Part 2

Building a CGI-Support Multi-Threaded Web Server

Description

Web servers are frequently implemented as multi-threaded applications. This is done for two reasons. First, multiple threads allow the web server to utilize multiple processors. Second, using threads provides a convenient way to allow each processor to handle multiple requests in parallel.

For this project, you will implement a multi-threaded web server. This project is designed to give you some practice writing client-server socket programs and writing multithreaded programs, as well as familiarizing you with the HTTP protocol.

In the end, you will have built a multi-threaded Web server that is capable of processing multiple simultaneous service requests in parallel. You should be able to demonstrate that your Web server is capable of delivering static and dynamic pages to a Web browser.

Static and Dynamic web pages

Static web pages are very simple. It is written in languages such as HTML, JavaScript, CSS, etc. For static web pages when a server receives a request for a web page, then the server sends the response to the client without doing any additional process. And these web pages are seen through a web browser. In static web pages, pages will remain the same until someone changes it manually.

Dynamic web pages are written in languages such as CGI, AJAX, ASP, ASP.NET, etc. In dynamic web pages, the content of pages is different for different visitors. It takes more time to load than the static web page. Dynamic web pages are used where the information is changed frequently, for example, stock prices, weather information, etc.

The dynamic web page content can vary depending on the number of parameters. It does not just simply send HTML page in response. The web server calls a program which can access a database, perform transaction procedure, etc. If the application program produces HTML output, which is used to construct an HTTP response by the web server. The web server sends the HTTP response thus created, back to the web browser.

CGI

The CGI (common gateway interface) provides an easy way to build dynamic websites using any language of your choice. A CGI program is executed in real-time, so that it can output dynamic information. It can be written in Perl, Php, C/C++ , Java, Python, or other programming languages.

The CGI is a standard way for a Web server to pass a Web user's request to an application program and to receive data back to forward to the user. When the user requests a Web page

(for example, by clicking on a highlighted word or entering a Web site address), the server sends back the requested page. However, when a user fills out a form on a Web page and sends it in, it usually needs to be processed by an application program. The Web server typically passes the form information to a small application program that processes the data and may send back a confirmation message. This method or convention for passing data back and forth between the server and the application is called the common gateway interface (CGI).

A file with the CGI file extension is a Common Gateway Interface Script file. They are text files but since they're written in a programming language like C/C++ or Perl, they can function as executable files under certain conditions. These script files are often seen in a web server's "cgi-bin" directory.

Writing a CGI program

There are two main differences between "regular" programming and CGI programming.

First, all output from your CGI program must be preceded by a MIME-type header. This is HTTP header that tells the client what sort of content it is receiving. Most of the time, this will look like:

Content-type: text/html

Secondly, your output needs to be in HTML, or some other format that a browser will be able to display. Most of the time, this will be HTML, but occasionally you might write a CGI program that outputs a gif image, or other non-HTML content.

Apart from those two things, writing a CGI program will look a lot like any other program that you might write.

Web Server Structure

The basic design of the web server would be the following:

1. Create a server socket on port 8888.
2. Enter an infinite loop:
   (1) Accept the next connection.

   If there are already max connections, kill the oldest thread by closing its client socket. This will cause the worker thread to receive an error or exception the next time it tries to read from or write to the socket. The worker thread should then exit. If the total number of simultaneous open connections gets above maxConnections, your server will start closing the oldest open connections and their associated worker threads will die. The killed worker thread should clean up any shared state and close its end of the socket before dying. Remember that connections can be closed other ways too. For example, the client-side can close the connection. In this case the associated worker thread should detect that the socket was closed, clean up any shared state, and exit.

(2) Create a new thread to handle the new client's connection, passing it the client socket returned by accept.

The main server thread should exit only if it encounters an error or exception.

The worker thread can be in an infinite loop that only exits if it encounters an error or exception or if the socket is closed by the main server thread or by the client. Otherwise, the worker threads continue to handle HTTP requests from the client.

Requirements:

HTTP

you can just implement version 1.0 of HTTP, where separate HTTP requests are sent for each component of the Web page.

Your server must handle GET, POST and HEAD client requests.

It should return appropriate status codes, including 200, 400, 403, and 404. If the server returns an error code to a client, it should also return headers and a message body with a simple error page. For example:

<html><body>Not Found</body></html>

Ports and Addresses

The server will be able to handle multiple simultaneous service requests in parallel. This means that the Web server is multi-threaded. In the main thread, the server listens to a fixed port. When it receives a TCP connection request, it sets up a TCP connection through another port and services the request in a separate thread.

Remember that you are not serving through the standard port 80, so you need to specify the port number within the URL that you give to your browser. For example, if your machine's name is `host.someschool.edu`, your server is listening to port `6789`, and you want to retrieve the file `index.html`, then you would specify the following URL within the browser:

`http://host.someschool.edu:6789/index.html`

Max Connections and Thread pool

Your server can limit the max number of connections. If the total number of simultaneous open connections gets above maxConnections, your server will start closing the oldest open connections and their associated worker threads will die.

