

# CS421 - Computer Networks

## Programming Assignment 2 HTTP Parallel Download Agent Due Date: Dec. 27 2010, 11:59 PM

In this programming assignment, you are asked to implement an HTTP parallel download agent, an *HTTP/1.1 client* that transfers files over the Internet using parallel HTTP connections. You will test your HTTP client against HTTP servers running on our server machines. The client should support HTTP HEAD and HTTP *ranged* GET requests and allow the user to specify the number of parallel connections to be opened. The client should also be able to download parts (*ranges*) of the **same** file from different servers in *parallel*. Details about the expected command-line syntax and the input syntax can be found below.

For this programming assignment, you are **not** allowed to make use of any third party HTTP client libraries or the HTTP specific core or non-core APIs supplied with the JDK including but not limited to `java.net.HttpURLConnection` [please see the *Restrictions* section for a more detailed list]. The goal of this assignment is to make you more familiar with the internals of the HTTP protocol and to develop a basic notion of **multi-threaded network programming** and so, using any third party, JDK supplied core or non-core APIs providing any level of abstraction specific to the HTTP protocol is prohibited. In short, you have to implement the HTTP client using barely the Socket API of the JDK. If you have any doubt about what to use or not to use, please ask the course instructors or assistants.

The program itself must be a command-line Java application. Your main method (where JVM begins execution of your program) should reside in a public class named as **Downloader**. Your client should be runnable with the following shell command:

```
java Downloader
```

There are two scenarios that your program should support:

### First Scenario:

The first scenario is downloading a resource (file) from a **single** server using one connection or using more than one **parallel** connections. When your program is run, it should ask via stdout for the URL of the resource to be downloaded and the number of parallel connections to be used. The syntax of stdin input for this scenario consists of the following two parameters:

1. The URL of the file to be downloaded and,
2. The number of **parallel** HTTP connections to be opened in order to retrieve the specified URL.

So the single input line is as follows for the first application scenario:

```
<URL> <# of parallel connections>
```

### Second Scenario:

The second application scenario is to download the very same file from **multiple** servers using multiple **parallel** connections. In order to realize this scenario, we will set up multiple HTTP servers on different machines serving the same set of test files. The syntax of stdin input for this scenario consists of multiple URL strings each on its own line (one URL per line) **followed by an empty line**. Please note that these lines will NOT contain an integer for the number of connections to be opened. For this

application scenario, you will open one connection per URL specified. So the syntax of stdin input for this scenario is as follows:

```
<URL-1>
<URL-2>
.
.
.
<URL-n>
<empty line that terminates the input sequence>
```

Here are three example sessions that show the expected syntax:

Example for the First Scenario:

Please enter a URL followed by the number of parallel connections to be used or multiple URLs terminated with an empty line:

```
http://wlab.cs.bilkent.edu.tr/PA2/test15MB.bin 3
Downloading /PA2/test15MB.bin using 3 parallel connections...
File downloaded and saved under ./test15MB.bin
```

Example for the Second Scenario:

Please enter a URL followed by the number of parallel connections to be used or multiple URLs terminated with an empty line:

```
http://wlab.cs.bilkent.edu.tr/PA2/test10MB.bin
http://www.cs.bilkent.edu.tr/PA2/test10MB.bin
Downloading /PA2/test10MB.bin using 2 parallel connections...
File downloaded and saved under ./test10MB.bin
```

Another example for the Second Scenario:

Please enter a URL followed by the number of parallel connections to be used or multiple URLs terminated with an empty line:

```
http://wlab.cs.bilkent.edu.tr/PA2/test15MB.bin
http://wlab.cs.bilkent.edu.tr/PA2/test15MB.bin
http://www.cs.bilkent.edu.tr/PA2/test15MB.bin
Downloading /PA2/test15MB.bin using 3 parallel connections...
File downloaded and saved under ./test15MB.bin
```

In each of the above example sessions, boldface text denotes the standard input (stdin). Non-boldface text denotes the standard output (stdout).

The following discussion applies to both of the application scenarios:

If the number of HTTP connections to be opened is 1, then the client will simply transfer the file through a single connection making an *unranged* GET request. If the number of connections to be used is greater than one, then the program should download the file with the given URL using the specified number of HTTP connections in parallel and each of these connections should attempt to transfer an

**equal-length non-overlapping range** of the file by making an HTTP ranged GET request [please see the *Implementation Hints* section for a short discussion of ranged GET requests]. If the file cannot be divided into equal size parts, the sizes of individual parts should not differ by more than 1 Byte. The client should merge all the parts of the URL retrieved into a single disk file stored in the **working directory** of the program. You can safely assume that the file name component of the URL (but not the *path*!) will consist only of characters allowed in file names on Windows and Unix systems and hence you can use the file name of the URL as the name of the disk file to be stored. For instance, if the URL to be retrieved is `http://wlab.cs.bilkent.edu.tr/PA2/test15MB.bin`, then the file stored in the working directory of the client should be named as “test15MB.bin”.

