1. 소스코드 실행방법

(아래 방법은 윈도우 환경을 기준으로 작성했습니다)

1. cmd에서 server\_code 파일에 들어간다.
2. javac -d bin src/module-info.java src/server/MainServer.java 를 입력한다.
3. bin 폴더에 class파일이 생성되면, java -cp bin server/MainServer 를 입력해 서버를 실행한다.
4. 새로운 cmd에서 client\_code 파일에 들어간다.
5. javac -d bin src/module-info.java src/client/MainClient.java 를 입력한다.
6. bin 폴더에 class파일이 생성되면, java -cp bin client/MainClient 를 입력해 클라이언트를 실행한다.
7. 클라이언트 – 서버 동작 절차
8. 연결 과정

서버: welcomeSocket을 통해 client의 접속을 받을 socket을 열고, 만약 해당 socket으로 연결이 일어나면 connectionSocket을 생성해 새로운 연결을 시작합니다.

클라이언트: serverIP와 control Portnum을 통해 server의 welcome socket과 통신하여 서버와 연결합니다.

1. command line 전달과정

클라이언트: 키보드로부터의 입력 스트림을 열어 사용자로부터 입력을 받습니다. 만약 사용자로부터의 입력 중 명령(order)에 해당하는 부분이 GET, PUT, CD, LIST, QUIT중 어느 하나에도 해당하지 않으면 오류 메세지를 출력하고 사용자로부터 다시 입력을 받습니다. 만약 order가 “QUIT”이라면 client를 종료합니다. 만약 order가 “PUT”이라면 사용자가 입력한 파일명이 존재하는지 확인하고, 존재한다면 해당 파일명과 파일 크기를 서버에 함께 전송합니다.

서버: 클라이언트로부터 command line을 전달받으면, 서버는 해당 command를 해석하는 과정을 거칩니다. 만약 “ “(spacebar 2회)가 존재한다면, \r\n으로 변경해 가독성을 올렸습니다. client로부터 메시지는 명령어에 해당하는 ORDER와 내용에 해당하는 VALUE로 나누어서 해석했습니다. ORDER와 VALUE는 ‘ ‘(spacebar 1회)로 구분되어 있으므로, 해당내용을 기준으로 ORDER와 VALUE를 분리했습니다. ORDER의 내용에 따라 서버의 response가 달라지게 구현하였습니다. 추가로 currenrDir라는 변수를 통해 현재 서버가 가리키고 있는 디렉토리를 관리하였습니다.

1. 서버에서 각 명령의 처리과정
2. “CD”를 입력: cd라는 클래스의 객체 CD를 생성하여 해당 명령을 처리하였습니다. cd클래스의 getPath() 메소드는 VALUE를 인자로 받아 client가 전달한 디렉토리 주소를 받습니다. 만약 VALUE가 상대경로라면, 절대경로로 변경합니다. 만약 존재하지 않는 경로이거나, 파일명이면 err를 출력합니다. 그렇지 않으면 해당 주소를 OUTPUT에 반환합니다.
3. “LIST”를 입력: cd클래스의 객체 CD를 생성해 사용자가 입력한 경로를 받아와 존재하는 경로인지 확인합니다. 만약 존재하는 경로이면, ls클래스의 객체 LS를 생성하여 해당 명령을 처리합니다. ls 클래스의 getList() 메소드는 contents 배열에 파일명들을 저장하고, list배열에 파일명, 크기 형식으로 변환하여 저장합니다. 그 후 OUTPUT에 해당 배열의 요소들을 저장합니다.
4. “GET”을 입력: cd클래스의 객체 CD를 생성해 client가 전송한 파일명이 경로에 존재하는지 확인하고, 만약 파일이 아닌 디렉토리라면 에러 메세지를 출력합니다. 그렇지 않으면 get클래스의 객체 GET에 getSize()메소드를 통해 객체CD의 필드를 설정해주어 추가적인 연결을 준비하고, 해당 파일의 사이즈를 OUTPUT에 반환합니다.
5. “PUT”을 입력: client로부터 온 메시지에서 파일명과 파일의 크기를 추출하고, put 클래스의 객체 PUT을 생성해 크기와 이름을 인자로 넘겨줘 파일을 받을 준비를 합니다.
6. 추가적 연결(데이터 채널)

클라이언트: 만약 사용자로부터 “GET”혹은 “PUT”의 입력이 들어오고, 서버로부터 에러 메세지가 오지 않았다면, 데이터채널을 열어 파일을 송수신할 준비를 합니다. 클라이언트는 dataSocket을 통해 서버의 dataPort에 접근해 서버와 새로운 연결을 시작합니다.

1. “GET”메시지: 파일을 서버로부터 받기 위해 파일을 받을 위치를 클라이언트의 현재위치로 지정합니다. 또한 해당 파일을 받기위해 클라이언트에 해당 파일명을 가진 빈 파일을 생성합니다. dataSocket을 통해 파일을 받을 stream을 설정하고, fileoutputStream을 통해 파일에 내용을 쓸 stream을 설정합니다. 그 후 서버로부터 데이터를 읽고 해당 내용을 data배열에 저장합니다. 해당 데이터는 5바이트의 메타데이터와 1000바이트의 청크데이터로 구성되어 있으므로, 상위 5바이트를 제외한 1000바이트의 데이터를 파일에 적습니다. 만약 이미 기존에 존재하는 파일명에 적는 경우, 에러메세지를 출력해 파일에 제대로 쓰이지 않았음을 알려줍니다. 그렇지 않으면 성공 메세지를 출력합니다.
2. “PUT”메시지: 파일을 읽어드릴 stream, 파일을 서버로 보낼 stream, 서버의 응답을 받을 stream을 생성합니다. data 배열에 5바이트의 메타데이터와 1000바이트 단위로 쪼갠 파일의 실제 데이터를 담고 서버로 1005바이트의 data를 전송합니다. 만약 파일이 빈 파일이라면 5바이트의 메타데이터만 전송합니다. 그 후 서버의 응답을 기다려 정상적으로 전송되었는지 확인합니다.

서버: 클라이언트로부터 “GET”, “PUT”의 메시지를 받고, 에러가 발생하지 않았으면 추가적인 연결을 준비합니다.

1. “GET”메시지: GET객체의 getFile() 메소드를 통해 파일을 송신합니다. dataSocket을 생성해 client의 응답을 기다리고, client가 연결되었으면 connectionSocket을 통해 파일을 전송할 준비를 합니다. 서버로부터 파일을 읽어드릴 stream, file을 클라이언트로 전송할 stream을 생성합니다. 파일을 전송할 data배열을 만들고, 5바이트의 메타데이터와 파일내용을 1000바이트단위로 쪼갠 부분을 함께 client로 전송합니다.
2. “PUT”메시지: PUT객체의 getFile() 메소드를 통해 파일을 수신합니다. GET객체의 getFile()메소드와 마찬가지로 client로부터 응답을 기다립니다. 또한 client로부터 파일을 수신 받을 stream, 파일을 새로 생성하고 쓸 stream, 파일이 성공적으로 저장되었는지에 대한 여부를 전송할 stream을 생성합니다. 파일을 전송받기 위해 1005 바이트의 data배열을 생성하고, 메타데이터인 상위 5바이트를 제외한 나머지 1000바이트를 파일에 새로 씁니다. 만약 같은 파일명의 파일이 이미 존재한다면 쓰지 않고 에러 메세지를 출력 후 전송합니다. 그렇지 않으면 성공적으로 수신했다는 메시지를 전송합니다.