**Programming assignment 1**

**File Sharing System (FSS) using only Java sockets**

You are charged with building a simple file sharing system (FSS) using a simple command line interface. The FSS consists of two types of entities: a *file server* and many *client*s. In this FSS:

1. The client MUST be able to upload files to the file server. It’s fine to upload them one at a time. If the file already exists, replace it.
2. The client MUST be able to download a given file from the file server, by providing the full filename path to the file server. If the file does not already exist, it MUST return an error message on standard error and return a non-zero error code.
3. The client MUST be able to list the file system objects (files and directories) on a file server directory, including the file server’s root (“/”) directory. If the requested directory does not exist, it MUST report an error message on standard error and return a non-zero error code.
4. The client MUST be able to create a directory (if the directory does not exist) and remove empty directories (if the directory is empty), and MUST be able to report whether or not these operations succeeded. If there’s an error, it MUST report an error message on standard error and return a non-zero error code.
5. The client MUST be able to remove a given file from the file server, by providing the full filename path to the file server. If the file does not already exist, it MUST return an error message on standard error and return a non-zero error code.
6. The file server MUST allow multiple clients to simultaneously connect to a single file server for upload/download, and allow for apparently-simultaneous transfer (e.g., it’s NOT okay for the file server to wait for one transfer to complete before another begins).
7. Clients MUST be able to cleanly shut down the file server.
8. The system MUST support the resume upload and download: If a file transfer between a client and a server is interrupted (because the network, server, or client has failed), the same client MUST be able to resume upload/download at the file server from the same point of progress by re-requesting the same filename. In other words, the client MUST NOT have to upload/download the data that is already uploaded/downloaded. Note that you have to handle server crashes too – you may want to use “flush()” in the server. In order for the TA to verify that you have implemented this functionality the program MUST print the progress of upload/download as it’s proceeding, including an indication that a partial download is skipping re-download of some portion of the file.

The client will need to know how to contact the server; for this exercise, the client MUST accept this in an environment variable named PA1\_SERVER with the computer name, a colon, and the TCP portnumber (e.g., “localhost:8000”). The server will need a place to store its data; its file system “root” MUST be the current directory for when it is started.

Do not make any other assumptions about the properties of the file. The file sharing system MUST be able to exchange files of any type and with arbitrary filenames. Note: Linux/Unix filesystems are normally case-sensitive, but Windows/MacOS are normally case-insensitive; you don’t need to provide case-sensitive functionality on non-case-sensitive systems, but it MUST maintain the upper/lower case provided to it since that information matters on some systems. If there’s no error, return an error code of 0 (no error).

When developing your system you MUST use Java, and for the networking portions you MUST use ONLY the Java sockets libraries (no JMI, etc.). No other use of third party software is allowed, unless explicitly permitted by the instructor. For purposes of this exercise we’ll totally ignore security, e.g., no authentication or authorization is needed. Note that illegally downloaded files (i.e., files violating the copyright laws) may not be used for developing, testing, or demoing your assignment.

**Deliverables**

1. Executable and self-contained jar files.
2. A read-me text file.
3. Source Code in Java.

**Executable and self-contained jar files**: The program MUST be implemented as a single jar file containing both server and client.

**A read-me text file**: The text file MUST provide step-by-step instructions for running and testing your software. It MUST also clearly list the location of server’s storage folder i.e. the location where the server stores the received files.

**Source Code in Java**: the source code submitted for your program MUST match the executable jar file.

Below are commands that your system MUST support (we plan to create a test script), in order of the overall requirements above. Unix syntax shown:

java -cp pa1.jar server start <portnumber> &

# To set up client so it knows what server to talk to:

export PA1\_SERVER=<computername:portnumber> # On Windows: set PA1\_SERVER=…..

java -cp pa1.jar client upload <path\_on\_client> </path/filename/on/server>

java -cp pa1.jar client download </path/existing\_filename/on/server> <path\_on\_client>

java -cp pa1.jar client dir </path/existing\_directory/on/server>

java -cp pa1.jar client mkdir </path/new\_directory/on/server>

java -cp pa1.jar client rmdir </path/existing\_directory/on/server>

java -cp pa1.jar client rm </path/existing\_filename/on/server>

java -cp pa1.jar client shutdown

Below is an example:

mkdir –p server ; cd server

java -cp pa1.jar server start 8000

cd .. ; mkdir –p client1 ; cd client1

export PA1\_SERVER=localhost:8000

java -cp pa1.jar client upload text1.pdf /folders/text1.pdf

*Uploading file …. 50% (changes real-time)*

*File uploaded.*

(start another client)

cd ..; mkdir –p client2 ; cd client2

java -cp pa1.jar client download /folders/text1.pdf copy\_of\_text1.pdf

*Downloading file …. 50% (changes real-time)*

*File Downloaded.*

java -cp pa1.jar client shutdown

Wrong instructions or faulty jar file would prevent the TA from grading your assignment.

Turn in your program (both source code and compiled jar) via Blackboard.