#Summary

P2P 방식의 스워밍 프로토콜을 완성하기 전 단계로 CS 방식의 파일 청크 다운로드 클라이언트 피어와 서버 피어를 구현한다. 클라이언트로 동작하는 피어와 서버로 동작하는 피어는 일대일로 TCP 연결을 맺고 클라이언트가 파일 청크를 일방적으로 다운로드한다.

server가 자신의 port number을 가지고 client의 접속을 대기하며, client는 server의 IP와 port number를 이용해 server에 접속한다. 전송 받을 파일을 파일 이름으로써 요구하며, 파일이 server에 존재할 경우 파일의 데이터들을 받아 received_[파일이름] 으로 저장한다.

파일은 10KB의 chunk로 나뉘어 전송되며, 남은 byte들은 해당 크기의 chunk로 마지막에 전송된다. 모든 파일 전송이 끝난 뒤 connection이 끊어지고, 프로그램이 종료된다.

#Client class

```
public static final int BUFFER_SIZE = 10240;
```

client는 server에게 파일을 요구하고, 전송받는다. 10KB chunk 단위로 데이터를 받으므로, `BUFFER_SIZE`를 10240로 선언한다.

```
String serverIP = args[0];
int port = Integer.parseInt(args[1]);
String file_name = args[2];
```

client 실행 시 command-line argument로 `serverIP`, `port` number, 요청할 파일 이름인 `file_name`을 입력받는다.

```
BufferedOutputStream bos =null;
DataInputStream dis = null;
DataOutputStream dos = null;
long fileSize = 0;
File copyFile = null;
```

BufferedOutputStream인 `bos`, DataInputStream인 `dis`, DataOutputStream인 `dos`, 받을 파일의 크기인 `fileSize`, 받은 파일의 데이터를 저장할 새로운 파일인 `copyFile`을 생성한다.

```
Socket socket = new Socket(serverIP, port);
dis = new DataInputStream(socket.getInputStream());
dos = new DataOutputStream(socket.getOutputStream());
dos.writeUTF(file_name);
```

`serverIP`와 `port number`을 인자로 갖는 새로운 Socket 오브젝트를 생성하고, `socket`의 inputstream과 outputstream을 각각 `dis` 와 `dos`에 할당한다. 커맨드라인으로 입력했던 `file_name`을 server에 전송한다. 이는 server에서 해당 파일이 존재하는 지 미리 확인하기 위함이다.

```
if (dis.read()==0) {
    System.out.println("File not exist, exit connection");
    System.exit(0);
}
else System.out.println("Ready to receive file ["+ file_name+"]");
```

dis.read( )는 server측에서 전송한 `file_name`에 해당하는 파일의 존재 여부를 읽는다. dis.read( )가 0이라면, 요청했던 파일이 server에 존재하지 않는다는 뜻이므로 "File not exist, exit connection" 메시지를 출력 한 뒤 프로그램을 종료한다. dis.read( )가 0이 아니라면, 해당 파일이 존재한다는 뜻이므로 "Ready to receive file ~"을 출력한다.

```
copyFile = new File("received_"+file_name);
bos = new BufferedOutputStream(new FileOutputStream(copyFile,false));
fileSize = dis.readLong();
int totalChunk = (int) (fileSize/BUFFER_SIZE);
int lastBytes = (int) (fileSize%BUFFER_SIZE);
int readBytes;
long BytesCnt = 0;
byte[] buffer = new byte[BUFFER_SIZE];

System.out.println("Received bytes will be stored in ["+copyFile+"]");
System.out.println("File size of ["+file_name+"] is "+fileSize);
System.out.println("Total "+totalChunks+" 10KB chunks and one "+ lastBytes+" bytes chunk");
```

server에 파일이 존재함을 확인한 뒤 다음 작업들을 수행한다.

