MinCGI

You are tasked with building a web server that will handle multiple connections and execute applications that are compliant with a subset of the Common Gateway Interface specification. The Common Gateway Interface is used by web-servers to allow compatible programs to be executed on the server, to process HTTP requests and compute a HTTP response. The web-server uses standard input and output as well as environment variables for communication between the CGI program and the itself. HTTP requests that are received by the web-server can be passed to a CGI program via shell environment variables and standard input.

What is a web-server?

A web-server is a common application which serves web pages and commonly executes queries as part of a request. A web-server interacts with clients through the HTTP protocol, a client will send a HTTP Request, the web server will process it and send a HTTP response to the client. A client is typically a web-browser such as Mozilla Firefox, Google Chrome, Microsoft Edge and Apple Safari.

What is CGI?

Common Gateway Interface is a specification for a web-server to allow programs and scripts to process requests that are given to it via standard input and environment variables. A CGI Program will process the data and output a HTTP request via standard output.

HTTP Request

The following are request headers that are sent by a web browser. You will need to retrieve each one. Each header is **optional**, content-type and content-length are

- Accept - Content type of the request
- Host - Domain of the server
- User-Agent - The browser used by the client
- Accept-Encoding - Encoding of the stream
- Remote-Address - The IP Address of the client
- Content-Type - The content type of the request body
- Content-Length - Size of the request body in bytes

Query Strings is data after a `?` symbol that is placed at the end of the request URL. A query string will list variables in a string, each variable and their value pair will be separated by a `&` character. Below is an example of a query string where the variables are `name` and `age`.

```
http://localhost:9001/hello?name=James&age=25
```

A HTTP request will specify the method, resource name and the HTTP protocol version on the first line, followed by the request headers on each line afterwards. The request body is separated by an empty new line after the last request header.

```
<METHOD> <RESOURCE> <HTTP PROTOCOL>
<REQUEST HEADERS ...>

<REQUEST BODY>
```

Example - GET Request 1

URL Example:

```
http://127.0.0.1/
```

Protocol Example:

```
GET / HTTP/1.1
```

Example - GET Request 2

URL Example:

```
http://127.0.0.1/
```

Protocol Example, web browser can provide additional headers:

```
GET / HTTP/1.1
Host: 127.0.0.1
```

Example - GET Request 3

URL Example:

```
http://127.0.0.1/js/helper.js
```

Protocol Example:

```
GET /js/helper.js HTTP/1.1
Accept: application/javascript
```

Example - GET Request 4 (Query String Example)

URL Example:

```
http://127.0.0.1/cgibin/posts?id=20
```

Protocol Example:

```
GET /cgibin/posts?id=20 HTTP/1.1
Host: 127.0.0.1
```

CGI Environment Variables

The following are common gateway variables which you will retrieve from request headers, configuration file and execution environment. You will need to set these variables (if found in the above sources) as a shell environment variable for the CGI Program.

- HTTP_ACCEPT - Content type of the request, this value is retrieved from the HTTP request, omitted if it is not part of the request
- HTTP_HOST - Domain of the server, this value is retrieved from the HTTP request, omitted if it is not part of the request
- HTTP_USER_AGENT - The browser used by the client, this value is retrieved from the HTTP request, omitted if it is not part of the request
- HTTP_ACCEPT_ENCODING - Encoding of the stream, this value is retrieved from the HTTP request, omitted if it is not part of the request
- REMOTE_ADDRESS - The IP Address of the client, this value is retrieved from the connection
- REMOTE_PORT - The port the client is connected on, this value is retrieved from the connection
- REQUEST_METHOD - The HTTP method, this value is retrieved from the HTTP request
- REQUEST_URI - The resource requested by the client, this value is retrieved from the HTTP request
- SERVER_ADDR - The server address, this value is retrieved from server socket.
- SERVER_PORT - The port the server is listening on, this value is retrieved from configuration or server socket
- CONTENT_TYPE - Content type of the information supplied by the body of a POST request, omitted if it is not part of the request (POST request only)
- CONTENT_LENGTH - Length of the request body, this value is retrieved from HTTP request, omitted if it is not part of the request (POST request only)
- QUERY_STRING - Information after a '?' at the end of the request URL. Will map variables to a value in a single string, separated by '&' characters, omitted if it is not part of the request

You will need to pass the HTTP request body via standard input to the CGI Program. The request body is important when form data is sent to the server.

