Assignment 4

In assignment 2, you have built your own thread library. Now, you have got a good familiarity with the thread concepts. This assignment demonstrates the application of threads.

Basic requirements:

1. You need to implement the web server in C.
2. For assignment, we will use port 8888.
3. Implement HTTP 1.1 protocol, HTTP 1.0 protocol.
4. Max_connections_threshold should be configurable.
5. **You have to use the thread library built in assignment 2.**

Pointers:

1. Socket programming : http://beej.us/guide/bgnet/
   http://beej.us/guide/bgnet/pdf/bgnet_usl_c_1.pdf
2. HTTP : https://www.jmarshall.com/easy/http/

Assignment Details :

In this assignment, we will be implementing a multi-threading web server, one thread per client connection. This assignment tends to familiarise you all with *client-server socket programming, writing a multi-threaded server, signal handling and HTTP protocol.*
The server should be able to handle requests from multiple-clients simultaneously (Figure 1).

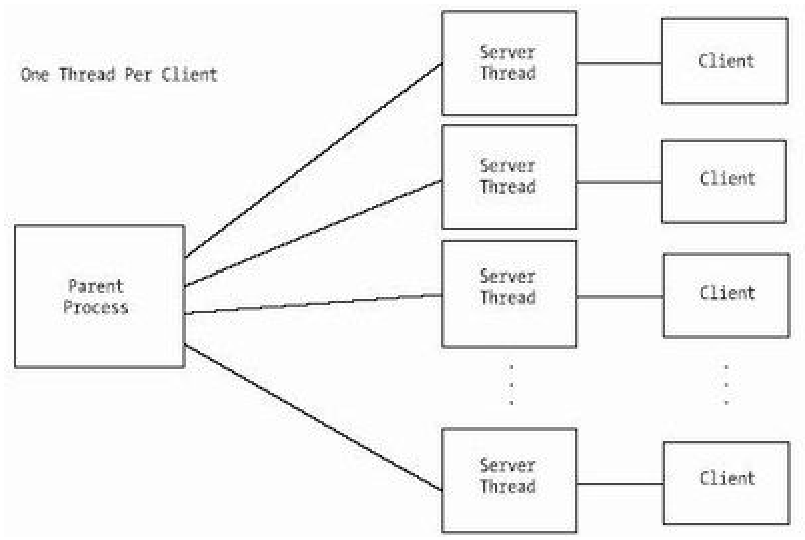
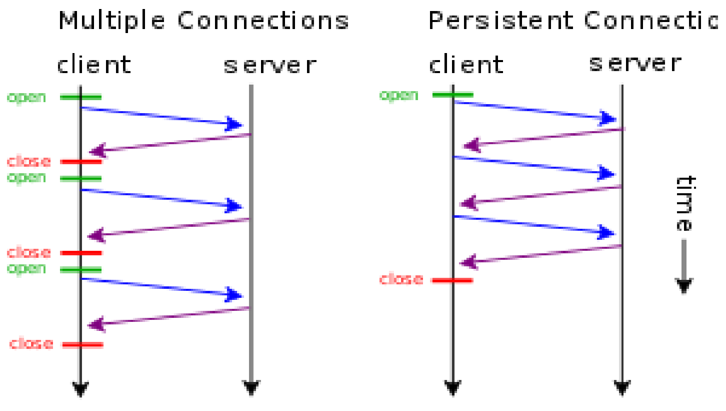


Figure 1: Multi-threaded server

HTTP is the foundation of data communication for the World Wide Web. The HTTP protocol defines a specific format for the contents of a message from a client to request information from a web server. With HTTP, the client sends a message requesting data, which may be a static page or a page that the server will dynamically generate. The server then sends data back, usually in the form of an HTML, XHTML or similar document.

For the assignment, you need to *implement HTTP 1.0 as well as HTTP 1.1* protocol, In HTTP 1.0 the server maintains an open connection to the client even after the response is sent. In short, in HTTP 1.1, the client-server need not to repeat the TCP connection protocol for each request and hence, increase the communication faster. While, in Http 1.0, the client closes the connection as soon as the response is sent to the client as shown below.



HTTP 1.0 HTTP 1.1
Figure 2 : HTTP 1.0 vs. HTTP 1.1

The server should have an algorithm to handle too many simultaneous connections.
Approach :

1. Set Max_connections_threshold value.
2. If (number_open_connection + requested one more connection) > Max_connections_threshold
   close the connection in FIFO order.

The server must handle GET,POST and HEAD client requests, you may leave POST or any other requests. The server should return the appropriate status code including 200, 400, 403, 404. The error code should be wrapped in a header and a message body with a simple error page.
e.g. “<html> <body>Error code = “error_code”</body></html>”
Header parameter of HTTP requests: content-length, content-type, date.

Details :

Next, we explain each component of the assignment in detail.

