Assignment 2: Peer-to-peer File Sharing

Due date: *Monday, November 27, 2017 at 22:00*

> *This is an individual assignment. You may discuss it with others, but your code and documentation must be written on your own. You might receive a bonus for an outstanding solution.*

Implement a peer-to-peer file sharing system consisting of a number of servers, the *seeders*, sharing a set of files with a client, the *downloader*. The file must be transferred in *chunks*: a single file is broken into fixed-size chunks, so that one or more downloaders can request different chunks of the same file from different seeders.

You must implement the two components of the system, called *Downloader* and *Seeder*, respectively.

Downloader takes multiple command-line parameters: first the file to download, and then one or more seeder addresses. *Downloader* must download the file using all the seeders available, whenever possible. The information about the file size, or the number of chunks, must be obtained from one of the seeders. The download must be resumable, that is, if *Downloader* is stopped before the download is complete, then restarting *Downloader* for the same file should download only the missing chunks, and not the entire file again. *Downloader* must log to the standard output every request sent to a seeder, stating also how many chunks the file is composed of, and how many chunks have been downloaded so far.

Below is a sample execution of the *Downloader* with a file composed of 1000 chunks.

```
$ java Downloader killbill.mp4 10.63.129.13:9999 10.63.129.15:1234 10.63.129.17:3421
  requesting chunk 0 to 10.63.129.13:9999
  requesting chunk 978 to 10.63.129.13:9999
  downloaded chunk 0 from 10.63.129.13:9999; progress 1/1000
  requesting chunk 250 to 10.63.129.17:3421
  ...
```

Seeder, instead, should listen for connections and respond to commands received from *Downloader*. The *Seeder* must share all the files contained in a specific directory. The *Seeder* application takes two optional command-line parameters. The first parameter is the listening port number; the second parameter is the directory in which the server manages files. The default directory is the current directory. Different seeders may manage different sets of files. *Seeder* must log to the standard output every request received by a downloader, stating the name of the file, the id of the chunk, and the address of the downloader. Below is a sample execution of the *Seeder*.

```
$ java Seeder 3421 data/
  sending killbill.mp4 chunk 0 to 10.63.129.2:7512
  sending killbill.mp4 chunk 7 to 10.63.129.2:7512
  ...
```

Notice that, in developing your peer-to-peer system, you must also design an appropriate protocol for the communication between downloader and seeder. You may use either TCP or UDP for the communication. You may statically define the size of a chunk. You may design your own strategy for choosing which seeder to contact to request a chunk. However, you must guarantee that a downloader will use multiple seeders (if there are many and if the requested file has enough chunks). You may assume that every file is uniquely identified by its name.

Extra points (up to 10%) will be assigned for implementations of *Downloader* and *Seeder* that use concurrent data transfers for faster downloads. You should limit the number of concurrent connections to a maximum of 16.

Submission Instructions

Use only the basic library functions for strings and for network sockets. You may not use any other network-specific library. Submit your solution through iCorsi in a single archive file. Include a README file with a list of all the material you used from other sources and a list of all known limitations and bugs of your solution.