DS A1 Report

Jialiang Cheng 1251403

Introduce

Multi-threaded Dictionary Server was the first small project I completed in Java. Before this, I mostly used C ++ for programming during my undergraduate period. In the process of completing this project, I not only learned simple distribution system construction, but also gained a lot of Java programming knowledge. I have to say, I benefited a lot. Thank you guys.

Problem

Without a doubt, the biggest problem for me is probably programming in Java. Compared with C++, I think Java is a more flexible language. On the whole, it does not have the rigorous architecture of C++, but it provides more and more flexible implementation methods. It gives me a much better programming experience than C++.

Besides, the threads brought some difficulties in understanding and using them, but I finally solved it by giving a new thread for each request - this greatly simplifies the threading problem and saves me a lot of time.

The GUI is also a problem, because I have no GUI experience with Java. But the tut code has been a huge help to me, and my client-side GUI is based on TUT code as well. Of course, I also made some improvements to make the GUI better suited to a dictionary program's needs.

About the system

The system consists of two jar files, which present the server side and client side. This system design is based on the tut demo code.

Server

Usage: java –jar DictionaryServer.jar <port> <dictionary-file>

The DictionaryServer.jar requires 2 arguments – the port and the initial dictionary file path. It will print the error if there were something wrong with the argument and terminate the server. If everything is working correctly, the server will print “The dictionary server has

started...” and wait for client to connect. Notice that server does not have a GUI since I thought it is unnecessary. The server handles each request by a new thread, so it support multi-thread and a same time.

Client

Usage: java –jar DictionaryClient.jar <server-address> <server-port>

The DictionaryClient.jar requires 2 arguments – the server address and the server port. The client has an GUI. On the top of the interface, I set a menu bar with necessary information: “help for how to use the dictionary and “about” for information about me. Under the menu bar, there is a text filed for user type the command, and submit the request to server through the buttons on the right side. The feedback from server will be displayed on the text area. For each command, client will pass request to server by a new socket – that is, why the server handles each request by a new thread. Also, this guarantee that access from different clients does not conflict. If there is anything wrong, the error will print to the text area and the client will not terminate. All the GUI design is inspired by the tut demo code. And my goal for the GUI is simply and easy to understand and use.

New feature

Save

I add the save button on the client GUI. By pressing the save button, the server will overwrite the current dictionary to the initial dictionary path. So when next time the server starts, the change of this time will be preserved.

Rethink

To be honest, I have to admit that the overall design of the system is too simplistic. I did not have much knowledge about java when I just began the project. The whole code framework is built based on the demo code provided by tut. Luckily, the general functionality has been successfully implemented.

Anyway, I'm satisfied with the Java experiment and hope I can do it better next time.