Networked Systems Architecture 3: Laboratory Task 4

Dr. Colin Perkins

20 February 2008

The final laboratory task is to extend your web server to accept multiple connections in parallel. This should improve its scalability and response time on multicore systems, and on systems with relatively slow disks compared to the speed of their network connection. There are two parts to this task: introducing concurrency to your web server by starting a new thread for each connection, and then extending the server with a thread pool, to hide thread creation overheads. Both parts should be implemented using the pthreads API functions.

A Concurrent Web Server: Extend your server so that it starts a new thread to process each network connection accepted on a socket. That is, when the `accept()` function completes, start a new thread to process the newly accepted connection. The new thread should accept the file descriptor for the new connection as a parameter in the `pthread_create()` call. Once created, the thread will process HTTP requests until the connection is closed by the client, then it will close the connection socket, and exit. The main thread should remain open and accepting new connections.

Test your system to demonstrate that it can successfully handle multiple connections in parallel. A standard web browser such as Firefox should open multiple connections to a site, or you could modify the web downloader you wrote in the first laboratory session to use threads and download several pages at once.

A Concurrent Web Server with Thread Pool: Starting a new thread for each connection can be inefficient, since threads take some time to start. A more scalable approach creates a pool of worker threads before accepting any connections, and passes each new connection to the next free thread in the pool.

Such a system comprises a single controller thread, with a pool of worker threads. The controller maintains a list of workers, along with a state variable for each indicating if it is busy of idle. Newly accepted connections are passed to the first idle worker in the list, with new connections not being accepted until there is an idle worker thread.

Implement such a system using pthreads to create the worker threads. Take care to provide appropriate locking when manipulating condition variables shared between controller and workers. Ensure your workers block on the condition variable while waiting for new work, rather than continually polling. Test your system, to demonstrate that it works correctly.

A worked solution to this programming task will be provided in lecture 18.