Content-Type Mapping and Processing

Your web-server will need to map each file extension to the following content-type. When a http requests a file of one of the following type, the content-type should be mapped to the following.

- Any file ending with `.txt`, is mapped to `text/plain`
- Any file ending with `.html`, is mapped to `text/html`
- Any file ending with `.js`, is mapped to `application/javascript`
- Any file ending with `.css`, is mapped to `text/css`
- Any file ending with `.png`, is mapped to `image/png`
- Any file ending with `.jpg` or `.jpeg`, is mapped to `image/jpeg`
- Any file ending with `.xml`, is mapped to `text/xml`

HTTP Response

Your server only needs to handle GET HTTP and be able to return statuses for both files and CGI programs. Your data must return the HTTP Protocol Version, Status Code and Status Message,

However, your web-server does not need to handle all of the different return statuses and these will typically be defined by the web application itself.

For files that will not involve the command gateway interface, your web-server should do the following.

- If the file exists, it should return `200 OK` and the content of the file. Make sure the content-type is set to what the extension maps to.
- If the file does not exist, it should return `404 File not found`, your server should send the following HTML to the client.

```
<html>
<head>
    <title>404 Not Found</title>
</head>
<body bgcolor="white">
<center>
    <h1>404 Not Found</h1>
</center>
</body>
</html>
```

Command gateway interface programs may encounter a problem during execution. The web server should check the exit code of the application executed to detect that the command executed and if the program returned data. If the command is not found or the program crashes (exits with non-zero status), the web-server must respond with `500 Internal Server Error`.

Special case: HTTP request on `/`, your server must retrieve a file named `index.html` at the root of your `staticfiles` path.

HTTP Responses follow the following form

```
<HTTP PROTOCOL> <STATUS CODE> <STATUS MESSAGE>
<RESPONSE HEADERS ...>

<RESPONSE BODY>
```

Variation for Status Code

CGI programs are able to specify the status code of the HTTP response as well as the message. If the status code is absent on the first line of the response, your web-server should assume it returns status code 200 with the message `OK` (`HTTP/1.1 200 OK`).

However, a CGI Program can specify the status code on the first line in this format.

```
Status-Code: <Status Code Value> <Message>
```

In the event, the CGI program specifies this format on the first line, your web-server should fix the HTTP response from CGI Program to format the first line by extracting the `Status Code` and the `Message` and using them as part of the standard response.

Example - Status 200, 1

HTTP Request

```
GET / HTTP/1.1
```

Example Response

```
HTTP/1.1 200 OK
Content-Type: text/html

<p>Hello World!</p>
```

Example - Status 200, 2

HTTP Request

```
GET /goodmorning.html HTTP/1.1
Accept: text/html
```

HTTP Response

```
HTTP/1.1 200 OK
Content-Type: text/html

<html>
<body>
    <h1>Good Morning!</h1>
</body>
</html>
```

Example - Status 404, 1

HTTP Request

```
GET /missingfile.html HTTP/1.1
```

HTTP Response

```
HTTP/1.1 404 Not Found
Content-Type: text/html

<html>
<head><title>404 Not Found</title></head>
<body>
    <h1>404 Not Found</h1>
</body>
</html>
```

Example - Status 200, 3, Image Type

HTTP Request

```
GET /image.png HTTP/1.1
Accept: image/png,*/*
```

HTTP Response

```
HTTP/1.1 200 OK
Content-Type: image/png

<image data>
```

Configuration File

You will read in a configuration file that will specify the static file directory and the cgi-bin directory for your application. As with most web servers, your configuration will need to know the port it is listening on.

