Due Date: Wednesday, Mar 5, 11:59pm (Late days may be used.)

This project can be done in groups of 2 students.

1 Introduction

This assignment introduces you to the principles of internetwork communication using the HTTP and TCP protocols, which form two of the most widely used protocols in today's Internet. In addition, the assignment will give you some insights into how to construct web services that are based on the popular REST [3, 4] architectural style, and show an example of how to implement a multi-threaded server.

2 Functionality

In this assignment, you will implement a basic HTTP web service that publishes a Linux system's status as reported by the kernel via the `/proc` file system. The web service must provide this information in JSON [1] format. The web service should implement persistent connections as per the HTTP/1.1 [2] protocol.

The web service shall respond to requests for at least the following resources:

Service URL	Example JSON Output	Based On
`/loadavg`	`{"total_threads": "174", "loadavg": ["0.00", "0.00", "0.03"], "running_threads": "1"}`	`/proc/loadavg`
`/meminfo`	`{"SwapTotal": "5799928", "SwapFree": "5793240", "MemFree": "2434304", ... }`	`/proc/meminfo`

In addition, you must support query arguments as per RFC 2616, Section 3.2.2. You must support a 'callback' field. If given, you must return proper syntax for a JavaScript function call in which the value of the field appears as function name and the JSON object appears as argument. You must ignore all additional field=value pairs. For example, a request to `/loadavg?callback=jsonp1258749550540&_=1258749554624` would return `jsonp1258749550540({"total_threads": "174", "loadavg": ["0.00", "0.00", "0.03"], "running_threads": "1"})`. (The second field `_` is ignored.)

You can see a demo at http://cs3214.cs.vt.edu:9011/loadavg and http://cs3214.cs.vt.edu:9011/meminfo.

You should return appropriate error codes for requests to URLs you do not support.

2.1 Multiple Client Support

Your implementation should support multiple clients simultaneously. It must be able to accept new clients and process HTTP requests even while HTTP transactions with already accepted clients are still in progress. You should use a single-process, multiple-threads approach. It is up to you whether you spawn new threads for every client, or use a thread pool. You may modify or reuse your thread pool implementation from the accompanying exercise, but you may not share your implementation with your project 4 partner until after you submitted the exercise.

To test that your implementation supports multiple clients correctly, we will connect to your server, then delay the sending of the HTTP request. While your server has accepted one client and is waiting for the first HTTP request by that client, it must be ready to accept and serve additional clients. Your server may impose a reasonable limit on the number of clients it simultaneously serves in this way.

2.2 Widgets

Widgets are HTML elements that, when inserted into HTML documents, function as placeholders for enhanced information or functionality. This functionality is usually provided by supporting JavaScript code that interprets the widgets and their parameters. I have designed two small widgets that can interact with the web service you'll create. These widgets graphically display the load average and memory usage of the machine on which your service runs.

You can find a demo of the widgets at
http://courses.cs.vt.edu/ cs3214/spring2010/sysstatwebservice/widget/. Please consult the HTML code of this file for how to include the widgets into your own page for your testing and demonstration, as discussed in Section 4.3.

2.3 Relay Server

The use of network-address translation (NAT) that hides individual machines or even entire networks behind firewalls has grown significantly in the past years. This trend is motivated both by security concerns and by the increasing shortage of routable IPv4 addresses. Understanding the implications of NAT is a crucial skill for application developers for at least the foreseeable future.

Many, if not most, NAT setups allow connections to be initiated only from the inside of the firewall to the outside. The NAT device is typically the default gateway for the hosts behind the firewall, thus allowing it to monitor TCP connection requests to outside servers that pass through the gateway. The NAT device can then establish and keep up-to-date the necessary data structures to translate the addresses for this connection. Correct translation in the other direction is more difficult: if a connection request arrives

on the public-facing interface of the NAT device, then the NAT device would need to know to which server on the inside to forward the request. This information needs to be provided by an administrator of the NAT device, making this approach unsuitable when administrative access is not granted.

A commonly used technique to circumvent this restriction is the use of relay servers. In this technique, a server located behind a firewall creates a TCP connection to a relay server, which is running on a machine that has a routable IP address. Clients then connect to the relay server, which forwards requests to the actual server and relays responses from the server to the clients.

You should implement the ability to provide an HTTP/1.1 service through a relay server. When run in relay server mode, your web service should initiate a TCP connection to the relay server, and then send a single line terminated by `\r\n` with a unique identifier (such as your SLO login). Subsequently, it should respond to HTTP/1.1 requests on that connection. The relay server will create a URL for clients to connect to, which includes that identifier as a prefix.

A relay server is running on cs3214.cs.vt.edu. Port 9050 is accepting connections from web services wishing to make use of the relay service. It accepts connections from HTTP clients on port 9051. Visit http://cs3214.cs.vt.edu:9051/ to see a status page. If a web service connects and send 'prefix' as the first line of data in that connection, requests to `http://cs3214.cs.vt.edu:9051/prefix/loadavg` will be forwarded to the web service as requests for `/loadavg` (after stripping the prefix).

Note that all connections from `rlogin.cs.vt.edu` machines will appear as going to a port on machine `hn1.cs.vt.edu`, which is the DNS name of the public-facing interface of our NAT gateway behind which the machines of the rlogin cluster are located.

Like most NAT gateways, the gateway at rlogin.cs.vt.edu times out inactive connections after a set period. To ensure continuous reachability, your service should periodically reconnect if no request was processed within the time out interval (currently 300 seconds).

