CS 118 Project 1 Report

Web Server Implementation using BSD Sockets

Yukai Tu 204761085
Xufan Wang 204477479

January 31, 2017

Introduction

For this project, we implemented a web server in C using the BSD socket programming library on a 32-bit Ubuntu Virtual machine. The server implementation include two parts. For part A, the server can reiceive HTTP requests and then dump request messages to the console. For part B, one more functionality is added to the server, so that it can further parse the requests, generate an HTTP response message consisting of the requested file preceded by header lines, and send the respone back to the client.

Compilation and Usage

Tester can use the following steps to compile the server.

```
1.  Navigate to the directory containing the source files.
2.  Run the command "make".
2.  Run the command "chmod +x serverFork".
```

After compilation, use the command:

```
./serverFork <portnum>
```

to start the server. In the above command, <portnum> is a port number that the server will bind to, so the same port should be used for the client to send request.
After the server starts, the client can connent to the server by accessing following URL using a web browser.

```
http://<machine name>:<portnum>/<html file name>
```

In the above command, <machine name> should be the name of the server machine, and <html file name> should be the name of a file that the client would like to receive.

High-Level Description

For both Part A and Part B, the server builds a communication point through a port by using functions in the libraries:

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
```

which define different data sturectures and socket operations.

To be more specific, the server first use the `socket` function to create a socket. It then uses the fucntion `bind` to bind the new socket to a port on the local machine. The port number is specified by the user. After doing those, the server starts listen to the socket for any client requests, and this is done by using `listen` function. By now, the socket is ready and requests can be sent through it by the client.

The server accepts requests with the function `accept`. To keep the server ready to accept new requests, once the server receive a request, it will fork a child process to serve the request, so that the parent process can continue to run `accept` and wait for the next request. When the child exits, the parant handles the signal `SIGCHILD` to prevent zombie processes.

As the child process is forked, it starts to parse the request, sends console outputs and responses, and exits when all the work is finished for that request. It mainly runs two functions, `dostuff` and `serveReqiest`.

The dumping functionality in part A is implemented in `dostuff` function:

```
void dostuff(int sock, RequestHeader* header);
```

It first read the request through socket file descriptor `sock` into a buffer, then print out the content of the buffer, which is a request line followed by several header lines. It also extract the header of the request and store it into `header`. This header is used for part B.

The functionality of sending response in part B is implemented in the `serveRequest` function:

```
void serveRequest(int sock, const RequestHeader header);
```

The function first tries to read the file specified in `header`. If the file does not exist or it cannot be opened, it will send an error message to inform the client. If the file exists, the server will response the client with an HTTP response message with the file included.

Difficulties

The main difficulty of this project lies in the formating of response messages. Based on RFC 1945, the header can include many header lines. It took us some time to figure out which ones should be included in the header and which should not. Also the response could not be read by the browser if any syntax error in that message, which made debugging hard.

Sample Output

Conclusion

This project implementation meets all the requirements in part A and part B successfully. The web server gives us a better understanding on the client-server model and socket programming with the libraries. It also illustrates an example for HTTP request and response. Though the functionality of the server is simple, it does cover the important concepts, like port, socket, HTTP messages, and etc.