CS 549—Distributed Systems and Cloud Computing—Fall 2024
Assignment One—RMI and Sockets

You are provided with the partial RMI implementation of a simple FTP-like server and client. Your assignment is to complete the implementation and deploy it. You should complete the implementation of file upload and download using TCP sockets, as well as the code for binding the client to the server. You should use the Intellij IDEA Ultimate IDE for this and future assignments.

The code is structured into a Maven project called "`ftp`," with three sub-projects or "modules" called `ftpinterface`, `ftpserver` and `ftpclient`. A quick start introduction to Maven is available here: http://maven.apache.org/guides/getting-started/index.html. You should make sure that Maven is installed on your machine. It is available as a command line tool `mvn` on Unix machines. Type `mvn -version` to see which version (if any) you are running on Unix.

To summarize briefly: The philosophy of Maven is "convention before configuration." Rather than wasting a lot of time writing boilerplate make or ant scripts, Maven allows you to get started as quickly as possible by defining a lifecycle of "build phases" in software development, from code generation from tools, through compilation, unit testing, packaging as a jar file, etc. Maven itself is just an interpreter that executes "goals" for "plugins" that are defined for each of these phases (or you can execute a goal of a plugin directly). A description of the build lifecycle is provided here: http://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html. You execute the Maven command in the root module, `ftp`. If you type "`mvn install`," Maven will package the compiled code along with other resources into jar files, one for each submodule. If you type "`mvn clean`," Maven will delete the class files and packages, just leaving the source files. All of these commands are also available within Intellij IDEA, using the Maven tool window. Maven places the generated jar files in this directory:

```
~/tmp/cs549/ftp-test
```

with the names `ftpd.jar` and `ftp.jar`, for server and client respectively. Make sure that this directory is defined.

Type "`java -jar ftpd.jar`" to start the server. In another terminal window, type "`java -jar ftp.jar`" to start the client. The latter will start a command line interface once it has bound to the server. You will receive a diagnostic in the server window whenever a client binds. In the client window, type "`help`" for information about the client commands[1].

You can also run these commands with optional command line arguments:

```
java -jar ftpd.jar [ --serverIp server-addr ]
                   [ --serverPort server-port]
java -jar ftp.jar [ --clientIp client-addr ] --serverAddr server-dns
                                             --serverPort server-port
```

[1] The app assumes a folder called "root" in the test folder, and it serves the file system from that folder. If this folder does not already exist, you will have to create it. Make sure this folder has files and subfolders for testing purposes.

Once you have your code running locally, you should test it on EC2. You should run the client and the server on separate EC2 instances, using OpenJDK on those instances (install with yum if it is not already installed). You should ssh to the instances using the "`-i`" option to define the private key file that you use to authenticate yourself to the instance. Back on your own laptop, run scp[2] to copy the jar files to the client and server EC2 instances. Then run the client and server programs with the appropriate command line parameters (see above). There should again be a subfolder called "`root`" for the test file system, in the director where you run the server. You should be able to run the client on one instance and connect to the server running on the other instance. To do this, you will have to define a security group for the EC2 server instance that allows access from your client machine to the server. It is strongly recommended that you not simply disable the firewall for your server instance. Instead allow access on all ports from your client machine to your server machine, for passive file transfer. For active file transfer, you will have to do the opposite.

Once you have your code working, please follow these instructions for submitting your assignment:

- Create a zip archive file, named after you, containing a directory with your name. E.g. if your name is Joe Blow, then name the directory `Joe_Blow`.
- In that directory you should provide the zip archive that contains your sources, and the client and server jar files. Compile these jar files with the **local** profile.
- **Provide short videos demonstrating the successful testing of your instance**, with the scenarios described above (client and server in EC2 with active and passive mode file transfer).
- **Also include in the directory a completed rubric for your assignment.**

It is very important for your grade that you do adequate testing. You should at a minimum demonstrate testing of these scenarios:

1. Changing directory on the server, at multiple levels, up and down in the file system.
2. Listing the contents of the remote directory as you navigate the remote file system.
3. Upload and download of files.
4. Testing both active and passive modes (show the security groups for client and server EC2 instances).
5. Your test documentation should include videos demonstrating a working solution, with audio or documentation of what the videos are demonstrating.

Remember the format of the submission: A zip archive file, named after you, with a directory named after you. In this directory, provide these files: a zip archive of your source files and resources, client and server jar files, and a completed rubric and videos for your submission. Failure to follow these submission instructions will adversely affect your grade, perhaps to a large extent.

[2] Winscp is available on Windows. Alternatively, you can install Cygwin and do everything in Unix.