Homework #3

B05902120 / Yu-Ting, TSENG

Jan 2, 2018

Problem description

The goal of this assignment is to be familiar with the concepts of signal and inter-process communication covered in Chapter 10 and 14 in the textbook. In this assignment, you will handle the communication between parent processes and child processes faultlessly between a web server and CGI processes.

In this assignment, we will provide you a sample web server, which you can download from ceiba. First you should write a simple program called `file_reader`, which does a simple job: read a `filename` from `stdin` and write the content of the file to `stdout` if the file is accessible. If the file is not accessible, the program should output appropriate error message to `stdout` and exit with a nonzero return code.

In the following texts, CGI program just refer to some simple program that reads from `stdin` and writes its result to `stdout`. Then, your task is to revise the web server to assure that the web server can handle the CGI requests faultlessly, including invoking CGI program and communicating with the CGI program, and write the proper information to the output. The use scenario of the application is described in the following:

The web server will be started first. After the web server starts, the user can send the request to the web server via a URL like the following:

`http://your_ip:port/cgi_program?filename=filename`

where `cgi_program` specify the name of CGI program to be invoked. This name should only contain 'a' 'z', 'A' 'Z', '0' '9', and '_', or your web server should inform the client of an error. The first and only query parameter is the `filename`. The `filename` should only contain 'a' 'z', 'A' 'Z', '0' '9', and '_' too, or your web server should inform the client an error.

The web server interacts with CGI process by passing the `filename` to `stdin` of `cgi_program`. If everything works correctly in the above example, with `cgi_program` being `file_reader`, the server process will output the content of the file `filename`.

On the other hand, if the CGI program crashes or encounters any error, the web server should handle it properly and notify the client.

Your server should properly write HTTP header to the client. You can refer to the comment in line 274 284 in sample code as an example of how to write a valid (although simple) HTTP header. You may need to change "`200 OK`" to some other status, and the argument of "`ContentLength:`" to real value of the output.

In this assignment, you should complete the following tasks:

1. Use your `cgi_program` program to read the data form `filename` and show on web browser.

2. If `cgi_program` or `filename` doesn't match the condition above(invalid name), you should notify the client by HTTP status code "`400 Bad Request`", with some appropriate error message.

3. Server executes `cgi_program` by `fork` and communicates with child processes by `pipes`. After the CGI program writes output to its `stdout`, your child process should return the content back to parent process using only `pipes`, then the parent would add HTTP header to it, and output the result to the client.
   But if the CGI program exits with a nonzero exit code, the server should change HTTP status code to "`404 Not Found`", and output the error message of the CGI program.

4. When a user requests a `cgi_program` or `filename` which does NOT exist, you should notify the client by HTTP status code "`404 Not Found`", with some appropriate error message.

5. If the URLis: `http://your_ip:port/info`, your server should **NOT** try to execute a CGI program named `info`. Instead, your server forks a child process and the child process sends `SIGUSR1` signal to the server process.

After the server process receives `SIGUSR1`, it shows the child processes' information, including the CGI processes that are still running (with pids each) and the CGIs that are terminated after the server process starts.
You may write a CGI program and make it `sleep(5)` so that you can show the running processes' `pid`. Of course, other methods will be fine if you can demo this situation.

6. Use `mmap()` to record the exit time of your `cgi_program` and the `filename`. Your server should read the info from the share memory and show the last exit time and the filename on `http://your_ip:port/info`.

Format for input and output

The request to server:

```
./server [port] [logname]
```

If the content of file `example_file` is "`abcdef`", and when you type the url in the browser: `http://yourip:port/file_reader?filename=example_file`, where `yourip` is the ip where you're running the server, you should see

```
abcdef
```

If you type the url in the browser: `http://yourip:port/info`, you should see something like the following in the browser:

```
2 processes died previously.
PIDs of Running Processes:  1034, 4871, 5327
Last Exit CGI: Sat Dec 3 22:08:44 2016, Filename:  [filename]
```

Tasks and Scoring

There are 7 subtasks in this assignment. By finishing all subtasks you earn the full 7 points.

1. Successfully read and show data on web server. (1 point)
   - Your program should be able to read a file on web browser(`200 OK`).
2. I/O multiplexing. (1 point)
   - Your program should be able to handle multiple connections at once.

- You can demo this by using some slow responding `cgi_program`, with multiple connections to the server.

3. Multiprocess. (1 point)
   - Your program should be able to handle multiple running `cgi_programs`.
4. 400 Bad Request. (1 point)
   - Your program should return the right HTTP status code "`400 Bad Request`", if `cgi_program` or `filename` has invalid name.
5. 404 Not Found. (1 point)
   - Your program should return the right HTTP status code "`404 Not Found`", if `cgi_program` or `filename` doesn't exist.
6. Show process info. (1 point)
   - Show the correct info on `http://your_ip:port/info`, including `number of died processes` and `running processes`.
7. Show process advanced info. (1 point)
   - Show the exit time and the `filename` of your last exit CGI program on `http://your_ip:port/info`.

Notes

- All output / error messages above do not need to be strictly as the same as described above. For example, you can add some html tags around them. However, when you demo, it should be easy to identify which message corresponds to which case. Feel free to add more functionality or make the output look better.
- Also, you should not change the name of "`file_reader`", the url format given, and the server should behave correctly when given these urls.
- It would be beneficial to output some debug messages to console, so you can know what the server is doing better. It may also be useful in demo.

- You can assume that TA would not try to use some edge case / extreme case to challenge your server. (e.g. super long query string, weird characters in url, ...). But it's encouraged to handle as many exception cases as you can.