`copyFile`을 "received_`file_name`"라는 이름으로 생성하고, `copyFile`을 인자로 하여 버퍼 스트림과 파일 스트림을 연결한다. 그리고 파일 크기를 server로부터 받아 `fileSize`에 저장한다. 받아야 할 10KB의 chunk 수를 저장하는 `totalChunks`, 파일 크기에서 10KB chunk들을 제외하고 남은 크기를 저장하는 `lastBytes`, 현재 읽어들인 크기를 저장하는 `readBytes`, 읽은 byte 수를 누적하여 저장하는 `BytesCnt`, 10KB 크기의 `buffer` 배열을 선언하고, `copyFile`, `file_name`, `fileSize`, `totalChunk`와 `lastBytes`를 출력한다.

```
for (int i=0;i<totalChunks;i++) {
    if ((readBytes = dis.read(buffer, 0, BUFFER_SIZE))!=-1) {
        bos.write(buffer, 0, readBytes);
        BytesCnt+=readBytes;
        System.out.println("In progress... "+ BytesCnt*100/fileSize + "% received (total
"+BytesCnt+" bytes)");
    }
}
if (lastBytes>0) {
    if ((readBytes=dis.read(buffer, 0, BUFFER_SIZE))!=-1) {
        bos.write(buffer, 0, readBytes);
        BytesCnt+=readBytes;
        System.out.println("In progress... "+ BytesCnt*100/fileSize + "% received (total
"+BytesCnt+" bytes)");
    }
}
```

위 코드는 server로부터 데이터를 받고, 그것을 `copyFile`에 저장한다.

먼저, 반복문을 통해 10KB의 chunks을 받는다. `BUFFER_SIZE`만큼의 byte를 server로부터 성공적으로 받는다면, `copyFile`에 write한다. 현재 읽어들인 `readBytes`를 `BytesCnt`에 더하고, 진행상황을 출력한다. `lastBytes`가 존재하는 지 확인하여 위의 작업과 동일하게 수행한다.

```
dos.writeLong(BytesCnt);
System.out.println("Successfully received ["+file_name+"] as file
["+copyFile.getName()+"]");
```

성공적으로 받은 데이터의 Byte 수인 `BytesCnt`를 server로 전송하고, 그것을 메시지로 출력한다.

이하 코드는 예외를 처리하고, stream을 close한다.

#Server class

```
public static final int BUFFER_SIZE = 10240;
```

server는 client가 요구하는 파일을 전송한다. 10KB chunk 단위로 데이터를 전송하므로, `BUFFER_SIZE`를 10240로 선언한다.

```
int port = Integer.parseInt(args[0]);
```

server 실행 시 command-line argument로 `port` number를 입력받는다.

```
BufferedInputStream bis = null;
Socket socket = null;
ServerSocket server = null;
DataInputStream dis = null;
DataOutputStream dos = null;
String file_name = null;
File file = null;
```

BufferedInputStream인 `bis`, Socket 오브젝트인 `socket`, ServerSocket 오브젝트인 `server`, DataInputStream인 `dis`, DataOutputStream인 `dos`, 보낼 파일의 이름인 `file_name`, 파일 이름에 해당하는 실제 파일을 저장하는 `file`을 선언한다.

```
server = new ServerSocket(port);

System.out.println("This is port num "+port+", "+"ready to connected");
socket = server.accept();
System.out.println("Client Accepted");

dis = new DataInputStream(socket.getInputStream());
dos = new DataOutputStream(socket.getOutputStream());

file_name = dis.readUTF();
file = new File(file_name);
```

`port`를 인자로 하는 ServerSocket을 생성하여 `server`에 할당하고, 해당 `port`가 ready to connected 상태라는 메시지를 출력한다. client의 접속을 대기하고 있다가, client가 접속하면 accept하여 해당 socket을 `socket`에 할당하고 "Client Accepted" 메시지를 출력한다. 그리고 `socket` 의 inputstream과 outputstream을 각각 `dis` 와 `dos`에 할당한다. client와 connect 된 후 client가 요청한 파일의 이름을 받아 `file_name`에 저장하고, `file_name`에 해당하는 실제 파일을 `file`에 할당한다.

```
if (!file.exists()) {
    System.out.println("File ["+file_name+"] not Exist");
    dos.write(0);
    System.exit(0);
}
else dos.write(1);
System.out.println("File ["+file_name +"] ready for transmission");
```

