Project Logistics – Version 1

Due Dates

Midpoint check: Oct. 21, 11:59 pm
Final Project: Dec. 7, 11:59 pm
Late Policy: No late project will be accepted.

The project is fairly large (2000 lines of code or more). Please start early.

There will be a midpoint check. You are supposed to implement about half of the lines for your project (800 lines or more), which can pass compilation (see suggestions later). This is worth 5% of your course grade.

Group Size

- Three students per group. Please form your project group ASAP.
- It will be great if you find your own partners. If you need help to find project partners, please follow the TA's announced a process.

Project Submission

- Submit on Canvas for both the midpoint check and the final project.
- Combine all your files into one archive and/or compressed file with winzip on Windows PC or with the tar command on UNIX/Linux machines. An example:

```
tar cvf proj1.tar foo.java bar.java
```

Here, `proj1.tar` is the archive file, and `foo.java` and `bar.java` are your source files.

- The zip or tar file should contain all source files, all files needed to compile your program, and all files needed to run your program. Your zip file should also contain files for starting your remote processes, even if you use the files from the course web site without changing anything. Your zip file should not contain executable files or object files which can be generated by compiling your program. It should not contain Common.cfg, PeerInfo.cfg, and any other sample files for testing.

Demo

- After the submission, each group will show a demo of their project. This will take the form of 5-8 minutes recorded video. You can submit the video demo on Canvas if the file size is not too large. Otherwise, you can store the video on onedirve@uf and share a link in your submission.
- For the demo, the data file to be exchanged should be at least 20Mbytes. The piece size will be 16384 bytes. The required free disk size depends on how you handle partial temporary files. You may need more than 150Mbytes. Please make sure that you have enough free disk space for your program to distribute a 20 Mbyte file from a peer to five other peers.

- For the demo, you should use the Linux instructional machines at CISE (remote login is fine). Please note that the machines tend to get very busy around the project due date. Please don't wait till the last minute.
- To get a high score, it is very important to meet the requirements described in the implementation specifics. Writing logs is also important.
- More details will come later.

Questions

- If you have difficulty understanding the protocol description, you may search the BitTorrent protocol description on the Internet and study it. There are many sources of information that explain the protocol. Then, you can come back and read the project description again.

Grading

Each project group will receive a common grade first. After that, there may be an upward or downward adjustment for each group member depending on the amount of effort spent.

Partial Credits for Project

If your group can't get the project to work, please document what you have done and who has done what in the readme file and explain your situation during the demo. This helps each of you to receive partial credits based on your code.

Sample Files

On Canvas, under the direction '`Files/Project`', you can find three sample files related to the project.

`Sample Client.java`, `Sample Server.java`: These are self-explanatory.

`StartRemotePeers.zip`: Code for starting multiple peers automatically. It is not essential. The code used to work in the CISE environment. But, it is now known to have problems due to SSH. If you can get it to work, it can be helpful (see also the next file). Otherwise, you can always start peers manually. See the project description for explanation.

`startpeers-from-students.zip`: Code and instruction for getting around the SSH problem. This was discovered by some student group.

`project_config_file_small.zip, project_config_file_large.zip`: Two sets of sample configuration files and data file to be exchanged. One set has a small data file, around 2 MB; the other has a large data file, more than 20MB. You can play with both scenarios. To demonstrate choking, unchoking, and optimistic unchoking, the data file should be at least 20MB, or perhaps larger.

References

There are many online resources for JAVA and network programming. Feel free to use those resources. You might have to browse many and select the ones that are suitable for you.

The following book is very good.

Elliotte Rusty Harold. Java Network Programming, 4th edition. 2013. O'Reilly Media, Inc.

Some chapters of the 3rd edition can be found online. See here.

Academic honesty

You should not have someone else to write your code or copy the code from someone else. But, you are allowed to use the Internet resources and learn how to do network programming in general. Anyone who violates the above rule will get penalized. Please note that there are tools for us to check the similarity between two codes.

FAQ

Q1. Can I implement the project on Windows environment?

You may. But, in the end, you must be able to demonstrate your project on 5 different instructional Linux machines.

Q2. Can I assume that the subdirectories for peers have already been set up before I run the peer processes?

Yes. You may create subdirectories with proper names before you run your peer processes. That saves you the burden for managing subdirectories in your program. See the relevant discussion in the project description document.

Q3. How many TCP connections should be used between two peers?

The design may vary. For instance, you may use one TCP connection for each pair of peers. Consequently, a peer maintains only one socket for each neighboring peer. However, you may have two threads associated to the socket, one for getting messages from the neighboring peer and the other for sending messages to the peer. The socket will be shared by those two threads.

Q5. How is the 'have' message different from the 'bitfield' message?

The 'bitfield' message is useful to inform a neighboring peer the availability of many file chunks. You can use it at the start of a process. The 'have' message is used to inform a neighbor the availability of a single chunk. It is useful for incremental update between the neighboring peers. For instance, when a peer receives a new chunk, it can use the 'have' message to inform other neighbors.

Q6. How can I submit the final project?

Please read the section "Project Submission" carefully.

Q7. What should I submit for midpoint check?

You are asked to submit 800 lines or more of compiled code. That’s all. The objective is to make sure you have started working on it.

Q8. Should each project group submit only one copy of the code?

Yes. Only one of your teammates needs to submit the project (midpoint or final) on Canvas. Please list your group members in a readme file.

Q9. The project looks complicated. How should I start?

I recommend you first read the project description document CAREFULLY. It describes a protocol. The whole project may look complicated, but fundamentally, each peer just sends, receives, reads, replies messages through TCP connections based on the protocol described in that document. After several reading, you will have a broad picture of the project in your head.

Q10. Can I use my own machine to run the project code?

During development, it is easier to work on your local machine. You can run several peer processes on your machine. For the demo, you are asked to run your code on different Linux instructional machines in the CISE labs. You should leave enough time before your demo to make sure your code works on the CISE machines.

Q11. Which CISE Linux machines can I use? How to I log into those machines?

See page 8 of the project description document.

Q12. How do I log into CISE Linux machines?

See page 8 of the project description document.