School of Computing, University of Leeds

COMP2221
Networks

Coursework 2: Using sockets to transfer files between client and server

Deadline: 10am, Monday March 25th.

If you have any queries about this coursework, visit the Yammer page for this module (`COMP2221 1819`). If your query is not resolved by previous answers, post a new message. Caitlin Howarth will also be available to help during the timetabled lab sessions except for the week commencing March 11th.

Learning objectives

- Implement client and server file transfer applications.
- Design of a communication protocol to meet the specified requirements.
- Implement and test the use of sockets for data transfer over a network.

This piece of work is worth **25%** of the final module grade.

Task

Write two applications, a server and a client, that transfer files between each other using sockets.

The **server** application should:

- Run continuously.
- Use an `Executor` to manage a fixed thread-pool with 10 connections.
- Following a request by a client, query the local folder `serverFiles` and return a list of the files found there to the same client.
- Send a file from `serverFiles` to a client that requests it.
- Read a file from a client and place it in the `serverFiles` folder.
- Log every request by a client in a local file `log.txt` with the format: `date:time:client IP address:request`.

The **client** application should:

- Accept one of the follow commands as command line arguments, and performs the stated task:
  - `list`, which lists all of the files on the server's folder `serverFiles`.
  - `get fname`, which requests the server send the file `fname`. This should then be read and saved to the client's local folder `clientFiles`.

  - `put fname`, which sends the file `fname` from the client's local folder `clientFiles` and sends it to the server (to be placed in `serverFiles`).
- Exits after completing each command.

The listening port should be 8888. Your solution should work in principle for any files, not just those provided.

The communication protocol between the server and its clients is not specified. You are free to devise any protocol you wish, provided the above requirements are met.

For the purposes of this coursework both the client and the server should run on the same machine, *i.e.* with hostname `localhost`. They should not attempt to access each other's disk space; all communication must be *via* sockets. UNIX filenames should be used.

Both applications should have basic error handling and your code should adhere to the Java coding standards described in `JavaCodingStandards.pdf` on Minerva (the same as Coursework 1).

Guidance

Download the file `cwk2.tar.gz` from Minerva and unarchive it using `tar xzf cwk2.tar.gz`. You should now have a directory `cwk2` containing the following files and subdirectories:

```
cwk2 --- client --- Client.java
      |          |
      |          -- clientFiles --- potHoles.jpg
      |
      -- server --- Server.java
                 |
                 -- log.txt
                 |
                 -- serverFiles --- SWJTU.jpg
                                 |
                                 -- lipsum1.txt
                                 |
                                 -- lipsum2.txt
```

This provides you with both of the local storage directories `clientFiles` and `serverFiles` containing some example files for you to transfer. Empty class files for the client and server have also been provided. **Do not change the names of these files**, as we will assume these file and class names when assessing (see below). You are free to add additional `.java` files to the `client` and `server` directories but should not create extra directories.

You may like to first develop `Client.java` and `Server.java` to provide minimal functionality, following the examples covered in Lectures 7 and 8. You may like to add another class that handles the communication with a single client. This will make it easier to implement the multi-threaded server using the `Executor`. You can then add functionality one item at a time, *i.e.* each of the `list`, `get` and `put` options. The logging can be left until the end.

Input and output streams, and how to chain them, were covered in Lecture 6. It is not recommended that you chain multiple times from the same stream (*e.g.* chain two output streams from `socket.getOutputStream()`), as this can cause ill-defined and non-reproducible behaviour. Depending on your choice of stream. you may need to manually `flush()` when sending data.

Marks

This coursework will be marked out of 25.

5 marks	:	Basic operation of the `Server` application, including use of thread pool and log file output.
4 marks	:	Correct implementation of the `list` command.
9 marks	:	Correct implementation of the `put` and `get` commands.
7 marks	:	Good structure and commenting. Basic error handling. Adherence to the Java coding standard provided.
Total: 25		

Submission

Ensure that the `clientFiles` directory only contains the file `potHoles.jpg`, and `serverFiles` only contains the files `SWJTU.jpg`, `lipsum1.txt` and `lipsum2.txt`. Delete all extraneous files (*e.g.* any IDE-related files). You should then archive your submission as follows:

1. `cd` to the `cwk2` directory
2. Type `cd ..`
3. Type `tar czf cwk2.tar.gz cwk2/*`

This creates the file `cwk2.tar.gz` that you should submit *via* Minerva.

The following sequence of steps will be performed when we assess your submission.

1. Unarchive the `.tar.gz` file.
2. `cd` to `cwk2/client` directory and compile all Java files: `javac *.java`
3. `cd` to `cwk2/server` directory and do the same.
4. To launch the server, `cd` to the `cwk2/server` directory and type `java Server`
5. To launch a client, `cd` to the `cwk2/client` directory and type `java Client ...`
6. Multiple clients will be launched.

Your submission must work when this sequence is followed.

Disclaimer

This is intended as an individual piece of work and, while discussion of the work is encouraged, what you submit should be entirely your own work. Code similarity tools will be used to check for collusion, and online source code sites will be checked.

The standard late penalty of 5% per day applies for work submitted after the deadline.