Lab: HTTP Download Manager

Overview

Download managers, popularized in the mid-late 90s, have changed the way we download files from the internet. While today most of these features are integrated into modern browsers, you can still find them handy, e.g., to download files faster from the command line (see: aria2 and axel).

In this lab you'll get a glance into the implementation of download managers, you'll learn how to resume broken downloads, why using multiple HTTP connections accelerate some downloads, and how to download different parts of a file from multiple servers.

Resume Downloads

Your Download Manager needs to be able to recover from a previously stopped download (e.g., process stopped with a signal, power outage, network disconnection, etc.). Make sure to reliably store your progress to a file and use HTTP Range request (see: https://tools.ietf.org/html/rfc7233) to resume broken downloads.

Concurrent Connections

You'll need to support downloading a file using multiple HTTP connections. This download acceleration method is commonly used to circumvent per-connection server side limitations. Use the multithreading Java programming techniques acquired in previous semesters to manage multiple downloaders and a single disk writer thread. Use HTTP Range request to split the load between downloader threads.

Multi-server download

To avoid loading a single server, some files are uploaded to multiple servers (AKA mirrors). Your Download Manager will be able to fetch different parts of the file from different servers.

Usage Example

Resume Download

```
$ java IdcDm http://centos.activecloud.co.il/6.10/isos/x86_64/CentOS-6.10-x86_64-netinstall.iso
Downloading...
[13] Start downloading range (0 - 240123903) from:
http://centos.activecloud.co.il/6.10/isos/x86_64/CentOS-6.10-x86_64-netinstall.iso
Downloaded 0%
Downloaded 1%
Downloaded 2%
...
Downloaded 17%
^C

$ java IdcDm http://centos.activecloud.co.il/6.10/isos/x86_64/CentOS-6.10-x86_64-netinstall.iso
Downloading...
```

```
[13] Start downloading range (40988672 - 240123903) from:
http://centos.activecloud.co.il/6.10/isos/x86_64/CentOS-6.10-x86_64-netinstall.iso
Downloaded 17%
Downloaded 18%
Downloaded 19%
...
Downloaded 98%
Downloaded 99%
[13] Finished downloading
Download succeeded
```

Concurrent Connections

```
$ java IdcDm http://centos.activecloud.co.il/6.10/isos/x86_64/CentOS-6.10-x86_64-netinstall.iso 3
Downloading using 3 connections...
[13] Start downloading range (0 - 80041300) from:
http://centos.activecloud.co.il/6.10/isos/x86_64/CentOS-6.10-x86_64-netinstall.iso
[15] Start downloading range (160082602 - 240123903) from:
http://centos.activecloud.co.il/6.10/isos/x86_64/CentOS-6.10-x86_64-netinstall.iso
[14] Start downloading range (80041301 - 160082601) from:
http://centos.activecloud.co.il/6.10/isos/x86_64/CentOS-6.10-x86_64-netinstall.iso
Downloaded 0%
Downloaded 1%
Downloaded 2%
Downloaded 3%
...
Downloaded 97%
Downloaded 98%
Downloaded 99%
[13] Finished downloading
[15] Finished downloading
[14] Finished downloading
Download succeeded
```

Multi-server download

```
$ java IdcDm CentOS-6.10-x86_64-netinstall.iso.list 3
Downloading using 3 connections...
[13] Start downloading range (0 - 80041300) from:
http://mirror.nonstop.co.il/centos/6.10/isos/x86_64/CentOS-6.10-x86_64-netinstall.iso
[14] Start downloading range (80041301 - 160082601) from:
http://centos.activecloud.co.il/6.10/isos/x86_64/CentOS-6.10-x86_64-netinstall.iso
[15] Start downloading range (160082602 - 240123903) from:
http://centos.mirror.garr.it/centos/6.10/isos/x86_64/CentOS-6.10-x86_64-netinstall.iso
Downloaded 0%
Downloaded 1%
Downloaded 2%
...
Downloaded 96%
Downloaded 97%
Downloaded 98%
[14] Finished downloading
Downloaded 99%
[13] Finished downloading
[15] Finished downloading
Downloaded 100%
Download succeeded
```

