CA2: Programming Assignment Report

Name: Tsu Wei Quan
Matric ID: A0168411J

Explanation of Client Process

The client will firstly establish a connection to the server via the connect() function. After a successful connection, the client will proceed to send a HTTP GET Request.

GET /index.html HTTP/1.1\r\nHost: **127.0.0.1**\r\nConnection: **closed**\r\n\r\n

This first GET request to the server is to download the html text of the webserver. The GET request is created by inserting the index.html file name, ip address of the server and the connection type. This 3 information is enough to tell the server what the client expects.

After sending the HTTP GET request, the client waits for a HTTP response from the server. The server will proceed to send such a response.

HTTP/1.1 **200** OK\r\nDate: Mon, 06 Apr 2020 12:28:53 GMT\r\nServer: Apache/2.2.14 (Win32)\r\nLast-Modified: Wed, 22 Jul 2009 19:15:56 GMT\r\nETag: \"34aa387-d-1568eb00\"\r\nVary: Authorization,Accept\r\nAccept-Ranges: bytes\r\nContent-Length: 88\r\nContent-Type: **text/html**\r\nConnection: **closed**\r\n\r\n

The client receiving this response will check for the http status if it's 200, indicating a request success. The first HTTP response is assumed to carry the index.html data with it and the client now knows that it needs to download 3 objects from the index.html page.

If the client is non-persistent HTTP (Connection: closed), it will close the socket now. On the other hand, if the client is persistent HTTP (Connection: keep-alive), it will leave the socket open.

Before starting to send the HTTP GET request for a.jpg, if the client is non-persistent HTTP, it will re-establish connection via the connect() function. Now, the client will send a HTTP GET Request as such.

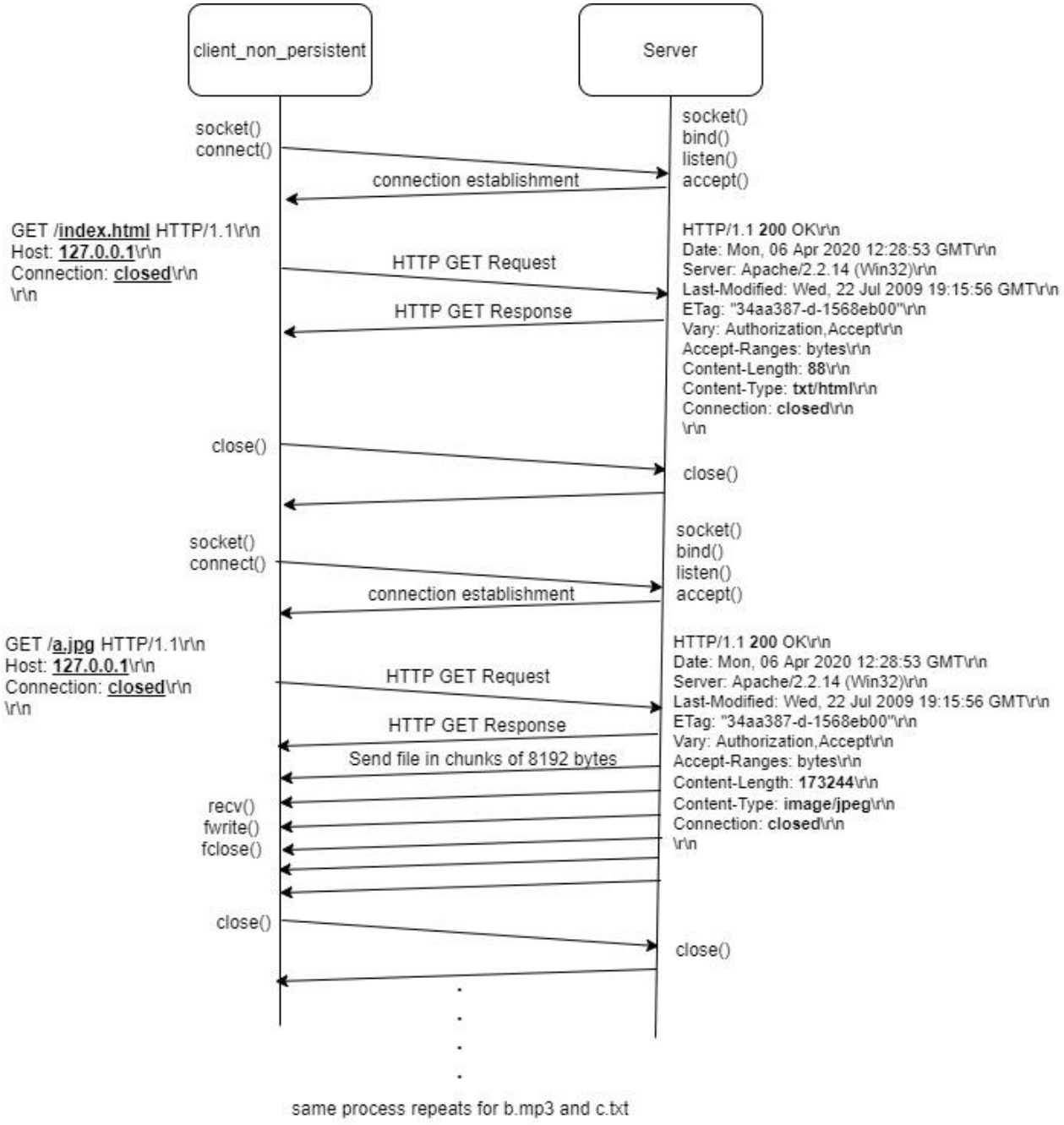
GET /a.jpg HTTP/1.1\r\nHost: **127.0.0.1**\r\nConnection: **closed**\r\n\r\n

