CSCI 4061: Intro to Operating Systems
Project 5: Specification Socket Programming and Performance Evaluation

Posted Dec 7 – Due Dec 14, 2:00 pm Groups of 2-3

1 Overview

For Project 5, you will implement the socket (HTTP and file versions) functions we gave to you in the form of object files in Project 4. You will write these functions using UNIX Sockets in C. Also, you will compare the performance of your file-version-program when using different numbers of threads and with or without caching. For this lab, you may disable the prefetcher threads.

2 Socket Programming / Function Implemenation

In this project, you will demonstrate your ability to use network sockets in C. To do this, you will implement the routines that we provided to you in the previous lab. You should use your own code for Project 4 as the starting point. If you could not get this working, you may use the example Project 4 solution.

The functions you will implement are: init, accept connection, get request, return result and return error (for the latter two, we will help you with the simple HTTP headers that you need, given below). You may find the function `fdopen` handy in turning a regular file descriptor into a FILE stream object (this will allow easy high-level standard I/O operations on the socket). You should also set the server socket to allow port reuse as discussed in class.

Your server should be configured to either have caching disabled (0) or with caching (1). This is specified by the caching argument from Project 4.

3 File (STDIN) Versions of Functions

You will need a file version of your web server. This version will be used for all performance testing. This is important because the threads are able to exit when an end-of-file is reached (from the request-list input file), which is great for timer placement for performance evaluation. Here are some things to keep in mind:

- The accept connection function should return 0 (representing standard input) if an end- of-file has not yet been detected. Otherwise, it should return a negative value.
- The get request function should (safely) fetch a line from standard input. If an end-of-file is encountered, it should return -1 and every subsequent call to accept connection should return a negative value.

The return result and return error functions are essentially no-ops. You should do nothing with the data in memory; it should not be used, freed or written anywhere.

4 HTTP Headers and Protocol

When a browser makes a request to your web server, it will send it a formatted message very similar to the text below. It is important to note that for this assignment, you only care about the contents of the very first line. Everything else can safely be ignored.

```
GET /path/to/file.html HTTP/1.1
Host:  your-machine.cselabs.umn.edu
(other headers)
(blank line)
```

When returning a file to the web browser, you must follow the HTTP protocol. Specifically, if everything went OK, you should write back to the socket descriptor:

```
HTTP/1.1 200 OK
Content-Type:  content-type-here
Content-Length:  num-bytes-here
Connection:  Close
(blank line)
File-contents-here
```

Similarly, if something went wrong, you should write back to the socket descriptor:

```
HTTP/1.1 404 Not Found
Content-Type:  text/html
Content-Length:  num-bytes-here
Connection:  Close
(blank line)
Error-message-here
```

5 Performance Evaluation

You will evaluate the performance of your web server for a given request file (using the web server file version). We will provide files that you can use with your server for testing. To time the server, place timer code in your main thread that determines the elapsed wall-clock time taken in processing the entire request file. Note: This is the total server time taken to serve all the requests. You will then compare your server performance: a) without caching, and b) with caching.

You will also compare performance of the server as a function of the number of dispatcher and worker threads. Vary the number of threads from 1, 5, 10 for caching on and caching off (6 total experiments). Report your findings in README files that you submit with the rest of the solution. Why do the results look like this?

6 Simplifying Assumptions

- The maximum number of dispatch threads will be 100
- The maximum number of worker threads will be 100
- The maximum size of the cache will be 100 entries
- The maximum length of the request queue will be 100 requests
- Any HTTP request for a filename containing two consecutive periods or two consecutive slashes (.. or //) must automatically be detected as a bad request by our compiled code for security reasons.

7 Documentation

You must include a README file which describes your program. It needs to contain the following:

1. The purpose of your program
2. How to compile the program

3. How to use the program from the shell (syntax)

4. What exactly your program does

5. Your results from Section 5 and explanation.

You must ALSO include the following information at the top of your README file and main C source file.

```
/* CSci4061 F2011 Assignment 2
* section:  one_digit_number
* section:  one_digit_number
* date:  mm/dd/yy
* name:  full_name1, full_name2 (for partner)
* id:  d_for_first_name, id_for_second_name */
```

8 Grading

20 points: Makefile and sources included is worth 5 points; indentations, readability of code, use of defined constants rather than numbers is worth another 15 points (a perfectly working program will not receive full credit if it is undocumented and very difficult to read).

80 points: Testcases, caching, results, and explanation. Be sure to handle images, html files, missing files and implement the caching strategy properly.

We will use GCC version 4.2 to compile your code.

9 Persistent Connection (10 extra points)

To obtain the extra credit, you will need to implement persistent connection on your web browser. A persistent connection does not close after the server returns the result to the web client. If the web client sends another request later to the same web server, it would use that persistent connection instead of creating a new connection (new TCP handshake, etc..), which has a bigger overhead.

To specify that a connection should be persistent, instead of returning Connection: close as part of the HTTP headers, the server will return Connection: Keep-Alive. The server will keep the connection open (persistent) until there is no activity (no data sent/received) within 5 seconds. A web client will include Connection: Keep-Alive (instead of Connection: close as part of its HTTP header to indicate that it would like to establish a persistent connection.

Your new web server should be able to handle both cases. If the web client specifies in its header file that the connection should be persistent, then the server should not close the connection. Instead, it should keep a timer (for 5 seconds), such that if no data is transferred over that connection, then it should be closed. After the timer expires, the server should close the connection. Please indicate CLEARLY if you are attempting the extra credit.

On Firefox, to specify that the client should use persistent connection, enter about:config in the location address, and filter network.http.keep-alive. Set the value to true to enable persistent connection or false to disable it. For other web clients, Google is your friend.

Disclaimer: This is extra credit and you are responsible for figuring out any quirks that might come up.