CSc 361: Computer Communications and Networks (Fall 2012)

Programming exercise 2: Simple Web Server (SWS)

Due: Nov 28/29 in lab

1 Introduction

In this programming assignment, you will implement a simple web server (SWS) in C++, following a simplified HTTP protocol and using a socket application programming interface (API) with the Transmission Control Protocol (TCP). The purpose of this assignment is to learn the `socket` API, and the client-server application model.

2 Background

2.1 socket API

`socket` is the API to network services in many operating systems. For a server-like application, normally you will need to use the following system calls.

- `socket()`: to create a new socket
- `setsockopt()`: to manipulate options of a socket
- `bind()`: to associate the socket to a local address
- `listen()`: to wait for an incoming connection request
- `accept()`: to accept the incoming connection request with a new socket
- `recv()`: to read from the socket
- `send()`: to write to the socket
- `close()` to close the socket and release the resources allocated
- `select()`: to monitor mutiple file descriptors

You will want to read the manual page (e.g., `man socket`) to better understand these system calls and their input arguments and return values. It is very important to check the return value of these function calls in your program to determine whether the intended operation is successful.

2.2 HTTP Protocol

HTTP is a client-server, request-response-based application-layer protocol for the Web. In this assignment, only a very simplified version of HTTP/1.0 is to be implemented.

2.2.1 Simplified HTTP Request

As stated in the HTTP/1.0 specification: "A request message from a client to a server includes, within the first line of that message, the method to be applied to the resource, the identifier of the resource, and the protocol version in use."

For example, when `wget -d www.csc.uvic.ca`, it shows

Message	Explanation (**required** or ignored by the simple web server)
`GET / HTTP/1.0`	**method, request URI, HTTP version (required)**
`User-Agent: Wget/1.10.2`	Request header (ignored)
`Accept: */*`	Request header (ignored)
`Host: www.csc.uvic.ca`	Request header (ignored)
`Connection: Keep-Alive`	Request header (ignored)
	a blank line indicating the end of request headers (required)

The simple web server will read the request line, which includes **method**, **request URI** and **HTTP version** separated by **string delimiters** (e.g., blank spaces or tabs), but will ignore all request headers, until it reaches the end of request headers indicated by **a blank line**.

Method and **HTTP version** are not case sensitive; however, **URI** is case sensitive.

The simple web server only supports the `GET` method, which obtains an HTTP object (often an HTML file) from the web server.

URI identifies the HTTP object. For example, `GET / HTTP/1.0` tries to obtain the default file `index.html` from the root of the web server document directory. If the web server's root document directory is `/tmp/www`, then `/tmp/www/index.html`, if existent, is retrieved and returned by the web server; otherwise, a `Not Found` error message is returned instead. `GET /icons/new.gif` will let the web server retrieve `/tmp/www/icon/new.gif`.

The simple web server only recognizes `HTTP/1.0` as the valid HTTP version supported.

The simple web server will return a `Bad Request` error message if an invalid request line or an incomplete request message is encountered. The minimal, valid request message contains a valid request line and a blank line indicating the end of the request.

2.2.2 Simplified HTTP Response

As being stated in the HTTP/1.0 specification: "After receiving and interpreting a request message, a server responds in the form of an HTTP response message."

When `wget -d www.csc.uvic.ca`, the web server returns

Message	Explanation (**required** or omitted in the simple web server)
`HTTP/1.1 200 OK`	**HTTP version, status code, reason phrase (required)**
`Date: Fri, 12 Sep ...`	response header (required)
`Server: Apache/2. ...`	response header (omitted)
`X-Powered-By: PHP/4 ...`	response header (omitted)
`Connection: c ...`	response header (omitted)
`Content-Type: text/ ...`	response header (required)
	a blank line indicating the end of response headers (required)
index.html content	**HTTP object(s) returned (required, if any)**

The simple web server will return the response line, which includes **HTTP version**, **status code** and **reason phrase** separated by **blank spaces**. It also returns **Date** and **Content-Type**, but will omit other response headers shown above. However, the simple web server will return **a blank line** indicating the end of response headers and the start of the returned HTTP object, if any.

