iss(msg);
    std::vector<std::string> parsed((std::istream_iterator<std::string>(iss)), std::istream_iterator<std::string>());
    std::ifstream f(".\\accessfolder" + parsed[1]);
    std::string content = "<h1>404 Not Found</h1>";
    std::string str((std::istreambuf_iterator<char>(f)), std::istreambuf_iterator<char>());
    content = str;

    //document sent back to client
    std::ostringstream oss;
    oss << "HTTP/1.1 OK\r\n";
    oss << "Content-Type: text/html\r\n";
    oss << "Content-Length \r\n";
    oss << "\r\n";
    oss << content;

    std::string output = oss.str();
    int size = output.size() + 1;

    sendClient(clientSocket, output.c_str(), size);
}
```

Figure 16: HTTP Response code implementation

4. Conclusion

To conclude, the webserver can handle multiple connections and provide the appropriate HTTP responses to any requests made by the browser. The webserver adheres to best practice in a number of ways:

- Select() has been implemented to handle the multiple connections. As discussed, it is believed to better performance wise and a more efficient way to handle multiple connections (Figure 17)



Figure 17: Handling Multiple Connections

- Winsock has been implemented for socket programming and provides the user with error messages in the creation of a socket, binding and listening
- Disconnections have been handled and as well as appropriate measures for closure of the sockets and clean-up of Winsock
- The webserver responds to requests with 200, 404 and 501.

5. Future Implementations

The following are future implementations/recommendations for the webserver:

- Implementation of HTTP responses into a class
- Webserver able to handle different file formats such as jpegs, xml, txt, gif, png and zip
- The code for the responses could be condensed down and an alternative way to be coded researched
- 404 response could be placed into its own handler
- Further HTTP responses could be implemented to respond to further HTTP requests made such as 204 No Content, 301 Moved Permanently and 403 Forbidden.

6. References

- [1] N. M. Ciccarello, "Networking: The Icing on a Good Résumé or Business Idea," ie University, 7 June 2017. [Online]. Available: <https://www.ie.edu/insights/articles/networking-the-icing-on-good-resume-or-business-idea/>. [Accessed 2020 May 16].
- [2] NGINX, "What Is a Web Server?," NGINX, 2020. [Online]. Available: <https://www.nginx.com/resources/glossary/web-server/>. [Accessed 17 May 2020].
- [3] Medium, "HTTP Server: Everything you need to know to Build a simple HTTP server from scratch," Medium, 16 March 2018. [Online]. Available: <https://medium.com/from-the-scratch/http-server-what-do-you-need-to-know-to-build-a-simple-http-server-from-scratch-d1ef8945e4fa>. [Accessed 28 May 2020].
- [4] T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1," tools.ietf.org, June 1999. [Online]. Available: <https://tools.ietf.org/html/rfc2616>. [Accessed 4 June 2020].
- [5] J. Marshall, "HTTP Made Really Easy," J Marshall, 10 December 2012. [Online]. Available: <https://www.jmarshall.com/easy/http/>. [Accessed 25 May 2020].
- [6] w3schools.com, "What is HTTP?," W3schools.com, 2020. [Online]. Available: https://www.w3schools.com/whatis/whatis_http.asp. [Accessed 30 May 2020].
- [7] Mozilla, "HTTP request methods," Mozilla, 2020. [Online]. Available: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods>. [Accessed 5 June 2020].
- [8] Microsoft, "select function," Microsoft, 12 May 2018. [Online]. Available: <https://docs.microsoft.com/en-us/windows/win32/api/winsock2/nf-winsock2-select>. [Accessed 5 June 2020].
- [9] GNU, "Waiting for Input or Output," GNU, [Online]. Available: https://www.gnu.org/software/libc/manual/html_node/Waiting-for-I_002fO.html#Waiting-for-I_002fO. [Accessed 29 May 2020].
- [10] GeeksforGeeks, "Socket Programming in C/C++: Handling multiple clients on server without multi threading," GeeksforGeeks, 2020. [Online]. Available: <https://www.geeksforgeeks.org/socket-programming-in-cc-handling-multiple-clients-on-server-without-multi-threading/>. [Accessed 5 June 2020].
- [11] A. Veidenbaum, J. Kazuki, A. Hideo and A. Hideharu, High Performance Computing, Tokyo: Springer, 2003.
- [12] Mozilla, "HTTP response status codes," MDN web docs Mozilla, 2020. [Online]. Available: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Status>. [Accessed 29 May 2020].
- [13] Fielding, "6 Response," w3.org, 2016. [Online]. Available: <https://www.w3.org/Protocols/rfc2616/rfc2616-sec6.html>. [Accessed 5 June 2020].

- [14] S. Kelly, "Building a Web Server in C++ [VS 2017] Part 1," YouTube, 23 December 2018. [Online]. Available: <https://www.youtube.com/watch?v=Kc1kwm1WyVM>. [Accessed 29 May 2020].
- [15] S. Kelly, "Building a Web Server in C++ [VS 2017] Part 2," YouTube, 31 January 2019. [Online]. Available: <https://www.youtube.com/watch?v=YqEqjODUkWY>. [Accessed 29 May 2020].
- [16] L. Ponik, "C++ HTTP multithreaded server with socket programming PT1," YouTube, 5 November 2014. [Online]. Available: <https://www.youtube.com/watch?v=IP-tx3IUlh8>. [Accessed 27 May 2020].
- [17] Tenouk, "NETWORK PROGRAMMING," Tenouk, 2020. [Online]. Available: <https://www.tenouk.com/Module41.html>. [Accessed 7 June 2020].
- [18] S. Overflow, "How to get key values from HTTP post request using C++ and Boost Asio," Stack Overflow, 2020. [Online]. Available: <https://stackoverflow.com/questions/59659968/how-to-get-key-values-from-http-post-request-using-c-and-boost-asio>. [Accessed 7 June 2020].
- [19] Microsoft, "Getting Started with Winsock," Microsoft, 31 May 2018. [Online]. Available: <https://docs.microsoft.com/en-us/windows/win32/winsock/getting-started-with-winsock>. [Accessed 24 May 2020].

Assignment2_Final

GRADEMARK REPORT

FINAL GRADE

45/100

GENERAL COMMENTS

Instructor

A reasonable attempt made, with a working, if limited, web server. The report is also a tad brief.

Some of the sources used are pretty weak - more academic sources would've been preferable.

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14

PAGE 15