CSCI 4061: Introduction to Operating Systems
Assignment 5: Specification Socket Programming

Posted Dec 3 – Due Dec 12, midnight
Groups of 2

1 Overview

For Project 5, you will implement the socket functions for an HTTP web server. You will write these functions using UNIX Sockets in C. We will provide code that uses your functions to connect to web pages.

2 Socket Programming / Function Implemenation

We will provide code to connect to your server in `server.c` (you may modify this file but it is not necessary) and you will use this given file to use some functions you will write in `util.c`. Function signatures are given in `util.h` – this file does not need modification, but contains comments about the functions you will implement.

The functions you will implement are:

`void init(int port)` – run ONCE in your main thread

– Each of the next two functions will be used in order, in a loop, in the dispatch threads (see Section 3 for more explanation of dispatch threads):

`int accept_connection(void)`

`int get_request(int fd, char *filename)`

– These functions will be used by the worker threads after handling the request ((see Section 3 for more explanation of worker threads) `int return_result(int fd, char *content_type, char *buf, int numbytes)` (or `int return_error(int fd, char *buf)`)

(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.

3 Compiling and Using server.c

The file we provide, `server.c`, implements a server which uses dispatch threads (each of which receives a request for a file and stores it in a queue) and worker threads (each of which actually finds and delivers files for a request). Compile the programs with

make -f makefile.linux (if on a Linux machine)

make -f makefile.solaris (if on a Solaris machine)

To run the program after compilation, type

./web_server port path num_dispatch num_workers qlen

where `port` is the port number (you may only use ports 1025 - 65535 by default) `path` path is the path to your web root location where all the files will be served from, `num_dispatch` is the number of dispatch threads listening for connections, `num_workers` is the number of worker threads servicing web pages, and `qlen` is the fixed, bounded length of the request queue.

More explicitly, the purpose of the dispatcher threads is to repeatedly accept an incoming connection, read the request from the connection, and place the request in a queue for a worker thread to pick up and serve, in a loop.

We will assume that there will only be one request per incoming connection. The purpose of the worker threads is to monitor the request queue, pick up requests from it and to serve those requests back to the client.

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 Simplifying Assumptions

- The maximum number of dispatch threads will be 100
- The maximum number of worker threads will be 100
- 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.

6 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

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

```
/* CSci4061 Fall 2012 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 */
```

7 Grading

1. 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).
2. 80 points: Testcases. Be sure to handle images, html files, and missing files.
3. We will use GCC version 4.2 to compile your code.

8 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.