컴퓨터네트워크

Socket Programming : TCP File Transfer

컴퓨터전공

2013011695 정태화

1. 목표

Socket Programming을 이용하여 TCP 통신 프로그램을 구현해본다. 이 프로젝트에서는 TCP를 이용한 파일 전송 프로그램을 구현하였다.

2. 개발 환경

OS : Mac OS X

개발 IDE : IntelliJ IDEA

사용 언어 : JAVA

3. 코드 설명

<Client>

```
import java.io.*;
import java.net.InetAddress;
import java.net.ServerSocket;
import java.net.Socket;
import java.net.SocketException;
import java.util.Scanner;

public class TCP_client {
    public static void main(String[] args) {
        Socket socket= null;
        InputStream input = null;
        OutputStream output = null;
        DataInputStream dis = null;
        DataOutputStream dos = null;
        FileInputStream fis = null;

        Scanner sc = new Scanner(System.in);

        // 클라이언트 본인의 주소를 출력한다.
        try {
            InetAddress a = InetAddress.getLocalHost();
            System.out.println(a.getHostAddress());
        }
        catch(Exception e){

        }

        // 연결할 서버의 주소와 포트를 입력한다
        System.out.println("[Client Mode]");
        System.out.println("Please input IP address & port number");
        System.out.print("IP : ");
        String ip = sc.nextLine();
        System.out.print("Port : ");
        String port = sc.nextLine();
        int port2 = Integer.parseInt(port);
```

```
        // 소켓을 생성한 후 데이터를 보낼 outputstream을 연다
        try {
            socket = new Socket(ip, port2);
            //데이터를 통신을 위해서 소켓의 스트림 얻기.
            input = socket.getInputStream();
            dis = new DataInputStream(input);

            output = socket.getOutputStream();
            dos = new DataOutputStream(output);
            System.out.println("Server Connected");
        } catch (Exception e){
//            e.printStackTrace();
            System.out.println("Connection Error!! Shutting down.....");
            return;
        }

        try {
            // 파일의 경로를 가져온다
            File path = new File( pathname: "");
            System.out.println(path.getAbsolutePath());

            System.out.println("보내고 싶은 파일의 이름을 입력해주세요!");
            String sendfile = sc.nextLine();

            // 가져온 파일의 절대경로를 이용하여 존재하는 파일인지 확인 후
            // 없을 경우 에러를 출력한다.
            // 그렇지 않으면 내용없이 이름을 가진 객체만이 전송되어 서버측에서 빈 파일이 생긴다
            path = new File(sendfile);
            System.out.println(path.getAbsolutePath());
            System.out.println(path.exists());

            // 파일 이름을 보낸 후
            // 1메가 단위로 버퍼를 끊어 데이터를 전송한다.
            // data를 버퍼크기 단위로 받아와 전송하는 DataOutputStream에 write하며
            // EOF인 -1을 리턴할때까지 파일 데이터를 받아 생성한 객체에 write 후
            // EOF를 만나면 종료 후 스트림을 닫는다
            if(path.exists()) {
                dos.writeUTF(sendfile);
                System.out.println("Sending File : " + sendfile);

                fis = new FileInputStream(path.getAbsolutePath());
                BufferedInputStream bis = new BufferedInputStream(fis);

                byte[] buf = new byte[1024];

                while (true) {
                    int data = bis.read(buf);
                    if (data == -1)
                        break;
                    dos.write(buf,  off: 0, data);
                    dos.flush();
                }

                bis.close();
                fis.close();
            } else
                throw new FileNotFoundException();

        } catch (IOException e){
//            e.printStackTrace();
            System.out.println("File Error!!");
            return;
        }

        System.out.println("File Transfer complete!!");

        // 나머지 스트림과 소켓 close

        try {
            dos.close();
            output.close();
            socket.close();
        } catch (IOException e){
            e.printStackTrace();
        }
    }
}
```

클라이언트는 실행 시 접속할 서버의 ip주소와 포트 번호를 입력받는다. 그 후 맞는 주소일 경우 연결이 되며, 연결된 후 보낼 파일의 이름을 입력한다. 보낼 파일은 현재 컴파일 경로와 같은 위치에 존재해야하며, 존재하는지 여부를 exist() 함수를 이용하여 체크 후 TRUE일 경우에 전송을 시작한다. 전송이 끝난 경우에 스트림을 닫고 연결을 종료한다.

