Project 1: Web Server Implementation using BSD Sockets

Zhao Weng, 304946606

Honglin Zheng, 304947026

1. Give a high-level description of your servers design.
For the client, it is almost the same as the UDP protocol, it gets the socket and try to connect. After building the connection, it will write to the socket and read the results returned from the server. After that, client will close the socket. On the server side, the situation is different. First server creates the socket and it listens for connection. Server registers sigaction to reap zombie processes. Server moves on to accept the socket from client and opens a new child process to handle the data communication. Child process reads input from the client, parses the input, generates the response, and writes back to the client. Finally, child process exits and become zombie process to be reaped.

2. What difficulties did you face and how did you solve them?
Initially, we thought we can use single thread to handle all the cases. However, if we use single thread to accept and handle inputs from the client. All clients will share the same port on the server and gets the same response. So, according to the specification of TCP, main thread should be responsible for handling accept, and new child threads will be used to handle communication and return control back to the main thread. Data communication will use different ports on the server side.

3. Include a brief manual in the report to explain how to compile and run your source code
in the terminal, you first need to go to our project directory and type “make”. That will compile our source codes including client.c and server.c. First, you should start server by typing “./server 9500”. After that, you should start client by typing “./client localhost 9500”. Following those two steps, you can type in request from client. For image testing, you should type “GET /test.html HTTP/1.1”. For gif test, you should type “GET /test.gif HTTP/1.1”. For html test, you should type “GET /test.html HTTP/1.1”.

4. Include and briefly explain some sample outputs of your client-server (e.g. in Part A you should be able to see an HTTP request).
For example, for testing html request from client, the request and response are shown below.
request: GET /test.html HTTP/1.1
response:
<!DOCTYPE html>
<html>
<head>

```
</head>
<body>
        <p>This is a test html file</p>
</body>
</html>
```