Due Date: Friday, Dec 11, 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.

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.

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. This can be implemented in multiple ways: using a multi-process approach (fork()), using a multi-threaded approach (with 1 thread per client, or using a thread pool), or via I/O multiplexing with select() or an equivalent function. The recommended strategy is to create one thread per connection.

To test that your implementation supports multiple clients correctly, we will connect to your server, then delay the sending of the HTTP request. The 'telnet' client program can be used for that. 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.

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/fall2009/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 skills 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 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 request 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.

2.4 Simplifications

Due to the short time allotted for this project, we will make some simplifications. You may assume that HTTP requests are well-formed and that clients and the relay server adheres to the HTTP/1.1 protocol when contacting your service. In relay server mode, you may ignore the fact that the connection will time out after some time, although a more robust implementation would automatically reconnect when that happens.

2.5 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.) You must integrate the web service and widgets into a web page you create.

These minimum requirements can be met using a single-threaded, one-request-at-a-time

HTTP/1.0 implementation. Support for the relay server is not required to meet the minimum requirements.

2.6 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 may 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. If using a multi-threaded design, the helgrind checker must not flag any warnings. Your code should compile under -Wall without warnings, we recommend the use of the `-Werror` flag as part of CFLAGS.

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 Demonstration

To show that your service works, start two instances of your service on a rlogin machine - one in server mode, and one in relay server mode. Email the following information to `cs3214-staff@cs.vt.edu`: (1) the hostname and port number of the machine on which your server runs in server mode (this may be a machine that is part of the rlogin

cluster, but not a machine in the systems lab). (2) 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 the instances of your service.

If you submit your request to demonstrate your service before the submission deadline (expanded by late days if applicable), you will obtain feedback on how well your service works before the final submission deadline. There is no penalty for demonstrations that fail or uncover problems with your service.

If a successful demonstration was completed before 8pm on the date on which you submit (expanded by applicable late days), then the TA will not run your submission. (Though we will still verify that it builds correctly as described in Section 4.2)

If you did not successfully demonstrate your service by 8pm on the date on which you submit, the TA will perform the testing for you and apply **a 10-point penalty**.

We expect to assign between 60 and 80 points for this project. Meeting the minimum requirements will yield 30 points.

4.4 Extra Credit

For 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.