Project 1 : "Simple" - A simple multi-threaded HTTP/1.0 Web Server

14–740 Fundamental of Telecommunication Systems

Released: February 17, 2016, 6:00pm EDT
Due: March 1, 2016, 6:00pm EDT

Contents

1 Objectives

In this project you will be using the Socket APIs to write a simple web server which conforms to a subset of the HyperText Transfer Protocol 1.0 (HTTP 1.0) as defined in RFC 1945[11]. The purpose of this project is to give you experience in developing multi-threaded network applications.

Why are we making you create your own web server when there are so many other web servers that exist for all kinds of purposes (Apache, IIS, lighthttpd, Mongrel,...)? HTTP is one of the most prevalent application level protocol today. Understanding HTTP and web server internals will give you an idea of how much of today's Internet works. The socket APIs that we will be using are the base of all the network programming. This project will give you a good idea how these APIs work and how they make things easy for the programmers by masking away all the details below the transport layer.

By the end of this project you will have a functional web server which can be tested using your favorite browser application :-)

2 Logistics

- You will work in teams of 2, so the very first step in doing the project is to "Go find a partner!".
- You can write the web server in C or Java. We prefer C because that is what most high performance network applications use. However, there is no penalty or bonus involved with the selection of the programming language.
- We will provide you with (heavily commented) support code, in both C and Java, to make sure you understand the socket APIs before you start implementing the web server.
- The grading criteria is listed at the end of this handout.
- Submissions will be made via Blackboard.

3 The "Simple" server

Your server will implement HEAD and GET methods of HTTP 1.0 (static requests only). These should comply with the specification in the RFC.

- GET requests a specified resource; it should not have any other significance other than retrieval
- HEAD asks for an identical response as GET, without the actual body – no bytes from the requested resource (i.e just send the response header)
- For all other commands your server must return "501 Method Unimplemented" error response.

3.1 Part I : Implementing a sequential server

The first step is implementing a basic sequential web server that handles GET / HEAD request one at a time. Once started, your server should listen for connections on the port number that is passed via the command line. When the server receives a connection request, it should accept the connection, read the request, parse it and respond to the request. You will need to read the file that is requested and send its contents back to the client as part of the response. After sending the response completely, the server must close the connection (as defined in the HTTP/1.0 RFC) and go back to listening for new connections.

- **Processing HTTP Requests**
  A typical HTTP/1.0 request is of the form

```
GET /index.html HTTP/1.0[CRLF]
Host: www.example.com[CRLF]
Connection: close[CRLF]
[CRLF]
```

where CRLF is carriage return "\r" followed by the line feed "\n" character.

When your server receives such a request, it must read the file "index.html" and create an HTTP 1.0 response to be sent to the client. A typical HTTP 1.0 response is of the form

```
HTTP/1.0 200 OK[CRLF]
Server: Simple/1.0[CRLF]
Content-Type: text/html[CRLF]
[CRLF]
This is an example response
```

The "Server:" header in the response **must** be `Simple/1.0` in your implementation of the server.

3.2 Part II : Building a concurrent web server with threads

Actual servers do not respond to requests in a sequential manner. They implement concurrency using either threads or events. We will be using threads for implementing concurrency in Simple. A common method for this is to spawn a new thread to handle each request that comes in. In this architecture, the main thread simply accepts connections and spawns worker threads to read requests and respond to the clients. The threads terminate when they are done. You can find more information about pthreads (for C)[2] and Java Threads[9] in the References. You can refer to Ch. 12 of the CS:APP[10] book for a quick introduction to thread programming.

You will be required to use threading in Project 2. Use this Project as an opportunity to get used to implementing multi-threaded programs.

4 Support Code

The support code is available in both C and Java. The Java version can be directly imported into Eclipse[12] as a project. Download the tar-file from blackboard. The support contains the following (both in C and Java) :

- A simple TCP echo server implementation
  This should give you a very good idea of how to create server to accept client connections. You can use this as the base for your implementation of Simple.
- A simple TCP client
  This read an input line from the user, sends it to the server then prints whatever it gets from the server. You can modify this and use it to test your implementation of Simple.
- A utility to get MIME types of the file
  This will give you the MIME type of the given file. You will need this to populate the Content-Type header in the response.
- `log.h`
  The C implementation contains a logging infrastructure that you can use. Please go through the comments in `log.h` and the use of various log function in other files to understand how to use it.

Makefiles are provided for both the C and Java codes. You can use them to compile the code and also use them as reference for your own Makefiles.

The support code also contains a reference implementation of Simple in Python. The code reads very well and it should give you an idea of what you need to do in your C/Java implementations. You can run this version of Simple and check its responses to various requests. Your implementation of Simple should do the same. You can run the reference implementation as :

```
./simple.py 8080 $PWD/www
```

We have included a sample `www` directory which contains an `index.html` page. You can use this directory when you test Simple.

