2013 System Programming - Assignment 7

Announce: Wed, Dec 25, 2013
Due: 23:59 Tue, Jan 14, 2014

I. 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 non-zero 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 "Content-Length: " to real value of the output.

In this assignment, you should complete the following tasks:

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

Handle Error condition

But if the CGI program exits with a non-zero exit code, the server should change HTTP status code to "404 Not Found", and output the error message of the CGI program.

3. When a user requests a CGI program which does NOT exist, you should notify the client by HTTP status code "404 Not Found", with some appropriate error message, e.g. "CGI Program not found".
4. If after the server executes fork(), the child process closes pipes before the parent write queries to its child. The server should notify the client by HTTP status code "500 Internal Server Error", with some appropriate error message, e.g. "Pipe closed by client".
5. If after the server starts the CGI program, the child process terminates abnormally, the server should determine whether the CGI program terminates normally or not, and notify the client by HTTP status code "500 Internal Server Error", with some appropriate error message, e.g. "CGI Program terminates abnormally".
6. If the URL is:

   `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.

Your job is to write your own server, i.e., parent process, and some CGI process, i.e., child process.

II. Sample execution

```
$ ./server 12345 tmp.log
```

The sample code has 2 arguments: `[port]` and `[logname]` (not used in the sample code). You can change the arguments to whatever you want. The sample code doesn't do much things, but if you've finished your server, you should observe the following behavior:

If the content of file `example_file` is `"abcdef"`, and when you type the url in the browser:

`http://``your-ip`**`:12345/file_reader?filename=example_file`**

where `your-ip` is the ip where you're running the server, you should see "abcdef" in your browser. If you type in the url:

`http://``your-ip`**`:12345/info`**

you should see something like the following in the browser:

```
3 processes died previously.
PIDs of Running Processes: 1034, 4871, 5327
```

III. Tasks and scoring

There are 7 subtasks in this assignment. By finishing all subtasks you earn the full 14 points.

1. I/O multiplexing (2 points)
   Your program should be able to handle multiple connections at once. You can demo this by using some slow-responding CGI programs, with multiple connections to the server.
   Hint: As in Programming Assignment 1, use select().
2. Multi-process (2 points)
   Your program should be able to handle multiple running CGI programs.
   Hint: fork() & exec()
3. Show process info (3 points)
   Your program should handle the case when the url is `/info`, and correctly output the result.
   Hint: Handle signal `SIGUSR1, SIGCHLD`.
4. Detect the limits on `cgi_program` / `filename` (1 point)
   You should correctly impose the limits to these arguments of the url.
5. Detect the CGI program which close the pipe before the server writes to it (2 points)
   Your program should be able to detect if the CGI program closes the pipe before the server writes to it, and return appropriate error messages.
   To demo this, it's fine to add some special case in your server (for demo purpose only), e.g. delay the server write in these cases. Such special cases may be triggered by some name of `cgi_program`.
   Hint: Handle signal `SIGPIPE`.
6. Detect the CGI program when it terminates abnormally (2 points)
   Your program should be able to detect if the CGI program terminates abnormally, and return appropriate error messages.
   Hint: Use wait()/waitpid() and WIFEXITED().
7. Detect the CGI program when it exits with a non-zero exit code (2 points)
   Your program should be able to detect if the CGI program exit with a non-zero exit code, and return appropriate error messages.
   Hint: WEXITSTATUS().

IV. Submission

For this assignment, each student would have 5 minute time to demo his server and each functionality. You should prepare some CGI programs that can be used for the demo of each task above (and convince the TA that you do really conquer that subtask).

The score would be only based on the demo, NOT on any automatic testing.

Your assignment should be submitted to the course website by the due. Or you will receive penalty. At least four files should be included:

1. `server.c`
2. `file_reader.c`

3. Makefile (as well as other *.c)
4. REAMDE.txt

Put your student ID (lower-case) and name in a comment at the first line of your source code, with the following format

```
/* STUDENT_ID NAME */
```

In `README.txt`, please briefly state how to compile your program, how to run your server, what functionality had you done, and anything you have done besides the basic functionality.

These files should be put inside **a folder named with your student ID** and you should compress the folder into a `.tar.gz` before submission. Please do not use `.rar` or other file type.

The commands below will do the trick. Suppose your student ID is b02902000, and your name is 林聆凌:

```
$ mkdir b02902000
$ cp Makefile README.txt *.c b02902000
$ tar -zcvf SPHW4_b02902000.tar.gz b02902000
$ rm -r b02902000
```

And the first line of `merger.c` would be

```
/* b02902000 林聆凌 */
```

Please do **NOT** add executable files to the compressed file. Error in the submission file (such as files not in a directory named with your student ID, or so on) may cause deduction of your credits. (Really expensive, isn't it? Please follow them carefully!)

V. 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.

To demo that you returned the correct HTTP status code, you can use inspector in Chrome or Firefox (ctrl+shift+i, Network) to show the status code.

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.

Please try to use informative name for the `cgi_program` in your demo. For example, to demo the detection of CGI program terminates abnormally, using `crash` as the name of cgi_program would be better than using `program_6`.

VI. Punishments

- Plagiarism

  Plagiarism is strictly prohibited.

- Late punishment

Your credits of this assignment will be deducted 5% for each day of delay submission. (That is, you will lose all your credits on this assignment after 20-day delaying!)

Even though your credits will be deducted for delaying, a late submission will still be much better than absence. Just remember one thing: there are System Programming courses to attend next morning.

If you have any question, refer to the slide to see how to contact us.