Web-server :

Design:

- Create a listen socket on port 8888
- Go to an infinite loop
  - Accept the new connection
  - If(number_open_connection > Max_connections_threshold) Kill the connection in FIFO order.
  - Create the new thread to handle new connection amd pass the socket returned by accept in step 1
  - For HTTP 1.1, the worker thread should be an infinite loop that only exists if there is an error condition returned by a system call or the thread receives a

SIGALARM from the main thread and kills itself. Otherwise the worker threads continue to handle HTTP requests from the client.
  - For HTTP 1.0, the worker thread will wait to complete requests from the client and then close the connection.

However, the main server thread should run in an infinite loop, waiting to accept the next client connection. It can exit on receiving appropriate return values from accept, send, recv, read, and write.

Signals, sockets, threads :

The web server will use signals (use SIGALRM) to notify a worker thread that it should die when it reaches Max_connections_threshold limit.
A signal is a software interrupt, that can be synchronous or asynchronous. One process or thread can send (or post) a signal to another one, and when the other one receives the signal it stops doing what it is currently doing and runs a special signal handler function. Processes (and threads) can block some signals, register their own handler functions on some signals, or just use the operating system's signal handler functions (this is the default). For example, when you type CNTL+C in the terminal that is running a program, the running process is sent a SIGKILL signal telling it to die. SIGKILL is an example of a non-blockable signal, meaning that a process cannot choose to ignore a SIGKILL...it must die.

Threads will also need to detect and handle other cases when they should exit, and clean up any global state associated with them, including closing their socket before exiting. One place where this may occur is if the client side disconnects and closes its end of the socket.

1. HTTP 1.0

Use your own library to create a thread each time a client wants to connect to the server. While server threads have a dedicated connection to the client and open to serve the GET, POST and HEAD requests from the client until or unless, the client closes the connection.

2. HTTP 1.1 and multiple simultaneous connections :

Server threads have a dedicated connection to the client and are open to serve the GET, POST and HEAD requests from the client until or unless, the client closes the connection or the Max_connections_threshold limit is reached.

Your main server thread should return back to its accept loop after spawning the server thread so that it can handle a connection from another client. This way your server can simultaneously handle requests from different clients. Test that this works by connecting to your server from different clients simultaneously and sending multiple requests from these clients.

General format of a server request:

Format	Example
Initial line Header1: value1 Header2: value2 Header3: value3 (Optional message body goes here)	HTTP/1.1 200 OK Date: Sun, 10 Jan 2010 18:17:43 GMT Content-Type: text/html Content-Length: 53 <html> <body> <h1>CS 87 Test Page</h1> </body></html>

Note:

- Each header line ends with a "\r\n"
- A blank line (another "\r\n") between the headers and the message body.

WebClients:

You can use multiple programs to connect to your web server and send it HTTP commands:

1. telnet server_IP port_num , then type in a GET command (make sure to enter a blank line after the GET command). For example:
   ```
   $ telnet server_IP port_num
   GET /index.html HTTP/1.0
   ```
   telnet will exit when it detects that your web server has closed its end of the socket (or you can kill it with CNTL^C, or if that doesn't work use kill or pkill: pkill telnet). Use ifconfig to get a machine's IP address

2. Firefox: Enter the url of the desired page specifying your web server using its IP:port_num (e.g. http://server_IP:port_num/index.php)
   You can also just use localhost or the host name on our system:
   localhost:8888/index.php
   tomato:8888/~cfk/

3. wget: wget -v server_IP:port_num/index.html
   wget copies the html file returned by your web server into a file with a matching name (index.html) in the directory from which you call wget.

Output:

1. Test 1: Single client-server communication 20 marks
- A simple hello message with client ID from client to server
- Client requests for index.html from server
2. Test 2: Multiple client-server connection 20 marks
- Simultaneous communication on a different connection
- Try to create more than Max_connection_threshold connection
3. Test 3: Plot the time taken to transfer files from the server using HTTP 1.0 and HTTP 1.1 and draw time vs. file size graph using GNUPlot. 10 marks

4. Now that you have built a multi-threaded web server; extend this to build a multi Chat working on the browser. The server acts as an intermediate between clients and stores chat history as web server logs. This part is open-ended; you can use your creativity to make it a more realistic web chatting application. 35 marks
5. Once the chat server is built, the player wants to play a game. In-game, the server initiates a global integer variable, and other players want to update the value. However, only one player can update the value at a time. You need to use the synchronization method built-in assignment 2. 15 marks

Submission:

1. Zip the project and rename to ENTRYNUMBER_FIRSTNAME_SECONDNAME.zip
2. Follow standard project directory structure for the submission, i.e. /src/ : source files, /lib/ : library.
3. You should use svn to manage the commit versions of your project. Submit history of changes from svn log on every submission
4. Add a readme file with details for executing your code.