**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

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

* 200 OK – request was processed successfully
* 400 Bad Request – there was a problem with format or semantic of the request
* 404 Not Found – object was not found on the server

**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 server socket object. When it receives a TCP connection request, it sets up a new TCP connection through another socket and services the request 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 process it.

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.

2

**Program 1** Main Thread

1. **while** not shutdown **do**
2. Listen for connection requests from clients
3. Accept a new connection request
4. Spawn a worker thread to handle the new connection
5. **end while**
6. Wait for worker threads to finish
7. Close the server socket and clean up

**Program 2** Worker Thread

1. Parse the HTTP request
2. **if** format error **then**
3. Send bad request error response
4. **else if** non-existence object **then**
5. Send not found error response
6. **else**
7. Send Ok response header lines
8. Send the object content
9. **end if**
10. Close the socket and clean up

**2.3** **Header Lines**

To be compliant with what most web browsers expect from a well-behaving web server, include the following header lines in your server response whenever the response is 200 OK:

* 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 response codes 400 and 404, include the following header lines:

* Date
* Server

3

* 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 as-signment.

**2.4** **Server Response**

You can use a String object to format the entire header part of the message using method String.format(). Then use String.getBytes("US-ASCII") to convert the header data into a byte array, which can be directly written to the socket output stream. To read the object content from the local file system, class FileInputStream can be used, which is a low level byte stream for reading both text and binary data.

**3** **Web Server Shutdown**

**3.1** **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().

**3.2** **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.

**4** **Software Interfaces**

**4.1** **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

4

and methods run and shutdown. Refer to the Javadoc documentation provided in the source file for more information on these methods.

**4.2** **Exception Handling**

Your implementation should include exception handling code to deal with all checked excep-tions in your program. Print exception messages (the stack trace) to the standard system output.

**4.3** **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.

**4.4** **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 test for proper multi-threading function-ality, a small program called IdleTcp is provided on D2L. You can run this program to open multiple TCP connections to the server, while running an HTTP test. Refer to the inline docu-mentation in the source file IdleTcp.java for more information. 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.

**4.5** **Console Output**

Add print statements (using System.out.println) in your code to print the following infor-mation on the console:

1. Client information (IP address and port number) every time the server accepts a client con-nection.
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

**4.6** **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 imple-mentation. 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.

6