전송해야할 `file`이 존재하지 않는다면, "File not Exist" 메시지를 출력한 뒤 client에 파일이 존재하지 않는다는 의미의 0을 전송하고, 프로그램을 종료한다. `file`이 존재한다면 client에 1을 전송하고, "ready for transmission" 메시지를 출력한다.

```
long fileSize = file.length();
int totalChunk = (int)fileSize/BUFFER_SIZE;
int lastBytes = (int)fileSize%BUFFER_SIZE;
long readBytes=0;
long totalTransmitted_Bytes = 0;
byte[] buffer = new byte[BUFFER_SIZE];
long completeBytes;

dos.writeLong(fileSize);
bis = new BufferedInputStream(new FileInputStream(file));

System.out.println("File size of ["+file_name+"] is "+fileSize);
System.out.println("Total "+totalChunk+" 10KB chuncks and one "+ lastBytes+" bytes chunk");
```

server에 파일이 존재함을 확인한 뒤 다음 작업을 수행한다.

파일 크기를 저장하는 `fileSize`, 보내야 할 10KB의 chunk의 수를 저장하는 `totalChunks`, 파일 크기에서 10KB chunk들을 제외하고 남은 크기를 저장하는 `lastBytes`, 현재 읽어들인 크기를 저장하는 `readBytes`, 읽은 byte 수를 누적하여 저장하는 `totalTransmitted_Bytes`, 10KB 크기의 `buffer` 배열, client로부터 전송 완료된 byte수를 받아 저장하는 completeBytes를 선언하고 `file_name`, `fileSize`, `totalChunk` 와 `lastBytes`를 출력한다. 그리고 `fileSize`를 client에게 미리 전송하고 `file`을 인자로 하여 버퍼 스트림과 파일 스트림을 연결한다.

```
for (int i=0;i<totalChunk;i++) {
    if ((readBytes=bis.read(buffer, 0, BUFFER_SIZE))!=-1){
        dos.write(buffer, 0, (int)readBytes);
        totalTransmitted_Bytes += readBytes;
        System.out.println("In progress... "+ totalTransmitted_Bytes*100/fileSize + "%
transmitted (total "+totalTransmitted_Bytes+" bytes)");
    }
}

if (lastBytes>0) {
```

```
    if ((readBytes = bis.read(buffer, 0, lastBytes))!=-1) {
        dos.write(buffer, 0, (int)readBytes);
        totalTransmitted_Bytes += readBytes;
        System.out.println("In progress... "+ totalTransmitted_Bytes*100/fileSize + "%
transmitted (total "+totalTransmitted_Bytes+" bytes)");
    }
}
```

위 코드는 file에서 데이터를 읽어 client에게 전송한다.

먼저, 반복문을 통해 10KB의 chunks을 `file`로부터 read한다. `BUFFER_SIZE`만큼의 byte를 성공적으로 읽는다면, client에게 전송한다. 현재 읽어들여 전송한 readBytes를 totalTransmitted_Bytes에 더하고, 진행상황을 출력한다. lastBytes가 존재하는 지 확인하여 위의 작업과 동일하게 수행한다.

```
System.out.println("File transmission completed");
completeBytes = dis.readLong();
System.out.println("Client successfully received "+completeBytes+" bytes of file");
```

모든 전송이 끝난 후 "File transmission completed" 메시지를 출력하고, client가 전송한 성공적으로 받은 byte 수를 읽어 completeBytes에 저장하고, 그것을 메시지로 출력한다.

이하 코드는 예외를 처리하고, stream을 close한다.

#how to Execute

2018008331_박민경 _assignment1.zip 파일을 다운받아 압축을 해제한다. 압축 파일 안에 있는 Server.java와 Client.java, 그리고 client에서 server에게 요구할 파일은 하나의 폴더에 위치시키도록 한다.

1. window에서 2개의 cmd창을 띄운다.
2. 두 창 모두 cd [.java 파일과 전송 파일이 존재하는 절대경로] 를 입력한다.

   ex) cd C:\Users\박민경\Documents\Eclipse-workspace\BitTorrent-like Swarming Protocol\src
3. 하나의 창에서는 javac Client.java, 다른 하나의 창에서는 javac Server.java로 각각을 컴파일한다. (편의 상 전자를 Client 창, 후자를 Server 창이라고 한다.)
4. Server 창에 java Server [port number] 을 입력한다.

   ex) java Server 8001
5. Client 창에 java Client [IP address] [port number] [파일이름] 을 입력한다. 이 때, port number은 Server 창에 입력했던 것과 같다.

   ex) java Client 127.0.0.1 8001 test.txt