<Server>

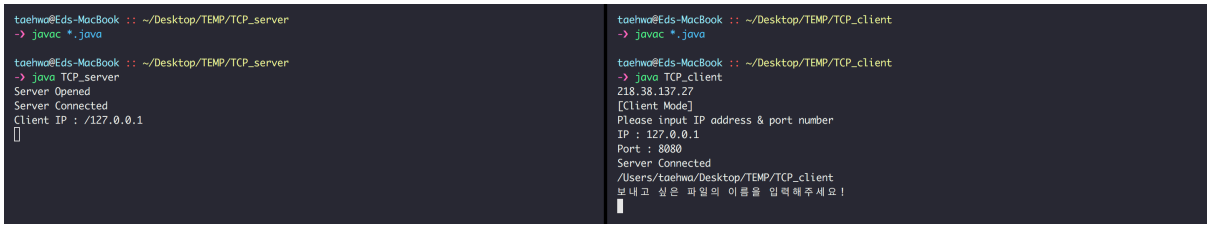
```java
public class TCP_server {
    public static void main(String[] args) {

        ServerSocket server = null;
        Socket socket = null;
        InputStream input = null;
        OutputStream output = null;
        DataInputStream dis = null;
        DataOutputStream dos = null;
        FileOutputStream fos = null;

        // 서버 파일이므로 서버소켓을 생성하고, 클라이언트에게서 accept를 받을 소켓을 생성한다
        // 그리고 클라이언트에게서 데이터를 전송받을 DataInputStream을 열어준다.
        try {
            server = new ServerSocket( port: 8080);
            System.out.println("Server Opened");
            socket = server.accept(); //클라이언트접속 대기
            System.out.println("Server Connected");
            // 접속한 클라이언트의 ip를 출력한다
            System.out.println("Client IP : " + socket.getInetAddress());
            input = socket.getInputStream();
            dis = new DataInputStream(input);

            output = socket.getOutputStream();
            dos = new DataOutputStream(output);
        } catch (Exception e) {
            e.printStackTrace();
        }
```

```java
        // 먼저 파일명을 받아온 후, 받은 파일명으로 FileOutputStream을 통해 서버쪽에 객체를 생성한다
        // 그 후 BufferOutputStream으로 확장한 후
        // data를 버퍼크기 단위로 받아와 File에 write를 한다
        // EOF인 -1을 리턴할때까지 파일 데이터를 받아 생성한 객체에 write 후
        // EOF를 만나면 종료 후 스트림을 닫는다
        try {
            byte[] buf = new byte[1024];

            String rcvfile = dis.readUTF(); //클라이언트로부터 파일명얻기
            System.out.println("Receiving File : " + rcvfile);

            fos = new FileOutputStream(rcvfile);
            BufferedOutputStream bos = new BufferedOutputStream(fos);

            while (true) {
                int data = dis.read(buf);
                if (data == -1)
                    break;
                bos.write(buf, off: 0,data);
            }
            bos.flush();
            bos.close();
            fos.close();
        }catch (IOException e){
//            e.printStackTrace();
            System.out.println("Connection lost!!");
        }

        System.out.println("File Transfer Complete!!");

        // 나머지 스트림과 소켓 close
        try {
            dis.close();
            socket.close();
            server.close();
        } catch (IOException e) {
            e.printStackTrace();

        }
    }
}
```

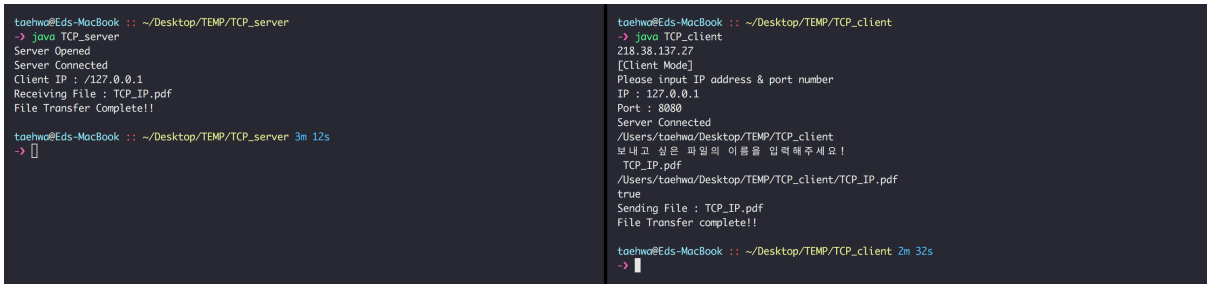
서버는 실행된 후 클라이언트의 접속을 기다리고 있다. 포트는 8080으로 설정해놓았다. 클라이언트가 접속할 경우 스트림이 연결되며 클라이언트의 ip를 출력하고 클라이언트가 보낸 파일을 버퍼단위로 끊어서 받는다. 모든 파일을 받을 경우 스트림을 닫고 연결이 종료된다.

4. 실행 방법 및 결과

첨부한 TCP_server 폴더와 TCP_client 폴더 안에 소스파일이 들어있다. 각각 폴더로 들어가 javac *.java 를 입력하여 컴파일 후, 각각 java TCP_client, java TCP_server로 실행하였다.



client 폴더에 위치한 30MB짜리 컴네 전공 pdf를 한번 옮기기 시도해보았다. 파일명은 TCP_IP.pdf 이다.



파일 전송을 완료 후 서버와 클라이언트 모두 종료됨을 볼 수 있다.

직접 디렉토리에 들어가서 손상 없이 파일이 전송완료 되었는지 확인할 수 있었다.

5. 느낀 점

말로만 듣던 소켓 프로그래밍을 직접 해보니 생각보다 상당히 어려웠다. 그러나 다 끝마친 후에는 이론상으로만 듣던 프로토콜들의 통신과정을 직접 구현하게 되어 어떤 식으로 돌아가는지 이해할 수 있었다. 또한 생각보다 java에는 소켓을 위해 구현된 API가 많아서 어떤 것을 써야 좋을지 선택하는데도 오래 걸렸다. 하지만 한번 성공하고 나니 편리한 기능들이 매우 많음을 느꼈다.

시간이 된다면 한 서버에 여러 클라이언트가 접속 가능한 다중 접속 서버를 만들어 보려 했는데 도저히 다른 과목들이 여유 시간을 전혀 주지 않아서 아쉽다.