Implementation Details

Create a command line application which accepts up to 2 parameters (in that order):

1. URL or File (containing list of URLs)
2. (optional) Maximum number of concurrent HTTP connections

The program will download the file specified in the URL (following redirects) into the current directory, e.g., "https://archive.org/download/Mario1_500/Mario1_500.avi" will be downloaded to "Mario1_500.avi".

The program may create additional files during download (i.e., metadata and temporary files), all of which:

- Should start with the same name as the downloaded file, e.g., "Mario1_500.avi.tmp".
- Should be deleted after a successful download.
- Should be smaller (in total) than **n/1024** in size, where **n** the size of the downloaded file.

The program should be able to properly resume download after the previous invocation was terminated due to a signal (any signal) or network disconnection (you should define relevant timeouts and document that in the code).

The program can decide to use less than the specified maximum number of concurrent HTTP connections if it considered the file to be too small (you should define "too small" and document that in the code).

The program should print its progress in terms of "percentage completed" similar to the example above (in case of a small file, it may print less than 100 times). A resumed download should continue from the last printed percentage. When finishing without getting a signal, the program should either print "download succeeded" or "download failed".

Do's and Don'ts

1. You are **not allowed** to use Java ready made classes that implement any of the 3 core features.
2. You are **not allowed** to use any external library (e.g., apache.commons, guava, etc.)
3. You are **allowed** to use anything from the following packages:
    a. java.io.*
    b. java.nio.*
    c. java.net.*
    d. java.util.*
4. You are **allowed** to assume:
    a. The program input is correct.
    b. The server in the URL supports Range queries, and specifically byte ranges.
    c. The URL resource size is known to the server and is returned in HEAD requests.
5. You are **encouraged** to use the provided design proposal, to save time on architecture (or you can write your own from scratch).
6. It is OK to use constant values in your code, as long as they are **well documented**. e.g.

```
Thread.sleep(1000); // adding maxBps to token bucket every second
```

