CS 549—Summer 2016
Distributed Systems and Cloud Computing
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 Java 8 and Eclipse Mars.

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.

The POM file for the main project defines several properties that are used in the three modules. You do not need to learn Maven or POM files, but you will need to modify this root POM:

1. There is a property `client.home` that defines the home directory in which you do testing on your home machine. Currently this is defined to be `${user.home}`, a Java property that is defined to be your home directory. However if you are working on a Windows machine, this will expand to a path with spaces in it (e.g. it will contain "`Documents and Settings`"), and there is an RMI bug where it assumes spaces as a delimiter in the codebase. You will need to modify the definition of `client.home` to be some path on your Windows box that does not contain spaces, e.g., `C:\CS549`.

It is recommended, though not required, that you use Maven goals defined in Eclipse. Information about Maven, including an on-line book and several video tutorials, are provided here: http://m2eclipse.sonatype.org.

Alternatively you will have to master the command line interface for Maven. If you are having problems with Maven within Eclipse, you should try issuing the Maven commands from the command line in the root directory of the parent Maven project. In either case, 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. The Maven scripts you are provided with where originally developed in Maven 2 and subsequently upgraded to Maven 3. Eclipse just calls out to Maven, it does not contain Maven itself.

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 directory, 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 Eclipse, by right-clicking the ftp project and choosing the "Run as…" submenu. If you build the jar files, these jar files are placed by default in

```
${client.home}/tmp/cs549/ftp-test
```

with the names `ftpd.jar` and `ftp.jar`, for server and client respectively. Make sure that this directory is defined. It should have a subfolder "`root`" that is the file system made available by the server to clients. Make sure this folder has files and subfolders for testing purposes.

The jar files include bash shell scripts for executing the server and client. Do the following (where you must replace `${client.home}` in the lines below, since it is a Maven property and not an environment variable):

```
cd ${client.home}/tmp/cs549/ftp-test
jar xf ftpd.jar ftpd.sh server.policy
jar xf ftp.jar ftp.sh client.policy
```

This extracts shell scripts for the server and client[1]. These scripts have paths coded into them for the location of jar files, codebase and policy files, based on the properties defined in the Maven root project POM file. Now type "`bash ftpd.sh`" to start the server. In another terminal window, type "`bash ftp.sh`" 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.

To incorporate the code you are provided with into Eclipse, select "`File | Import…`" You are given a pop-up window with a list of import sources. Select the "`Maven`" tab, then choose "`Existing Maven Projects,`" and navigate to the directory you obtained when you unzipped the archive file you have been provided with. This will import the root Maven project "`ftp`," and the submodules called "`ftpinterface`," "`ftpclient`" and "`ftpserver`" into your Eclipse workspace. The Eclipse workspace just contains metadata for the projects, which remain where you originally stored them (unless you chose to copy them into the workspace).

There are two profiles defined in our root POM file. The "`local`" profile is for generating the server jar file for testing it on your local machine. In Eclipse, choose "`File | Properties | Maven`" and set the profile to be "local." Do this for "`ftp`", "`ftpserver`" and "`ftpclient`." Then "`Run as… | Maven install`" to generate the jar file for "`ftp`." If you are using the command line interface for Maven, you will type something like: "`mvn -Dlocal install`" in the `ftp` directory.

Once you have your code running locally, you should test it on an EC2 instance. We will provide you with an AMI that you should use to launch your instances. Use the "`remote`" profile rather than the "`local`" profile, and this will generate the files (the shell scripts ftpd.sh and ftp.sh) assuming that the server will run on an EC2 instance. Copy the server jar file to the EC2 instance using "`scp`," using the "-i" option to define the private key file that you use to authenticate yourself to the instance. Then ssh to the instance and run the server script. It should be in a directory of the form

```
/home/ec2-user/tmp/cs549/ftp-test
```

[1] If you are working on a Windows machine, you will either have to install Cygwin to run these shell scripts, or you will have to write DOS batch files ftpd.bat and ftp.bat to perform similar functions. If you do write such batch files, please share them with the rest of the class. If you do not have Unis, consider installing git on your Windows machine. Git comes installed with a bash shell.

that you should have created, again with a subfolder called "`root`" for the test file system. Now you should be able to run the client on your home machine and connect to the server running on your EC2 instance. To do this, you will have to define a security group for the EC2 instance that allows access from your client machine to your EC2 instance. It is strongly recommended that you not simply disable the firewall for your EC2 instance. Instead allow access on all ports from your home machine to your EC2 instance. Another aspect of this is that, on EC2, the server will have to explicitly bind to the external IP address of the EC2 instance (assuming that the client is running outside the Amazon network). You will have to set this external IP address in the properties file associated with the server project.

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

- Export your Eclipse project to the file system.
- 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 client on your home machine connecting remotely to an EC2 server. Since you can only test in passive mode with the server running in EC2 (without endangering yourself), you should demonstrate testing the server in active mode only on your home or lab machine.
- Also include in the directory a report of your submission. This report should be in PDF format. **Do not provide a Word document.**

The content of the report is very important. A missing or sloppy report will hurt your final grade, even if you get everything working. In this report, describe how you completed the code for server and client so the client can bind to the server, and file transfers are enabled. You should include critical code snippets with explanation. You should also describe how you tested the code. Again, 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 both text and binary files.
4. Your test documentation should include a video demonstrating a working solution, with an explanation in your documentation of what the video is demonstrating.

Finally note any other changes you made to the materials provided, with an explanation.

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