Instruction of Project Submission and Demo

Due Dates

Midpoint check: 11/02/2020

Final Project：12/09/2020

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 Info

- Please check the group information through CANVAS->PEOPLE->GROUP

Project Submission

- Submit on E-Learning 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 & Video

- You will record a 20-30minute video to show your project after the submission. You should upload the video to onedrive@UF. Make sure you give access right to view your video.
- When uploading your source files on Canvas, please include a readme file, which should contain an URL for your video file.
- The video file should be dated no later than the project submission deadline.
- After projection, your group will be given the zip file that you already submit before the demo. You will be also given Common.cfg file, PeerInfo.cfg file, and a sample file for distribution. You will be asked to compile your program and to get ready for running it. Please make sure that it is your responsibility to set up the environment and demonstrate your program.

- For the demo, you should have at least 120Mbytes free disk space. We will use a sample file of about 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 120Mbytes. Please make sure that you have enough free disk space for your program to distribute a 20 Mbyte file from a peer to four other peers.
- It is very important to meet the requirements described in the implementation specifics for getting high points. In particular, writing logs is very crucial in this project.
- More details may 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

Since we grade your project based on your video, you are expected to show your program meet the following rubric:

0. Midpoint check: 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). 5%
1. Start the peer processes: 20%
    a) In your video, you should clearly show that your program will read the Common.cfg to correctly set the related variables. 5%
    b) In your video, you should clearly show that your program will read the PeerInfo.cfg to correctly and set the bitfield. 5%
    c) In your video, you should clearly show that your program will let each peer make TCP connects to all peers that started before it. 5%
    d) When a peer is connected to at least one other peer, it starts to exchange pieces as described in the protocol description section. A peer terminates when it finds out that all the peers, not just itself, have downloaded the complete file. 5%
2. After connection: 25%
    a) Handshake message: Whenever a connection is established between two peers, each of the peers of the connection sends to the other one the handshake message before sending other messages. 5%
    b) Exchange bitfield message. 5%
    c) Send ‘interested’ or ‘not interested’ message. 5%
    d) CORRECTLY send *k* ‘unchoke’ and ‘choke’ messages every *p* seconds. 5%
    e) Set optimistically unchoked neighbor every ‘*m*’ seconds. 5%
3. File exchange: 30%
    a) Send ‘request’ message. 5%
    b) Send ‘have’ message. 5%
    c) Send ‘not interested’ message. 5%
    d) Send ‘interested’ message. 5%
    e) Send ‘piece’ message. 5%
    f) Receive ‘have’ message and update related bitfield. 5%
4. Stop service correctly. 5%
5. Fully correct. 15%

Please note that the best way to let us understand your program is to print a clear log. During the video, you need to SHOW and EXPLAIN to us that your program meets all these requirements ONE BY ONE. As an example, your program may produce the following log to show that your program fits all the requirements in ‘Start the peer processes’ section, so there is no doubt at all.

P1:
At t1, P1 Start, set variables to xxx, bitfield to xxx.
At t3, a TCP connection is built between P1 and P2
At t5, a TCP connection is built between P1 and P3

P2:
At t2, P2 Start, set variables to xxx, bitfield to xxx.
At t3, a TCP connection is built between P1 and P2
At t6, a TCP connection is built between P2 and P3

P3:
At t4, P3 Start, set variables to xxx, bitfield to xxx.
At t5, a TCP connection is built between P1 and P3
At t6, a TCP connection is built between P2 and P3

If your program can’t work fully correctly, you need to show us what requirements your program can meet and what requirements your program can’t, so we can give you as much grade as possible.

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.

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 machines at the same time.

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. But, consider using 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.

Q4. How can I submit the final project and how will the demo be held?

Please read the section "Project Submission" carefully.