Assignment 1
Web Architectures 2018-2019

Nicolò Pomini

September 22, 2018

1 Introduction

The goal of this assignment is to edit a concurrent HTTP mini-server, in order to make the latter able to respond to a specific request activating an external process. Particulary, for all the requests that contain the token *process*, the server launches an external process (`localhost:8000/process` for instance). Furthermore, the requests with the form `process/reverse?par1=string&par2=booleanvalue` have to be satisfied using an external process that takes the two parameters, and in case the second one is true, it returns the reversed string of the first parameters, otherwise it returns whether the string of the first parameter is palindrome or not.

2 Explanation

The problem was solved creating two types of *external* processes:

- a *specialized* process, whose task is to fulfill the *reverse* query (`process/reverse?...`);
- a *generic* process that does a simple print on the standard output, which is launched in case the request is in the form `process<whatever string different than "/reverse">`.

Each thread that manage a single connection analyzes the request, deciding if it is the case to launch a generic process, a specialized process or to return the requested file, as the original version of the server was doing.

3 Implementation

The whole is developed in Java, in order to avoid any possible trouble with interpreters or compilers different from that of the server. In this way the entire system can be deployed and run on the same machine of the original version of the HTTP mini-server.

As regards the *generic* process, it is implemented with a simple `main` class that performes a print on the standard output. This file is called `Empty.java`

The *specialized* process for the reverse/palindrome string expects the two parameters (`par1` is the string and `par2` is the boolean value) passed as arguments. Its implementation is very easy: it checks if the `par2` is true, and in this

case it prints on the standard output the `par1` string reversed; if `par2` is false, it is checked if the `par1` string is palindrome, if it is the case the process prints `true`, otherwise `false`. In order to recognize what the server has to return as response to the request, a small and simple protocol was used: the reversed string, or the answer to the question about the palindrome string are preceded by a double asterisk. In this way is easy for the server to distinguish what are the output from what are the standard output or standard error messages. This file is called `Reverser.java`

Both the external process source files belong to the package `processes`.

The original server was modified, adding in the `TinyHttpdConnection` class the management of the request string in case it starts with the word *process*. In particular, if the *reverse* path is requested, the first part of the request string is cancelled (`"process/reverse?"`) in order to leave all the parameters of the query. At this point the string is splitted when the '&' character is faced, to separate every parameter, and `par1` and `par2` are looked for. In case both are found, the external process is launched, giving to it the two parameters as argument strings, otherwise an error message is displayed. The following code does this task.

```
int paramNumber = 0;    //counter for the parameters
String par1 = "";
boolean par2 = false;
String[] parameters = req.split("&");
for(String p: parameters) {
    if(p.startsWith("par1=")) {
        par1 = p.substring(5);
        paramNumber++;
    }
    else if(p.startsWith("par2=")) {
        par2 = Boolean.parseBoolean(p.substring(5));
        paramNumber++;
    }
}
if(paramNumber != 2) { //params are not enough
    System.err.println("Error, par1 and par2 are required");
    new PrintStream(out).println("Error, par1 and par2 are required");
} else {
    String output = startNewProcess(par1, par2);
    new PrintStream(out).println(output);
}
```

This portion of code allows the user to specify more parameters than `par1` and `par2`. If it is the case, the parameters in abundance are ignored. The function `String startNewProcess(String par1, boolean par2)` deals with starting a new process that analyzes the string `par1`. It prints on the standard output everything that the process prints on the same stream, and returns as a string what the server gives as response. To do that, it returns only those messages of the process that start with a double asterisk.

In case the *reverse* path is not requested, but the request contains the token *process*, a new process is started anyway. This process only prints a message on the standard output.

The following code start a new process. Firstly, the installation directory for the JRE is requested, secondly, the readers for the standard output and standard error of the process are declared, and finally the external process is launched. Once the streams are read, the termination of the external process is waited for, and then the readers are closed.

```
String javaexecutable = System.getProperty("java.home") + "/bin/java";
BufferedReader bri = null;
BufferedReader bre = null;
try {
    Process p = Runtime.getRuntime().exec(javaexecutable +
    " <process_name> " + " <arguments>");
    //at this point the external process has started
    bri = new BufferedReader(new InputStreamReader(p.getInputStream()));
    bre = new BufferedReader(new InputStreamReader(p.getErrorStream()));

    /*
    Code to read the streams and to do (eventually) something else
    */

    //Wait for the execution of the external process
    p.waitFor();

    //The external process is finished
    System.out.println("Process has finished.");
} catch (IOException | InterruptedException ex) {
    System.err.println("Exception catched: " + ex.getMessage());
} finally {

    try {
        if (bri != null) {
            bri.close();
        }
        if (bre != null) {
            bre.close();
        }
    } catch (IOException e) {
        System.err.println("Error while closing process streams.");
    }
}
```

The main class of the server was not modified.

4 Deployment

To deploy the server, some commands from the terminal are needed. Firstly, the working directory has to be the project main folder. Secondly, the source file folder is open.

```
cd src
```

Thirdly, the external process source files are compiled.

```
javac processes/*
```

Fourthly, the server is compiled.

```
javac tinyhttpd/*
```

Finally, the server is launched.

```
java tinyhttpd.TinyHttpd
```

In case the request does not contain the token *process*, the behaviour of the server is the same of the original version.

The following examples show the behaviour of the new part of the server:

- Usage with a non palindrome word

  Request: `http://localhost:8000/process/reverse?par1=roma&par2=false`

  Response (taken from the terminal):

  ```
  =========
  Request: GET /process/reverse?par1=roma&par2=false HTTP/1.1
  New process started
  false
  Process has finished.
  ```

- Usage to reverse a word

  Request: `http://localhost:8000/process/reverse?par1=roma&par2=true`

  Response (taken from the terminal):

  ```
  =========
  Request: GET /process/reverse?par1=roma&par2=true HTTP/1.1
  New process started
  amor
  Process has finished.
  ```

- Usage with a palindrome word

  Request: `http://localhost:8000/process/reverse?par1=anna&par2=false`

  Response (taken from the terminal):

  ```
  =========
  Request: GET /process/reverse?par1=anna&par2=false HTTP/1.1
  New process started
  true
  Process has finished.
  ```

- Behaviour with a generical `process` request

  Request: `http://localhost:8000/process/blabla`

  Response (taken from the terminal):

```
=========
Request: GET /process/blabla HTTP/1.1
Hello, I'm a new process
Process has finished.
```

- Behaviour with wrong parameters

  Request: `http://localhost:8000/process/reverse?a=4`

  Response (taken from the terminal):

  ```
  =========
  Request: GET /process/reverse?a=4 HTTP/1.1
  400 Bad Request: par1 and par2 are required
  ```

- Behaviour with a *reverse* request, with additional parameters (which are ignored and thus the request is satisfied correctly anyway)

  Request: `http://localhost:8000/process/reverse?par1=ciccio&par2=false&par3=tobeignored`

  Response (taken from the terminal):

  ```
  =========
  Request: GET /process/reverse?par1=ciccio&par2=false
  &par3=tobeignored HTTP/1.1
  New process started
  false
  Process has finished.
  ```

5 Comments and notes

The skeleton of the original HTTP mini-server is not changed, thus all the existing issues (security, memory usage, etc) persist.