5 Plan of Attack

- Read through this handout completely and understand what you are supposed to do.
- Read the annotated RFC and related parts of Lecture 3 to have a clear understanding of what needs to be implemented.
- Read and understand the sample socket code provided. Make sure you understand what is happening with the socket calls.
- Implementation :
  - The server can be implemented as 3 separate modules
    - A module that accepts new connections
    - A HTTP request parser and
    - a module to handle the response (this needs to be done in a separate thread for a concurrent server)
  - The first module will be very similar to the echo server given in the handout code.
  - You can implement and test the HTTP parser module separately.
  - Integrate the HTTP parsing with the echo server code and then add code to do response handling. You should have a sequential server running at this stage. Make sure you test this thoroughly before you move on to implementing concurrency.
  - Add threading. Use threads to handle creation and sending of responses to the clients. Test again.
  - Submit to Blackboard
- **NOTE:**
  - Use of the RIO wrappers from CS:APP is **NOT** allowed.
  - Use of ready-made jars and libraries for parsing HTTP requests and creating HTTP responses is **NOT** allowed.
  - We will read your code and your score will be reduced drastically if we find any of the above. Please contact the instructors if you plan to use any code other than the one provided in the handout.

6 Completing the Project

The expected server implementation of Simple must take 2 arguments and should be executed as :

```
./simple <port_number> <www_path>
```

OR

```
java Simple <port_number> <www_path>
```

where the `port_number` is the port on which the server listens for client connections and `www_path` is the path that server would be serving to the client (path where all your html files are).

Please create a tarball of your source code (all .C, .h and Makefiles or .java files) with the name `<andrew_id1>_<andrew_id2>.tar.gz` and upload it to Blackboard before **11:59pm** on the due date.

7 Help

- Debugging
  You will need to use `gdb`[1][7] and Eclipse's[12] inbuilt debugger to debug your code. While we believe that being a graduate student, you should be able to debug your own code, please contact your TAs if you don't make progress for a long time and need help debugging. Debugging multi-threaded processes can be a pain and your TAs can help you find better way to debug.
- Testing
  - Wireshark[4]
    Please employ the Wireshark skills you acquired in Homework 4 to test and debug the packets that are sent and received by the server and the client. This will be very helpful to figure out what your code is actually sending over the network.
  - `telnet`[3]
    Telnet can be a great tool to send random data to the server to test it for normal cases as well as testing it for robustness.
- Please check the References section for links to more help documents[5].

8 Grading

- Sequential HTTP 1.0 server handling GET and HEAD as expected **(70 points)**
- Adding concurrency using threads **(20 points)**
- Robustness **(5 points)**
  - How well does your code handle improper requests
  - Incomplete requests
  - Client disconnections
- Code style and documentation **(5 points)**
- **BONUS**
  Please note that BONUS points will be awarded only if you already have a concurrent server running. Do not try to complete the bonus parts after just implementing a sequential server - no bonus points will be awarded in such situations.
  This part of the Project is open ended. We want you to add features to your server. The following list can give you an idea of what we expect :
  - CGI Implementation
    Add support of dynamic requests using the Common Gateway Interface[6][13] protocol. Hint : Use `fork()` and `exec()`.
  - Fighting DOS attacks
    Since you will be creating one thread per connection in your server, you might end up in a situation where rogue clients can just create connections with your server and hog it - resulting in Denial of Service (DOS) for actual clients. Can you find some way to detect this and close such connections? Hint : Timeout functionality of `select()`
  - Benchmarking your server
    How well does your server perform. Benchmarking it using standard tools like `httperf`[8] and write a report about your findings.
  - Testing suite for Simple
    You can also write a testing suite of traces, a python client program etc to test your implementation of Simple.

These are some of the things that you can implement for the bonus points. Please contact your TAs or start discussions on Piazza if you need any help with these or have other ideas.

References

[1] Gdb: The gnu project debugger [online]. Available from: http://www.gnu.org/software/gdb/.

[2] pthreads man page [online]. Available from: http://linux.die.net/man/7/pthreads.

[3] telnet man page [online]. Available from: http://linux.die.net/man/1/telnet.

[4] Wireshark homepage [online]. Available from: http://www.wireshark.org/.

[5] Brian "Beej Jorgensen" Hall. Beej's guide to network programming [online]. Available from: http://beej.us/guide/bgnet/output/html/singlepage/bgnet.html.

[6] D. Robinson, K. Coar. The common gateway interface (cgi) version 1.1 [online]. Available from: http://www.ietf.org/rfc/rfc3875.

[7] FSF. Debugging with gdb: the gnu source-level debugger [online]. Available from: http://www.sourceware.org/gdb/current/onlinedocs/gdb.html.

[8] HP Labs. httperf homepage [online]. Available from: http://www.hpl.hp.com/research/linux/httperf/.

[9] Oracle. Java doc for threads [online]. Available from: http://docs.oracle.com/javase/7/docs/api/java/lang/Thread.html.

[10] Randal E. Bryant and David R. O'Hallaron. Computer systems: A programmer's perspective [online]. Available from: http://csapp.cs.cmu.edu/.

[11] T. Berners-Lee et. al. Hypertext transfer protocol – http/1.0 [online]. Available from: http://www.w3.org/Protocols/rfc1945/rfc1945.

[12] The Eclipse Foundation. Eclipse homepage [online]. Available from: http://www.eclipse.org/.

[13] Wikipedia. Cgi - wikipedia page [online]. Available from: http://en.wikipedia.org/wiki/Common_Gateway_Interface.