CSCI4730/6730 – Operating Systems

Project #2

Due date: 11:59pm, 03/03/2017

Description

In this project, you will design and implement two versions of multi-threaded web server. The code of a single-threaded web server[1] is available in ELC. You will convert it into multi-threded architectures.
Note that you **do not** write or modify any code that deals with sockets or the HTTP protocol. Your job is to handle with the multi-threading.

Part 1: Multi-threaded server- 25%

The main problem of a single-threaded web server is scalability. It cannot scale up to large numbers of clients.
To address the problem, you will convert the web server into the multi-threaded model. The main thread listens for client connections. When a request arrives, it creates a new thread and immediately returns to listening. The new thread processes the request, sends the response, and terminates.

- You will modify "webserver_part1.c" to build a multi-threaded server.
- You can use htmlget[2] (html client) to test of your program.
    - ./htmlget "host ip" "port"
    - ex) ./htmlget 127.0.0.1 4001
- You can also use htmlget_multi (multi-threaded html client) to test the performance of your program
    - ./htmlget_mu.ti "host_ip" "port" "# of threads"
    - ex) ./htmlget_multi 127.0.0.1 4001 10

Part 2: Thread Pool – 75%

Multi-threaded model will be much better to handle multiple clients simultaneously, but this design still has limitations. Thread creation and termination are expensive operations that are repeated for every request. To remove the unnecessary overhead, you will design and implement thread-pool model.
Your server will maintain a thread pool. When the server launches, it creates [N] child threads. Only the main thread listens for client connections, and each child

[1] It is slightly modified from the code in http://www.jbox.dk/sanos/webserver.htm
[2] http://coding.debuntu.org/c-linux-socket-programming-tcp-simple-http-client

thread initially waits for the signal from the main thread. When the request arrives, the main thread accepts a new connection and places the connection into a data structure such as a linked list. Any available child thread pulls a connection from the data structure (e.g., linked list), receives a request from the connection, produces the response and sends it to the client. You will need to carefully use synchronization methods to avoid concurrency problems such as race conditions or dead locks.

- You will modify "webserver_part2.c" file to build a thread-pool model.
- Number of thread(N) is given by the user. Your program will accept 2 command line arguments, "port number" and "number of threads".
    - ./webserver_part2 4000 100 // port 4000, create 100 threads
- Partial credit will be given to correct but not efficient solutions (e.g., busy-waiting).

Submission

Submit a tarball file using the following command
%tar czvf proj2.tar.gz README.pdf Makefile webserver_part1.c webserver_part2.c

1. README file with:
    a. Your name
    b. List what you have done and how did you test them. So that you can be sure to receive credit for the parts you've done.
    c. Explain your design of shared data structure and synchronization (and show that your solution does not have any concurrency problems).
2. All source files needed to compile, run and test your code
    a. Makefile
    b. All source files
    c. Do not submit object or executable files
3. Your code should be compiled in cf0-cf11 machine (cf0.cs.uga.edu – cf11.cs.uga.edu)
4. Submit a tarball through ELC.