Final Project
ECGR 6090/8090 Real Time Operating Systems
Spring 2014

A real-time priority based pre-threaded web server

The goal of the project is to develop a web server that interacts with web clients (web browsers) through a pool of worker threads using the producer consumer model. The server consists of a main thread and a set of worker threads with the main thread running at a higher priority than the worker threads. The main thread repeatedly accepts connection requests from clients and places the resulting connected descriptors in a bounded buffer. Each worker thread repeatedly removes a descriptor from the buffer, services the client, and then waits for the next descriptor. All the threads are scheduled with the FIFO policy. The webserver is capable of serving both static and dynamic content.

For this project, you can use the following C source code available on Moodle.

1. Tiny Webserver: Tiny is a simple web server than can serve both static and dynamic content. Code for adder as an example of dynamic content (CGI) is also provided. The web server and the adder code are from the book CS:APP, 2nd Edition, by Bryant and O'Hallaron.
2. Multi-threaded producer-consumer: An array is used to implement a bounded buffer. Producer and consumer threads write and read data from this buffer synchronizing with locks and condition variables using the Pthread API
3. Real time scheduling and priorities in Linux: Demonstrates how to assign real time priority and scheduling policy to threads on Linux. Note that this code needs root permission to run (sudo in Ubuntu).
4. Delay: Spins in a loop for delay seconds. Useful to emulate the execution of long-running threads.

The project has the following milestones

1. Ensure that the Tiny web server works. The web server listens to connection requests at any unused port. The web client connects to the browser on the loop back IP address. As an example- the web server listens at port 1024, the static file to be served is an image file in the jpg format (image.jpg), and the browser is pointed to the address 127.0.0.1:1024/image.jpg. The browser should display this image. You can try to display dynamic content as well. For the adder example provided, the address on the browser will be 127.0.0.1:1204/cgi-bin/adder?100&200. Here the adder executable is placed in the directory cgi-bin, and 100,200 are inputs to the adder.
2. Write a threaded version of the Tiny webserver. On every connection request, the server spawns a new thread to process the request. Note that each connection requires a separate connection descriptor (connfd) to prevent races. Use malloc to dynamically allocate memory for the individual connection descriptors.
3. Write a pre-threaded version of the Tiny webserver. Using a thread for each connection request is expensive due to the overheads involved in thread creation and destruction. A better approach is to have a pre-threaded server where a fixed number of worker threads process the connection requests. The main thread (manager) that accepts connection requests puts the connection descriptors in a bounded buffer and the individual worker threads remove requests from the bounded buffer before processing.

4. Extend the pre-threaded Tiny webserver so that the manager thread has higher priority than the worker threads.

Demo: Set the number of worker threads to the one less than the number of cores on your system. Set the buffer size to twice the number of threads. Insert a delay of 30 seconds for the worker threads. Now send a series of different client requests using different tabs of your browser. Since the manager is running at a higher priority and the worker threads are slow to process the requests (due to the delay), the manager will preempt the workers whenever there is a request from the client, resulting in manager finding the connection descriptor buffer full. Print appropriate messages from the manager and worker threads to demonstrate that the manager thread is indeed running at a higher priority.

Please organize the different parts of your code in separate C files and use the gnu make utility in compiling your code. If your final milestone does not work, be prepared to demonstrate milestone 2 and 3 for partial credit.

You can bring your laptop to the oral exam to demo your code. For those working on a home desktop, send me your tarred directory through Dropbox ahead of the oral exam.

To understand more about how the tiny web server works, read Chapter 11, Network Programming from CS:APP, 2nd Edition, Bryant and O'Hallaron.