Also, you are kindly asked to perform some delay measurements as explained in the Report section below.

## Implementation Hints

Each of the parallel HTTP connections should request non-overlapping ranges of the URL using HTTP ranged GET requests. You can specify the byte range of the file to be retrieved with the Range request header in the HTTP GET request. A sample GET request with the Range request header looks like as follows:

```
GET /PA2/test.bin HTTP/1.1
Host: wlab.cs.bilkent.edu.tr
Range: bytes=5000000-9999999
```

The above request is not for the whole file but rather for only the specified byte range of the file ([5000000-9999999]). This means that an HTTP/1.1 server will respond only with the specified bytes of the file rather than transferring all the bytes from the very first byte to the last byte of the file (i.e. the *whole* file). The motivation of this programming assignment is to transfer **multiple, non-overlapping** ranges of the same file simultaneously (in parallel) using multi-threaded programming. For example, if the file to be transferred is 300 bytes long and the number of parallel HTTP connections to be opened is 3, then the client should request byte ranges [0-99], [100-199] and [200-299] using ranged GET requests in 3 separate execution threads and store the retrieved ranges in a single disk file (in the correct byte order) in the working directory of the program. The byte ranges requested should be non-overlapping, meaning that no same byte of the file should be downloaded via two different connections. In order to maximize the expected performance, the ranges should be of equal size but if this is not possible (i.e. the length of the file is not evenly divisible by the number of connections specified or implied), the sizes of the ranges requested in parallel should not differ by more than 1 Byte.

In order to calculate the byte ranges to be requested in parallel, you need to know the file length in advance. You should obtain the size of the file via an *initial HTTP HEAD request*, to which the HTTP server will respond with a `Content-Length` response header (and with other related headers like the response code, last modification date, etc.). Having learned the file's size, the client will then calculate the byte ranges to be requested in parallel and start the individual threads to request those ranges with HTTP ranged **GET requests**. Please do not make any assumptions on the file sizes or names in your implementations because we will not use the same set of files that we have supplied for testing purposes while evaluating your submissions. You can find more information about the HTTP/1.1

protocol in RFC 2616 [<http://www.ietf.org/rfc/rfc2616.txt>].

## Testing Hints

You may test your client (in both of the scenarios) using the following URLs for the same file test5MB.bin:

- <http://wlab.cs.bilkent.edu.tr/PA2/test5MB.bin>
- <http://www.cs.bilkent.edu.tr/~ulucinar/CS421/PA2/test5MB.bin>
- <http://wlab2.cs.bilkent.edu.tr/PA2/test5MB.bin>

Also, the list of URLs for the file test15MB.bin is:

- <http://wlab.cs.bilkent.edu.tr/PA2/test15MB.bin>
- <http://www.cs.bilkent.edu.tr/~ulucinar/CS421/PA2/test15MB.bin>
- <http://wlab2.cs.bilkent.edu.tr/PA2/test15MB.bin>

Both of these files are binary files. Please note that while evaluating your submissions, we will test your clients with different sets of files on different servers. You can also test your client against any HTTP/1.1 server in your local network or on the Internet.

## Report

Together with your Java code for the HTTP parallel download client described above, you should also prepare a report comparing delay measurements of multi-connection parallel downloading from a single HTTP server (instance of the First Scenario), single connection (unranged) downloading (instance of the First Scenario) and multi-connection parallel downloading from multiple servers (instance of the Second Scenario). In your report, please:

- 1) Give at least 10 delay measurements and their average for downloading <http://wlab.cs.bilkent.edu.tr/PA2/test5MB.bin> through a single HTTP connection. In order to perform this experiment, you will run your client with the number of connections parameter set to 1 in an instance of the First Scenario.
- 2) Give at least **10** delay measurements and **their average** for downloading <http://wlab.cs.bilkent.edu.tr/PA2/test5MB.bin> through **3 HTTP connections in parallel** (from the machine [wlab.cs.bilkent.edu.tr](http://wlab.cs.bilkent.edu.tr) with the number of connections parameter set to 3, in an instance of the **First Scenario**).
- 3) Give at least **10** delay measurements and **their average** for downloading test5MB.bin using the following list URLs in an instance of the **Second Scenario**:
  - <http://wlab.cs.bilkent.edu.tr/PA2/test5MB.bin>
  - <http://www.cs.bilkent.edu.tr/~ulucinar/CS421/PA2/test5MB.bin>
  - <http://wlab2.cs.bilkent.edu.tr/PA2/test5MB.bin>

Since the list contains 3 URLs, you will have used **3 HTTP connections in parallel** to download the file.

**Please repeat each of the above three sets of experiments with the same settings** and report your measurements and their averages with the file **test15MB.bin**. Hence the set of URLs to be used for this second part is:

- <http://wlab.cs.bilkent.edu.tr/PA2/test15MB.bin> [to be used in experiments 1, 2 and 3]
- <http://www.cs.bilkent.edu.tr/~ulucinar/CS421/PA2/test15MB.bin> [only for experiment 3]
- <http://wlab2.cs.bilkent.edu.tr/PA2/test15MB.bin> [only for experiment 3]

So you should have performed a total of [# of sets]x[# of experiments in each set]x[# of files to repeat each experiment with] =  $3 \times 10 \times 2 = 60$  experiments.