The server receiving this request will firstly check if it is a GET command. Next, it will extract the filename, a.jpg and lastly it will take note if it's a persistent or non-persistent connection. The server proceeds to send a HTTP Response packet to the client first, following by sending a.jpg file in chunks of 8192 bytes.

HTTP/1.1 **200** OK\r\nDate: Mon, 06 Apr 2020 12:28:53 GMT\r\nServer: Apache/2.2.14 (Win32)\r\nLast-Modified: Wed, 22 Jul 2009 19:15:56 GMT\r\nETag: \"34aa387-d-1568eb00\"\r\nVary: Authorization,Accept\r\nAccept-Ranges: bytes\r\nContent-Length: **173244**\r\nContent-Type: **image/jpeg**\r\nConnection: **closed**\r\n\r\n

The client receives the HTTP response packet first, ensuring that it has a status 200. Next, the client extract the Content-Length field and take note of the number of bytes the incoming object has. The client now enters a while loop, expecting to receive multiple packets where the total size would add up to the content-length field. After the object has been downloaded, the client proceeds to close the socket if it's a non-persistent connection, otherwise the client leave the socket open. The client repeats the process where it proceeds to send the next HTTP GET request for the second object.

This is the communication process of what was described in the previous page.



If the client request for persistent connection, the socket will not close.

Explanation of Server Process

The server is implemented with the capability of handling multiple connection. It utilizes the select() function for synchronous I/O multiplexing.

For each new connection established to the server, the master socket, fd, is set in the fd_set. The FD_ISSET function will detect new connection established in fd and proceed to accept the connection. The socket description output by the accept() function is stored in the newfd array of size 30. This allow a maximum of 30 connections to the server. In all, each newfd element contains a socket descriptor. The socket descriptor is used to communicate with the client that is connected to the server.

```
if (FD_ISSET(fd, &readfds)) {
        if (next == max_client) next = 0;
        newfd[next++] = accept(fd, (struct sockaddr *)&client, &sockaddr_len);
}

for (int i = 0; i < max_client; i++) {

         if (FD_ISSET(newfd[i], &readfds)) {
         // handle connection here.
         }
```

For non-persistent connection, each new connection established after an object is download will open a new socket registered in newfd[*i*], whereas for persistent connection, the same socket is used for all objects downloaded.

