Java Code of File Transfer Using UDP

Project reference Java code
Summer, 2022
CSE6324 concurrent programming

The given Java code is an implementation of file transfer. It uses TCP to send commands/requests and UDP to transfer file data. The file data transmission is a solution based on monitor, which is an important concept in this course. The monitor is a Java built-in monitor.

This piece of code is given as a reference. It can give you some idea about Java programming on how to transfer data between two hosts, and most importantly, how the monitor concept is applied to a real-world case.

Please note that the code is tested properly without error. You can reuse it as the way you prefer. But you use it at your own risk.

Thread Design

File transfer is built in a client-server model. The client is the sender who wants to send a file to the server. The server is the receiver who will receive the file. Before it sends the file data to the server through UDP, the client first tells the server the UDP port number that it will use to send data and then gets the UDP port number that the server will receive data. Therefore, the client sends a request to the server through TCP that contains the UDP port number it will use. The server sends back a response which contains the UDP port number it will use to receive data. Once the client and the server know the UDP port number that the other will use, file data transfer can begin.

Below is the thread design for file transfer from the client to the server.

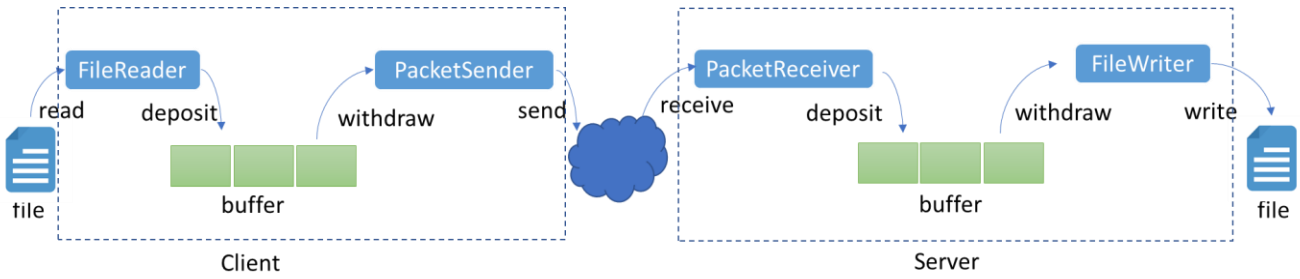

Figure 1 Thread Design of File Transfer

There are four threads, denoted by blue round boxes. Two threads are at the client side and the other is at the server side. The reason to have four threads is that the file resource and the network resource are different, thus pipelining the two resources can increase resource utilization, one benefit of concurrent programming.

At the client side:

> **FileReader** repeatedly reads packet-size data from the file and deposits them to the buffer until all the data are read.

PacketSender repeatedly withdraws packet-size data from the buffer and sends them to the server.

At the server side:

PacketReceiver receives packets and drops them to the buffer properly.
FileWriter withdraws packets from the buffer and writes to a file.

To make sure reliable data transfer, ACK (acknowledgement) packets are sent. PacketReceiver sends back an ACK packet for each packet it receives. PacketSender does not advance to send the next packet until the ACK packet of the current pending packet is received. This is the basic design to ensure reliable data transfer on UDP.

File Structure

The following shows the hierarchical structure of files in this reference code

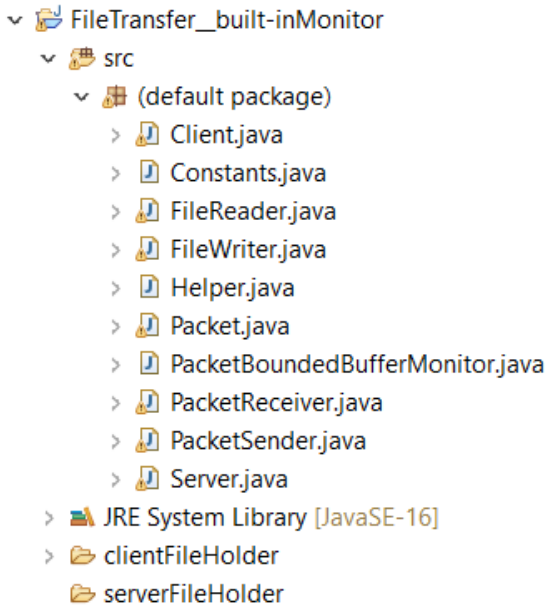

Figure 2 Hierarchical View of Files in Eclipse IDE

FileTransfer__built-inMonitor the project name, within which all java files and relevant folders are placed.

PacketBoundedBufferMonitor.java: it implements a bounded buffer monitor. The order of the data blocks deposited is the same as the order of data blocks withdrawn.

Client.java: the sender who wants to send a file. It starts two threads FileReader and PackeSender.

Server.java: the receiver who will receive data by running two threads: PacketReceiver and FileWriter.

Packet.java: define a class for datagram packet.

FileReader.java: the thread that reads data from a file and puts them to the buffer.

FileWriter.java: the thread that pulls data from the buffer and writes them to a file.

PacketSender.java: the thread that get packets from a buffer and sends them to the receiver.

PacketReceiver.java: the thread that receives packets and put them to a buffer.

Helper.java: a file that has a collection of common and useful functions.

Constants.java: a place to hold all constants that will be used in this code.

clientFileHolder and **serverFileHolder** are two directories to hold files for the client and the server respectively.

How to Run

Assume that you have set up the files in your project as shown in Figure 2 in Eclipse IDE.

Modify the folder paths in Constants.java according to the exact locations in your machine.

First open the Server.java and click the run icon to start the server.

Then open the Client.java and click the run icon to start the client.

Now, the client sends the file video_small.wmv to the server. So, in the serverFileHolder directory, you can this video file.