After performing these experiments and reporting your measurements and their averages, in your reports, we kindly ask you to:

- I.** Give a model for the network delay experienced by your client (like the simple delay model given in the first chapter of your course book, you can just use the same model if you believe it fits your setup).
- II.** Compare your average delay results for the single connection downloads (experiment set 1) and the parallel connection downloads of both application scenarios (experiment sets 2 and 3) and using the delay model you give, please try to rationalize your results. You may structure this discussion as follows:
  - First, please compare and discuss the (First Scenario) single connection downloading, the First Scenario parallel connection downloading and the Second Scenario parallel connection downloading average delays for the file “test5MB.bin”.
  - Then, please compare and discuss the (First Scenario) single connection downloading, the First Scenario parallel connection downloading and the Second Scenario parallel connection downloading average delays for the file “test15MB.bin”.
  - Finally please compare and discuss the (First Scenario) **single connection** downloading average delay for “**test5MB.bin**”, the First Scenario **parallel connection** downloading average delay for “**test15MB.bin**” and the Second Scenario **parallel connection** downloading average delay for “**test15MB.bin**”.

In your discussions, please be respectful to the reality of *your* experimental setups. For example, you may normally assume relatively small processing delays but if there is an issue with your HTTP implementation, the “processing delay” component, which may be present in your delay model, may dominate other components. Hence you may come up with “unexpected” average delay measurements. The point is that you should try to explain your experiment results and not the expected results.

## Restrictions

1. Since the purpose of this assignment is to gain hands-on experience with socket programming, the following classes (and others like them) must **not** be used:
  - Any class in the Java library beginning with the letters "HT"
  - Any class in the Java library beginning with the letters "Ht"
  - Any class in the Java library beginning with the letters "UR"
2. Usage of any third-party native or Java libraries is prohibited.
3. If, in any circumstance, you are in doubt about whether you are allowed to use an API or not, please consult course assistants or instructors.

## Submission Rules

You need to apply all the following rules in your submission. Any violation of these rules will result in discarding of your submission.

- You can do the assignment either individually or in groups of two. You can also choose your partner from the other section. If you implement the assignment as a group, please make a single submission for the group using the *CS421 File Submission System* as explained below.
- The assignment should be submitted via the *CS421 File Submission System* available at <http://wlab.cs.bilkent.edu.tr/SubmissionWeb/>. Please note that the submission system may not accept those files that do not conform to the submission rules explained in this section or to the programming rules explained in previous sections. The submission system shall be running even after the submission deadline. If you have done the assignment on your own, then you should leave the second text box (titled as “Student id #2”) empty and just put your Bilkent ID in the text box titled as “Student id #1”. If you have done the assignment as a group, then you should fill in both text boxes with the participating students' Bilkent IDs.
- Please use your **Bilkent student IDs** (and not your TC IDs) at *CS421 File Submission System*.
- All the files must be submitted in a **zip** file whose name is obtained by concatenating your name, your surname and your Bilkent ID. If your name is Ali Velioglu and your Bilkent ID is 20209999 then the zip archive's file name should be “AliVelioglu20209999” (without the quotation marks). For groups of two people, the name and the student ID of the second participant of the group should be concatenated to the first student's name and Bilkent ID. Following the example given above, if the second student's name is Veli Alioglu and his ID is 20208888, then the name of the zip archive should be **AliVelioglu20209999VeliAlioglu20208888.zip**.
- The file name should not contain any Turkish characters or spaces. The file must be a .zip file, not a .rar file or any other compressed file.
- Any other methods (e-mail/disk/CD/DVD) of submission will not be accepted.
- Please try to make a single submission. In case of duplicate submissions, the last one shall be graded.
- The programming language used for implementation must be Java. The virtual machine used to test the code will be version 1.6. Your sources should compile cleanly (warnings are OK). If your code does not compile, you will either receive **no credit** or your grade will be penalized **severely**.
- All of the files must be in the root of the zip file, directory structures are not allowed.
- Please note that this also disallows organizing your code into Java packages.
- In addition to the source code, the archive should contain a README file explaining your implementation, detailing any issues and/or other implementation details. If your program is incomplete, please clearly describe what works and what does not. This file should also serve as a user manual.
- The archive should NOT contain:
  - Any class files or other executables
  - Any third party library archives (i.e. jar files)
  - Any text files other than plain text files

- Project files used by *IDEs* (e.g. JCreator, JBuilder, SunOne, Eclipse, Idea or NetBeans...) You may, and are encouraged to, use these programs while developing, but the end result must be a clean, IDE-independent program.

● **The standard rules for plagiarism and academic honesty apply, if in doubt refer to “Student Disciplinary Rules and Regulation”, items 7.j, 8.l and 8.m.**