The server would check if it's a GET request, it will toggle the Boolean variable, isPersistent, to true. It will also check if it's a persistent or non-persistent connection by extracting the word out from the GET request.

```
if (strstr(copyData, "keep-alive") != NULL) {

   printf("Requesting Persistent Connecton!\n");

   isPersistent = true;

} else if (strstr(copyData, "closed") != NULL) {

   printf("Requesting Non Persistent Connecton!\n");

   isPersistent = false;

}
```

Furthermore, the server would extract the filename of each request to know what the client wants.

```
// Check Request Type (index.html or a.jpg or b.mp3 or c.txt)

int requestItem = 0;

if (strstr(copyData, "index.html") != NULL) {

   requestItem = 0;
```

```
} else if (strstr(copyData, "a.jpg") != NULL) {
    requestItem = 1;
} else if (strstr(copyData, "b.mp3") != NULL) {
    requestItem = 2;
} else if (strstr(copyData, "c.txt") != NULL) {
    requestItem = 3;
}
```

After knowing which file the client desire, the server will send the file over in chunks of 8192 bytes. If the client uses non-persistent http, the server proceeds to close the socket and clear the bit in the fd_set. The *FD_CLR(newfd[i], &readfds);* was utilized.

Download Analysis

No.	Time taken to download (µsec)								
	a.jpg (173244 bytes)		b.mp3 (6367745 bytes)		c.txt (48658 bytes)		Whole run		
	Persistent	Non-Persistent	Persistent	Non-Persistent	Persistent	Non-Persistent	Persistent	Non-Persistent	Difference between non-persistent & persistent
1	9548	19122	153045	169545	61372	77905	229663	270684	41021
2	13631	14008	149885	170725	75710	75766	245926	264051	18125
3	13853	20541	145792	190059	71873	71128	237929	288911	50982
4	14052	17405	167661	169706	63325	66942	250439	257958	7519
5	13779	14161	168869	173912	76979	76596	265322	267267	1945
6	9716	17406	184534	187772	76670	80915	276704	291146	14442
7	9941	18849	168416	171139	77253	76915	259397	270286	10889
8	11256	18906	162407	157514	71771	71329	251338	251818	480
9	11193	18096	161130	159678	74054	81477	251860	262385	10525
10	15060	17981	175531	191701	92346	80899	288922	294845	5923

In the last column, Non-Persistent connection overall take a longer time to transfer all 3 objects over. This is because in non-persistent connection, for each object to be downloaded, it needs to open a connection. This results in 2 RTTs per Object which is an OS overhead for each TCP connection. Whereas in persistent connection, it re-uses the same socket, saving time in opening and closing socket after each object download.

HTTP Response Time for Non-Persistent:

Total time = 1 RTT (TCP connection) + 1 RTT (HTTP req/resp for html page) + 1 RTT (TCP connection) + 1 RTT (HTTP req/resp for a.jpg) + file transmission time (a.jpg) + 1 RTT (TCP connection) + 1 RTT (HTTP req/resp for b.mp3) + file transmission time (b.mp3) + 1 RTT (TCP connection) + 1 RTT (HTTP req/resp for c.txt) + file transmission time (c.txt)

Simplify:

Total time = 8 RTTs + file transmission time (a.jpg) + file transmission time (b.mp3) + file transmission time (c.txt)

Hence, 2RTT + file transmission time per object.

HTTP Response Time for Persistent:

Total time = 1 RTT (TCP connection) + 1 RTT (HTTP req/resp for html page) + 1 RTT (HTTP req/resp for a.jpg) + file transmission time (a.jpg) + 1 RTT (HTTP req/resp for b.mp3) + file transmission time (b.mp3) + 1 RTT (HTTP req/resp for c.txt) + file transmission time (c.txt)

Simplify:

Total time = 5 RTTs + file transmission time (a.jpg) + file transmission time (b.mp3) + file transmission time (c.txt)