

Assignment #6

Due: March 17, 2017 11:59pm

Overview

For this assignment you will implement a simple webserver that will support a subset of the Hypertext Transfer Protocol (HTTP). This webserver will service multiple client requests by forking child processes to handle the actual request logic. In addition, your server will support functionality for program invocation similar to the common gateway interface approach.

Functionality

Your program, named `httpd`, will take one command-line argument specifying the port on which to listen for connections. The port is an integer value given to the operating system when setting up the network communications. A webserver usually listens for connections on port 80. Since this program will be run on university machines (and at the same time as the solutions written by others in the course), you will need to choose a different port. You may use any number between 1024 and 65535.

Basic Architecture

Your program must repeatedly receive and respond to requests. The protocol for the communication between a client (i.e., a browser) and the server is discussed below. This section outlines the basic architecture of the system.

Before accepting requests, your program must enable network communication. Once communication has been enabled, your program will loop infinitely accepting connections and handling requests. The basic architecture is

```
create_service
while (1)
{
    accept_connection
    handle_request      // done in a child process
}
```

Forking

To handle multiple requests your server will spawn child processes. Each request is to be handled by a single child process. Once the child process is spawned, your server will need to clean up any unneeded file descriptors and it will eventually need to wait for child processes (this should be done via a signal handler).

HTTP

HTTP is a plaintext line-based protocol. This assignment requires that you support only two of the HTTP request types: HEAD and GET.

Request

Each request will be one text line of the form: `TYPE filename HTTP/version`. For example,

```
GET /index.html HTTP/1.1
```

The server will attempt to locate the requested file and, if found, will send a reply with information pertaining to the file and, if a GET request, the contents of the file. A HEAD request will only send information about the file (the contents are not sent). For this assignment, though certainly not in a robust webserver, you can ignore the HTTP version information on a request.

Reply

Each reply begins with a header. The header contains information identifying the type of response, the type of any attached content, and the length of that content. HTTP supports additional fields in the header, but we are only interested in a subset of HTTP 1.0. The header ends with a blank line. Each line of the header must end with `\r\n`. The following is an example of a reply to a valid request (with the carriage-return, line-feed explicitly shown):

```
HTTP/1.0 200 OK\r\n
Content-Type: text/html\r\n
Content-Length: 5686\r\n
\r\n
<----- contents here ----->
```

The contents come directly from the file without any interpretation. For this assignment, your program may always specify `text/html` as the content type, regardless of file.

The response to a HEAD request provides the header only (no contents).

Error Responses

Erroneous requests will be responded to with an appropriate error response. Such a response should have an error type (see below), a content type of `text/html`, and an appropriate snippet of HTML to be presented to the user (this is often just the response type). The following are some standard responses that you may find useful:

```
HTTP/1.0 400 Bad Request
HTTP/1.0 403 Permission Denied
HTTP/1.0 404 Not Found
HTTP/1.0 500 Internal Error
HTTP/1.0 501 Not Implemented
```

You are free to customize the HTML message (which becomes the contents of the response).

cgi-like Support

In addition to the basic behavior of providing files, your server will also provide support for executing programs on the server and providing the output back to the user¹.

Only the programs in the `cgi-like` directory (a subdirectory of the server's working directory) may be executed. These programs may be passed arguments as part of the request. These arguments are provided, in the URL, after a `?` following the program name and the arguments are separated by `&` characters. The arguments themselves are simply strings. If there are no arguments, then there will be no `?`.

As such, the request header for an attempt to run (as an example) the `ls` program might be as follows (to list the two files mentioned in long format).

```
GET /cgi-like/ls?-l&index.html&main.html HTTP/1.1
```

Yes, the `cgi-like` directory name must immediately follow the `/` in order to be considered valid.

You will need to `fork` a process to execute this command. If the `fork` fails, then reply with a 500 error.

If the `exec` is successful, then your valid reply will need to include the size of the contents. This size, however, cannot be determined until the command completes. As such, you should have the command redirect to a file (use the child process's `pid` in the filename to avoid conflicts), read and reply with the contents of the file once the child has terminated, and remove the file after the reply is complete.

¹This brings with it serious security implications, so don't leave your server running for very long and implement the `..` check (see details) early.

Additional Details

- The client may close the connection while the server is processing the file. Be sure to handle this case without the server crashing.
- We will require the user to always specify a filename. No special action will be taken if no file name is provided (i.e., the request has /)².
- We will not support the notion of users, so a URL with a ~ will not attempt to search a home directory.
- All file searches will be done from the directory in which the server is executed. Your server should prevent any attempt to access files in directories above its working directory (e.g., using ..). If such a request is received, reply with an error. Note that some browsers will automatically remove any .. in a filename, so you may need to test this behavior using `telnet` (see below).

Resources

As with all programs, you must be careful to properly manage any resources used. Since a webserver is supposed to run forever (or close to), you will need to be especially careful about your resource usage. Be sure to free memory and close unused file descriptors. You can use the `top` program to monitor memory usage while your program executes.

Testing

You can certainly use a browser to test your program. If run on the department machines (and you are running your browser from a machine on the campus network, i.e., behind the firewall), then you can use `http://machine_name:port.selected/file`. An alternative is to use `telnet`. When you execute `telnet` you can provide the machine name (or IP address) and a port. `telnet` will then provide a prompt at which you can input data to be sent to the server.

Submit

- All source code.
- A Makefile that will build your program.
- A README file that specifies how to build your program and that describes anything about your program that you feel I should know during grading.

Grading

Feature	Percentage
HEAD request	15
GET request	30
cgi-like support	25
Error Handling & Resource Management	20
Reasonable Decomposition	10

²Many servers will attempt to open a default file such as `index.html`.