The response line is not case sensitive, but it is suggested to follow the convention shown above.

The simple web server only returns `HTTP/1.0` as the valid HTTP version supported.

The simple web server only supports the following list of status codes and reason phrases.

Status Code	Reason Phrase	Explanation
200	`OK`	good request with the requested object to be returned
400	`Bad Request`	bad request not understood by the server
404	`Not Found`	good request with no matching request object

If the simple web server can understand the request, and the request object is successfully retrieved, it will return `HTTP/1.0 200 OK` followed by a blank line and the content of the requested object. If the simple web server cannot understand the request, it will return `HTTP/1.0 400 Bad Request` followed by a blank line. If the simple web server can understand the request, but the requested object is not available (e.g., nonexistence), it will return `HTTP/1.0 404 Not Found` followed by a blank line.

The simple web server does not have to support persistent connections.

3 Requirement

3.1 Basic Features

Basic features are required in all implementations in order to get the full marks for this assignment.

3.1.1 Invoking the Server

The syntax to run the simple web server is

```
./sws <port> <directory>
```

This will invoke the simple web server binary `sws` in the current directory, and instruct the simple web server to wait at the TCP `port` for incoming requests and to retrieve requested objects under `directory`. If a wrong syntax is used when invoking the server, the server should print out error messages showing the proper usage and exit gracefully.

When it is invoked successfully, for example, the simple web server will print out the following message if being invoked as `./sws 8080 /tmp/www`

```
sws is running on TCP port 8080 and serving /tmp/www
press 'q' to quit ...
```

The client is not allowed to use `../../` to retrieve objects out of the root document `directory`; such requests will be responded with `HTTP/1.0 400 Bad Request` by the server.

3.1.2 Server Operations

The simple web server should respond to incoming requests at TCP `port` for all network interfaces on the machine running the server.

Once a request is served, the server should print out a log message in the following format.

```
Seq no. YYYY MMM DD HH:MM:SS Client-IP:Client-Port request-line; response-line;
[filename]
```

The **Seq no.** is a simple sequence number which identifies every log line. For example, if the server, on the noon of `Sep 12`, successfully served a request `GET / HTTP/1.0` from the client at port `4096` on host `192.168.1.100`, it should print out

```
1 2009 Sep 12 12:00:00 192.168.1.100:4096 GET / HTTP/1.0; HTTP/1.0 200 OK;
/tmp/www/index.html
```

3.1.3 Terminating the Server

The simple web server should continue serving HTTP requests until the `q` key is pressed in the terminal running the server. In order to respond to console input, you may need to use `select()` system call. When the `q` key is pressed, the server should finish serving all ongoing requests, close all created sockets, and release all allocated resources before exiting gracefully.

After terminating the server, or if the server is aborted, some socket resources may still be used by the system. If you rerun the server on the same port immediately (or if there is already a program on the same port), normally you will get a `bind()` error. To avoid this problem, you should use `setsockopt()` with level `SOL_SOCKET` and option `REUSEADDR` before `bind()` to allow port reuse.

4 Self-Testing

You can test your simple web server with any HTTP/1.0-compliant client, even netcat (or nc). Using `Telnet`, you can connect to your web server running at `port` on `host` by

```
telnet <host> <port>
```

You can then manually type the HTTP request after `Telnet` says:

```
Escape character is ^].
```

And the HTTP response should follow after a correct HTTP request is successfully processed.

To assist your testing, you will be provided a gzipped sample document directory file `www.tar.gz`. A sample request sequence `http.request` is also included.

However, your simple web server may be tested against different document directories and request sequences during code evaluation.

5 Submission

The entire programming assignment should be submitted electronically through Connex during your registered lab session.

Submit only one file

`L0_L1.exercise2.cpp`

where `L0` and `L1` are your team member's login names.