Assignment 3

Computer Science 441

Due: 23:55, Friday November 4, 2022

Instructor: M. Ghaderi

1 Objective

The objective of this assignment is to practice server side network programming with TCP. Specifically, you will implement a multi-threaded web server from scratch to serve web objects to HTTP clients over the Internet.

2 Specification

2.1 Overview

In this assignment, you will implement a simple web server program called `WebServer`. Your server is required to handle `GET` requests only. It has to check if the requested object is available, and if so return a copy of the object to the client. The server should operate in non-persistent HTTP mode. This means that once an HTTP request is served, the server closes the underlying TCP connection. To inform the client that the connection is closed, include the connection close header line in the server response:

```
Connection: close
```

2.2 Implementation

To handle multiple connections, the server has to be multi-threaded. In the main thread, the server listens on a fixed port using a `ServerSocket` object. When it receives a TCP connection request, the server accepts the request and processes it in a separate *Worker Thread*. That is, once the server accepts a connection, it will spawn a new thread to parse the incoming HTTP request and send an appropriate response to the client.

- To prevent non-responsive clients from hogging server resources, if the worker thread does not receive a request from the client within a given time period, the worker thread must send an error message (with status code 408) to the client and close the connection.
- The working directory of the web server is its root directory. That is, if the requested object path is `/object-path` then the file containing the object is located on relative path `object-path` in the working directory of the web server.

Programs 1 and 2 provide a high-level description of the web server's main and worker thread functionality.

3 Server Response

An HTTP response consists of a status line followed by a number of optional header lines. In the case that the response includes an object, the content of the object is separated from the header lines by an empty line.

Program 1 Main Thread

```
while not shutdown do
    Listen for connection requests from clients
    Accept a new connection request
    Spawn a worker thread to handle the new connection
end while
Wait for worker threads to finish
Close the server socket and clean up
```

Program 2 Worker Thread

```
Set a timer to receive the client request
Read and parse the request
if request timeout then
    Send Request Timeout error response
else if format error then
    Send Bad Request error response
else if non-existence object then
    Send Not Found error response
else
    Send Ok response
    Send the object content
end if
Close the socket and clean up
```

3.1 Status Line

The status line consists of three components, which are separated by one or more spaces:

```
HTTP/1.1  status-code  status-phrase
```

As this is a simplified web server, your program needs to return responses with the following status codes and phrases only:

- `200 OK`
  Request is valid and the requested object is transmitted after the header lines.
- `400 Bad Request`
  There was a problem with the format of the request. An HTTP request consists of a request line and several optional header lines. In this assignment, we assume that header

lines (if any header lines are present in the request) are properly formatted and only the request line may have formatting issues. A properly formatted request line consists of three *mandatory* parts which are separated by one or more spaces, as follows:

```
GET /object-path HTTP/1.1
```

The command `GET` and protocol `HTTP/1.1` are fixed, while the `object-path` is optional. If no `object-path` is provided, *i.e.*, the request only specifies '`/`', then assume `index.html` by default.

- `404 Not Found`
  The requested object was not found on the server, *i.e.*, the corresponding file does not exists in the working directory of the web server.
- `408 Request Timeout`
  The web server did not receive the client request within a pre-specified time period. This time period is passed to the server as an input parameter.

3.2 Header Lines

To be compliant with what most web browsers expect from a well-behaving web server, your web server must include several header lines in its response. Each header line is formatted as follows:

```
field-name: field-value
```

where there is one (or more) space between '`:`' and `filed-value`.

- If the response code is `200`, include the following header lines:

  `Date` – current date on the server

  `Server` – name of your web server (your choice!)

  `Last-Modified` – get it from the file system

  `Content-Length` – length of the object

  `Content-Type` – type of the object

  `Connection: close`

- For any other response code, include the following header lines only:

  `Date` – current date on the server

`Server` – name of your web server (your choice!)

`Connection: close`

To format the date header field, you can use class `SimpleDateFormat` with format pattern "`EEE, dd MMM yyyy hh:mm:ss zzz`". Also, for the type of the object being transmitted, whatever type is returned by method `Files.probeContentType()` is acceptable for this assignment.