In your implementation, you must have a master or main thread that begins by creating a pool of worker threads, the number of which is specified on the command line.

With the pool-of-threads approach, each thread is blocked until there is an http request for it to handle. Therefore, if there are more worker threads than active requests, then some of the threads will be blocked, waiting for new http requests to arrive; if there are more requests than worker threads, then those requests will need to be buffered until there is a ready thread.

CGI

Your Web server is capable of delivering static and dynamic pages to a Web browser.

The CGI provides an easy way to build dynamic websites using any language of your choice.

Your CGI programs can be written in Perl, Php, C/C++, Java, Python, or other programming languages.

Log File

Your server should provide a mechanism for logging everything that happens on your server. A server log is a simple text file which records activity on the server.

Each line in the log file represents one request (hit). If a visitor requests an HTML page which contains two images, three lines will be added to the log (one for the page and two for the images).

Each line may include some or all of the following information:

1. The IP address of the computer making the request (i.e. the visitor)
2. The identity of the computer making the request
3. The login ID of the visitor
4. The date and time of the hit
5. The request method
6. The location and name of the requested file
7. The HTTP status code (e.g. file sent successfully, file not found, etc)
8. The size of the requested file
9. The web page which referred the hit (e.g. a web page containing a hyperlink which the visitor clicked to get here)

It look something like this:

```
213.60.233.243 - - [25/May/2018:00:17:09 +1200] "GET /internet/index.html HTTP/1.1" 200 6792 "http://www.mediacollege.com/video/streaming/http.html" "Mozilla/5.0 (X11; U; Linux i686; es-ES; rv:1.6) Gecko/20040413 Debian/1.6-5"
151.44.15.252 - - [25/May/2018:00:17:20 +1200] "GET /cgi-bin/forum/commentary.pl/noframes/read/209 HTTP/1.1" 200 6863
"http://search.virgilio.it/search/cgi/search.cgi?qs=download+video+illegal+Berg&lr=&dom=s&offset=0&hits=10&switch=0&f=us" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; Hotbar 4.4.7.0)"
151.44.15.252 - - [25/May/2018:00:17:21 +1200] "GET /js/common.js HTTP/1.1" 200 2263 "http://www.mediacollege.com/cgi-bin/forum/commentary.pl/noframes/read/209" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; Hotbar 4.4.7.0)"
151.44.15.252 - - [25/May/2018:00:17:21 +1200] "GET /css/common.css HTTP/1.1" 200 6123 "http://www.mediacollege.com/cgi-bin/forum/commentary.pl/noframes/read/209" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; Hotbar 4.4.7.0)"
151.44.15.252 - - [25/May/2018:00:17:21 +1200] "GET /images/navigation/home1.gif HTTP/1.1" 200 2735 "http://www.mediacollege.com/cgi-bin/forum/commentary.pl/noframes/read/209" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; Hotbar 4.4.7.0)"
151.44.15.252 - - [25/May/2018:00:17:21 +1200] "GET /data/zookeeper/ico-100.gif HTTP/1.1" 200 196 "http://www.mediacollege.com/cgi-bin/forum/commentary.pl/noframes/read/209" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; Hotbar 4.4.7.0)"
151.44.15.252 - - [25/May/2018:00:17:22 +1200] "GET /adsense-alternate.html HTTP/1.1" 200 887 "http://www.mediacollege.com/cgi-bin/forum/commentary.pl/noframes/read/209" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; Hotbar 4.4.7.0)"
151.44.15.252 - - [25/May/2018:00:17:39 +1200] "GET /data/zookeeper/status.html HTTP/1.1" 200 4195 "http://www.mediacollege.com/cgi-bin/forum/commentary.pl/noframes/read/209" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; Hotbar 4.4.7.0)"
```

File Organization

Web server root directory:	/webroot
CGI programs:	/webroot/cgi-bin/
Log files:	/webroot/log/
Static web pages:	/webroot/
Default Page：	/webroot/index.html
404 page：	/webroot/404.html

Test Cases

GET method:

1. Default page: URL: http://hostname or IP address:port/
2. Specific page: URL: http://hostname or IP address:port/page1.html (you can create multiple pages)

POST method:

3. CGI programs and dynamic pages:

(1) **Calculator:** client input two numbers in the form on the web page, submit the form to a specific CGI program. The CGI program calculates the product or sum of these two number, and then send back the result to the client.
Form Action URL: http://hostname or IP address:port/cgi-bin/calculator.pl (assuming it is in PERL)

(2) **Data query:** client input Student ID in the form on the web page, submit the form to a specific CGI program. The CGI program can access MySQL database to query the student's name and class according to the student ID number. If found, send back the queried Student ID, Student Name and Class to the client.
Form Action URL: http://hostname or IP address:port/cgi-bin/query.pl (assuming it is in PERL)

Performance Analysis

You can use Load/Stress Test tools, such as http_load, webbench, etc., to measure the throughput and response time for the web server. For your tests, you should vary the number of worker threads from 1 to roughly 20 (you do not need to test every value in this range). You can use any file for the workload (/index.html is just fine).