The configuration file is a simple text file where each property is assigned to a value. When reading the configuration file, you will need to retrieve each of the following properties from the file and their value.

- staticfiles - string, path to the files that will be simply served by the web-server.
- cgibin - string, path to the CGI executables.
- port - integer, the port the server listens on.
- exec - string, executable path or command that will be executed on inbound cgibin requests.

This is a simple file where each line will have a key and a value separated by `=`.

```
staticfiles=./root
cgibin=./bin
port=8070
exec=/bin/python
```

The configuration file will be supplied as a command line argument for your server. For example, your server will be invoked like so `python webserv.py config.cfg`.

In the event the configuration is not passed to the program, the program should output `Missing Configuration Argument`.

If the path does not exist for the configuration file, the program should output `Unable To Load Configuration File`. If the configuration file does not contain the four fields specified, the program should output `Missing Field From Configuration File`.

Static Files

Your web server must retrieve all static files within a specified directory, this will also need to retrieve any file that exists within a folder in that directory. However, your web-server should forbid accessing files that exist in the parent directory of staticfiles.

Parse HTTP/1.1 - Supplementary

Your web server will need to parse HTTP/1.1, you can refer to HTTP/1.1 RFC located here: https://tools.ietf.org/html/rfc7230. This is a stateless protocol that will simply acknowledge a request and serve a file or in the case of `cgibin` files, parse the request, set the environment variables, launch the CGI Program and send the request body via standard input to the program.

Specifically, you can refer to section 2 which outlines the HTTP URI scheme.

You may refer to the following examples of HTTP requests that will be sent to a client. You can also test using your web browser by visiting your web server. You do not need to implement the full HTTP/1.1 specification such as cache-validation as this can be difficult to get right.

Common Gateway Interface - Supplementary

Common gateway interface for version 1.1 is outlined here: https://tools.ietf.org/html/rfc3875.

Common gateway interface is a web standard protocol for web-servers to communicate to executables and pass data to and from to these executables. Specifically, cgibin has its own executable folder and the configuration format outlines the executable to be used with the files within this directory.

The CGI Request is outlined in section 4 of the Common Gateway Interface RFC. The interface is quite simple and just involves using the standard input and standard output file descriptors as described in previous sections.

Extension

Try to implement one of the following extensions for your web server.

- Handling POST requests from browsers and accepting **form** data.
- Compressed packets to send back to the browser. Use the gzip module in python.
- Your own extension, please check with a staff member about the eligibility of your extension. It must be relevant to the problem and demonstrate a similar amount of complexity to the extensions above.

You must provide test cases to demonstrate how it works and provide a README with your submission outlining your extension.

CGI-Web for everyone

Refer to below as a task list for your web server. Prioritise the configuration, accepting connections and static files before getting CGI Programs working.

- Load a configuration file
- Accept connections
- Parse HTTP/1.1 data
- Set up all environment variables
- Fork and execute the CGI Program or retrieve static file
- If CGI Program, pipe the data from the application back to the web-server
- After implementing your server, try to improve performance and make all requests asynchronous by forking each connection that is accepted.

Marking - Due 22/11/20 11:59 PM

The assessment weight is 20%, it is divided into 3 parts. Your submission will be assessed on the following.

- Automatic Test Cases (10%) - Automatic test cases your submission will be submitted to. This will check that your application is compliant with the MinCGI specification.
- Test Cases (4%) - Your submission must include your own set of test cases, these test cases should be shell scripts that make a request to your web-server using `curl` and check the final result against an expected file using `diff`.
- Manual Marking (4%) - Your submission will be marked against a style, please make sure you have a consistent style and conform. You can supply a README file with your submission that describes your programming. If you do not supply a README that discusses your programming, we will assess your source against the PEP-8 specification.
- Extension (2%) - After implementing your CGI web-server, extend your it by implementing one of the extensions above.