4 Web Server Shutdown

4.1 Shutdown Method

Your server program is required to implement a shutdown method. When the shutdown is called, the server should exit listening on the socket, clean up and terminate gracefully. To implement shutdown, you can either use `interrupt()/interrupted()` methods of class `Thread` or define a flag variable in your `WebServer` class and set it when the server is asked to shutdown. Either way, since `ServerSocket.accept()` is a blocking call, the server never checks the loop condition (see Program 1) while it is waiting for a client connection request. Thus, you need to force the server thread to periodically time out to return from the blocking method `accept()`, and check the shutdown status. This can be accomplished by setting the socket timeout option using method `ServerSocket.setSoTimeout()`.

4.2 Thread Management

Once the server is signalled to shut down, it has to wait a reasonable amount of time for the currently running worker threads to terminate before it terminates. You can either write your own code for thread management, *e.g.*, keep track of active threads in a list and then wait (some amount of time) for all of them to terminate, or use the executor service `ThreadPoolExecutor` for executing and terminating threads. Refer to Java documentation for details and examples.

5 Software Interfaces

5.1 Method Signatures

The required method signatures for class `WebServer` are provided to you in the source file `WebServer.java`. There are three methods that you need to implement, namely a constructor and methods `run` and `shutdown`. Refer to the Javadoc documentation provided in the source file for more information on these methods.

5.2 Exception Handling

Your implementation should include exception handling code to deal with all checked exceptions in your program. This includes catching the exceptions and printing exception messages (or the stack trace) to the standard system output. Note that `SocketTimeoutException` is part of the expected behavior of your code (due to setting socket timeout options) and should not result in the web server termination.

- **Exceptions in the server's main thread:** If the server can recover from the exception (*i.e.*, the exception only affected a worker thread) then continue with the normal execution of the web server. Otherwise, terminate the server in an orderly way (*i.e.*, clean up and close all streams and sockets) as if it was shutdown by the user.
- **Exceptions in worker threads:** Clean up, close all streams and sockets and terminate (*i.e.*, return from the `run` method) the worker thread.

5.3 Running Your Code

A driver class named `ServerDriver` is provided on D2L to demonstrate how we are going to run your code. Read the inline documentation in the source file `ServerDriver.java` for detailed information on how to use the driver class.

5.4 Testing the Server

You should be able to use Telnet/putty, your Assignment 2 implementation, or even a web browser to test your web server implementation. Yet another handy alternative is to use `wget` or `curl` utilities to send `GET` requests to your server. To connect to your server, you need to specify the server port number in the URL, as described in Assignment 1. You can give your server a port number between 1024 and 65535. Keep in mind that some web browsers default to HTTPS, so you need to change the settings of your browser to use HTTP for your web server, otherwise you will get a connection error.

5.5 Console Output

Add print statements (using `System.out.println`) in your code to print the following information on the console:

1. Client information (IP address and port number) every time the server accepts a client connection.
2. HTTP request (request line and header lines) every time the server receives a request.
3. HTTP response (status line and header lines only) every time the server sends a response.

Do not directly print anything else to the console beyond exception messages and the above information. For debugging purposes, you can use the global `logger` object defined in the driver class, whose level can be set using the command line option `-v`. Refer to the driver class source file for more information. The logger can be used to write messages to console during code development.

5.6 Design Document

Prepare and submit a design document to explain what happens in your program if the method `shutdown()` is called while some worker threads are still executing. Describe the sequence of events that happens in your implementation once method `shutdown` is called. Follow the design document formatting requirements on D2L.

Restrictions

- You are not allowed to modify the class and method signatures provided to you. However, you can (and should) implement additional methods and classes as needed in your implementation. Any changes in `ServerDriver` class will be overwritten during marking.
- You are not allowed to use classes `URL`, `URI`, `URLConnection` or their subclasses for this assignment. Ask the instructor if you are in doubt about any specific Java classes that you want to use in your program. In general, as long as you are writing your own code to implement HTTP protocol over TCP sockets, you should be fine.