Putul Siddharth
17BCE1100

Digital Assg.
Prof. KUMAR R.

TOPIC – "Implementation of multi-threaded Web Server in C-language".

HTTP Background

Before describing what we will be implementing in this project, we will provide a brief overview of how a webserver works & HTTP Protocol.

Web browsers and web servers interact using a text-based protocol called HTTP. A web browser opens an internet connection to a web server & request some content with HTTP. The web server respond with the requested content and closes the connection. The browser reads the content & display it on scree

An HTTP request consist of request line, followed by zero or more request headers and finally an empty text line.

An HTTP response is similar; it consists of a response line, zero or more response headers, an empty text line and finally the interesting part, the response body.

Overview: New functionality

In this project, we will be adding functionality to both web server code and web client code.

We will be adding three key pieces of functionality to the basic web server. First, we make the webserver multi-threaded, with the appropriate synchronization. Second, we will implement different scheduling policies so that requests are serviced in different orders. Third we will add statistics to measure how the web server is performing. We will also be modifying how the webserver is invoked so that it can handle new input parameters.

We will also be adding functionality to the web client for testing. We should think about how this new functionality will help us test that the web server is implemented correctly. We will modify the web client so that it is also multi-threaded and can initiate requests to server in different, well controlled groups.

Part 1 : Multi-threaded Server.

The basic web server that we provided has a single thread of control. single-threaded web servers suffer from a fundamental Performance problem in that only a single HTTP request can be services at a time. Thus every other client that is accessing this web server must wait until the current HTTP request has finished; this is especially a problem if current HTTP request is long-running CGI program Or is resident only on disk. Thus, the most important extension that we will be adding is to make the basic web server multi-threaded

The simplest approach to building a multi-threaded server is to spawn a new-thread for every new http request. The OS will then schedule these threads according to its own policy. The advantage of creating these threads is that now short request will not need to wait for a long request to complete; further when one thread is blocked, the other thread can continue to handle over requests.

Part 2 : Scheduling Policies.

In this Project, we will implement a number of different scheduling policies. The scheduling policy is determined by a command line argument when web server is started & are as follows:

- Any Concurrent policy (ANY): When a worker thread wakes, it can handle any request in the buffer. The only requirement is that all threads are handling request concurrently.
- First-in-First-out (FIFO): When a worker thread wakes, it handles the first request in the buffer. Note that the http request will not necessarily finish in FIFO order since multiple threads are running concurrently.
- Highest Priority to Static Content (HPSC): when a worker thread wakes, it handles the first request that is static content; if there are no request for static content, it handles the first request for dynamic content.

Part 3 : Usage Statistics

We will need to modify our web server to collect a variety of statistics. For each request, we will record the following count or times :-

- stat-req-arrival-count : The number of request that arrived before this request arrived.
- stat-req-arrival-time : The arrival time of this request, as first seen by master thread. This time should be relative to the start time of web server.
- stat-req-dispatch-count : The number of requests that were dispatched before this request was dispatched.
- stat-req-dispatch-time : The time this request was dispatched. This time should be relative to the start time of web server.
- stat-req-complete-count : For static content, the number of request that completed before this request completed.
- stat-req-complete-time : For static content, the time at which the read of the file is complete & worker thread begin writing

the response on the socket.

Part 4: Multi-threaded client.

We provide a basic single-threaded client that sends a single HTTP request to the server and prints out the result. This basic client prints out the entity headers with the statistics that we added, so that we can verify the server is Ordering request as expected. While this basic client can help you with some testing, it doesn't stress the server enough with multiple simultaneous request to ensure that the server is correctly scheduling or synchronizing threads. Therefore, we need to modify the Web Server to send more requests with multiple threads.