2.4 Robustness

Network servers are designed for long running use. As such, they must be programmed in a manner that is robust, even when individual clients send ill-formed requests, crash, delay responses, or violate the HTTP protocol specification in other ways. *No error incurred while handling one client's request should impede your server's ability to accept and handle future clients.*

2.5 Performance

You should optimize for both latency and throughput. We will benchmark your services to measures its performance.

2.6 Minimum Requirements

Your web service must function at least well enough to support continuous use of the memory and cpuload widgets when run in server mode (i.e., accepting clients directly rather than via the relay server intermediary). Your server must support at least 2 clients simultaneously. Robust error handling is also required.

These minimum requirements can be met using one-thread-per-client, HTTP/1.0-only implementation. Support for relay server is not required to meet the minimum requirements.

We will provide a test driver that tests the minimum requirements.

2.7 Choice of Port Numbers and Relay Server Prefixes

Port numbers are shared among all processes on a machine. To reduce the potential for conflicts, use a port number that is 10,000 + last four digits of the student id of a team member.

If a port number is already is use, `bind()` will fail with `EADDRINUSE`. If you weren't using that port number before, someone else might have. Choose a different port number in that case. Otherwise, it may be that the port number is still in use because of your testing. Check that you have killed all processes you may have started while testing. Even after you have killed your processes, binding to a port number may fail for an additional 2 min period if that port number recently accepted clients. This timeout is built into the TCP protocol to avoid mistaking delayed packets sent on old connections for packets that belong to new connections using the same port number.

Choose the SLO login id of one team member as relay server prefix. When running in relay server mode, do not bind the socket to any port before connecting. This 'auto-bind' strategy allows the OS to assign any available port, eliminating the potential for port conflicts.

3 Strategy

Make sure you understand the roles of DNS host names, IP addresses, and port numbers in the context of TCP communication. Study the roles of the necessary socket API calls.

You should exploit a layered design that separates your TCP support code from the HTTP layer. Such a design will be crucial to easily implement the relay server mode while minimizing the changes to the HTTP implementation.

For the TCP layer, make sure you handle short reads correctly, as discussed in lecture and in the book. When assigning port numbers and IP addresses, pay attention to using

proper byte ordering. A recommended convention is to keep `sin_port` and `sin_addr.s_addr` fields always in network order, and to use host order elsewhere in any other storage location your program may store addresses or port numbers.

Since you will be using a multi-threaded design, use thread-safe versions of all functions. Specifically, you should use `getaddrinfo(3)`, `getnameinfo(3)` and `inet_ntop(3)` and you should avoid `gethostbyname(3)`, `getaddrbyname(3)`, or `inet_ntoa(3)`. Using these newer functions has the additional advantage that adding support for IPv6 would be easy (though this is not required for this project).

Familiarize yourselves with the commands `wget(1)` and `curl(1)` and the specific flags that show you headers and protocol versions.

4 Grading

4.1 Coding Style

Your service must be implemented in the C language. You should follow proper coding conventions with respect to documentation, naming, and scoping. You must check the return values of all system calls and library functions.

We will pay particular attention to how you separated the implementation of HTTP from your use of TCP sockets. We may use the helgrind checker to check your server for race conditions. Your code should compile under -Wall without warnings, the use of the `-Werror` flag as part of CFLAGS is strongly recommended.

4.2 Submission

You should submit a .tar.gz file of your project, which must contain a Makefile. Your project should build with 'make clean all' This command must build an executable 'sysstatd' that must accept the following command line arguments:

- `-p port` When given, your web service must run in server mode, accepting HTTP clients and serving HTTP requests on port 'port.' Multiple connection must be supported.
- `-r relayhost:port` When given, your web service should connect to host 'relayhost' on port 'port.'. It should accept both fully-qualified host names and IPv4 addresses in dot notation.

Submit a file called 'README' that lists group members and briefly describes your DESIGN.

Please test that 'make clean' removes all executables and object files. Issue 'make clean' before submitting to keep the size of the tar ball small. Please use the submit.pl script or web page and submit as 'p5'. Only one group member need submit.

4.3 Online Demonstration and Grade Breakdown

The provided test script will test that you meet the minimum requirements, except for the requirement that your web service be integrated into a web page.

To meet this second requirement, you must schedule an online demo with teaching staff. Start your service on a rlogin cluster machine. Email to `cs3214-staff@cs.vt.edu` the URL to a web page that integrates your web service when run in relay server mode. Once you receive a reply that we verified the correct functioning, you may shut down your service.

This project will count for 100 points; meeting the minimum requirements will yield 30 points. The remaining points will be awarded for documentation/coding style, the online demo of your widget integration, and the performance of your service.

4.4 Extra Credit

For 25 points of extra credit, implement your own relay server.

Good Luck!

References

[1] Douglas Crockford. *Introduction to JSON*. http://json.org/.

[2] Roy Fielding, Jim Gettys, Jeff Mogul, H. Frystyk, L. Masinter, P. Leach, and Tim Berners-Lee. Rfc 2616: Hypertext transfer protocol – http/1.1. http://www.w3.org/-Protocols/rfc2616/rfc2616.html.

[3] Roy T. Fielding and Richard N. Taylor. Principled design of the modern web architecture. *ACM Trans. Internet Technol.*, 2(2):115–150, May 2002.

[4] Leonard Richardson and Sam Ruby. *RESTful web services*. O'Reilly, 2007.