7. Your code will be checked with Java 11 and should compile **without warnings**!

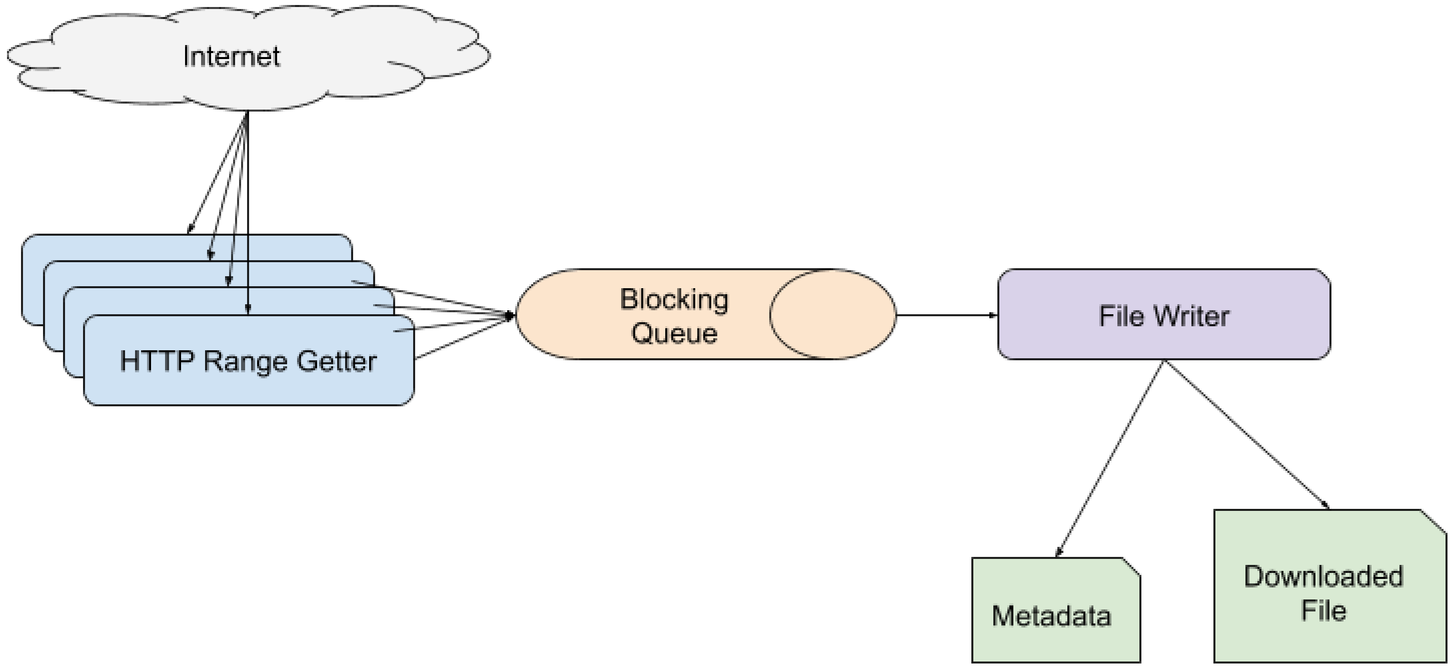
   a. You are allowed to suppress specific warnings, in a limited scope, with proper documentation.

```
$ javac -version
javac 11.0.2

$ javac -d build -source 11 src/*.java

$ cd build/

$ java IdcDm
usage:
        java IdcDm URL|URL-LIST-FILE [MAX-CONCURRENT-CONNECTIONS]
```

Error Handling

You should print only to STDERR (e.g., using `System.err.println`)
Your program shouldn't crash on exceptions, print a meaningful message, followed by "**Download failed**".

Design proposal



Tips, Spoilers, and Common pitfalls

1. The **HttpURLConnection** instance provided by **URL**'s **openConnection()** method allows you to manipulate many aspects of your HTTP request (e.g., method, headers, etc.), and some aspects of the connection (e.g. read/connect timeouts).
2. Splitting the download into chunks will allow more efficient communication between the downloader threads and the disk writer thread.

3. When writing to disk use **RandomAccessFile,** which allows you to open the output file in such a way that every change is written synchronously to disk, it also allows you to seek in the file before writing (useful when having multiple downloader threads).
4. Use Java Serialization to write your metadata to disk, but be careful- if your process gets a signal while writing a serialized object to disk you'll end up with corrupted metadata. One way to circumvent that issue is to serialize the metadata to another temporary file and then rename it.
5. When downloading from a server list, it might be easier to choose a server at random when starting the downloader thread then to handle consistent allocation
6. Test your downloaded file. The quickest way is to use the md5/sha1 utility available in most Operating Systems and compare its output for a file downloaded by your application, and the same file downloaded using the browser:
    a. **Linux**: `md5sum yourFileName`
    b. **MacOS**: `md5 yourFileName`
    c. **Windows**: `CertUtil -hashfile yourFileName MD5`

Submission guidelines

1. Submission **in pairs** by moodle by Sunday **26-01-2020**
2. File name should be lab_username1_username2.zip (e.g. "lab_israel.israeli_david.davidi.zip")
3. **Do not send any .class files in your zip**.
4. <u>Misformatted files will not be accepted.</u>
5. Make sure that your main function is found at the IdcDm.java file.
6. Attach a README file (in the .zip) containing
    a. The names of both students.
    b. For each file submitted: a single line describing its purpose.
7. **Use** the forum to ask any questions regarding this exercise.
    a. **Do not** upload any solution code to the forum.

Good luck!