Programming Assignment - Flow-Controlled FTP Program

Fall 2020

1. 개요

파일 전송을 지원하는 FTP 클라이언트/서버 프로그램을 구현하는 것이 이번 학기 프로그래밍 과제이다. 이 프로그래밍 과제의 목적은 TCP/IP 네트워킹 프로토콜을 이해하고 기본적인 자바 네트워크 프로그래밍 기법을 경험하고자 하는 것이다. FTP 프로그램은 명령어 채널과 데이터 채널이 분리되는 out-of-band signaling을 사용한다는 점에서 명령어와 데이터를 하나의 채널로 전송하는 in-band signaling 방식의 HTTP 프로토콜과는 정반대이다. 즉, 명령어는 클라이언트와 서버 사이의 상시 연결된 TCP 채널을 통해서 전송되고, 데이터 즉 파일의 전송이 필요하게 되면 그때마다 새로운 TCP 연결인 데이터 채널을 만들어서 사용한다.

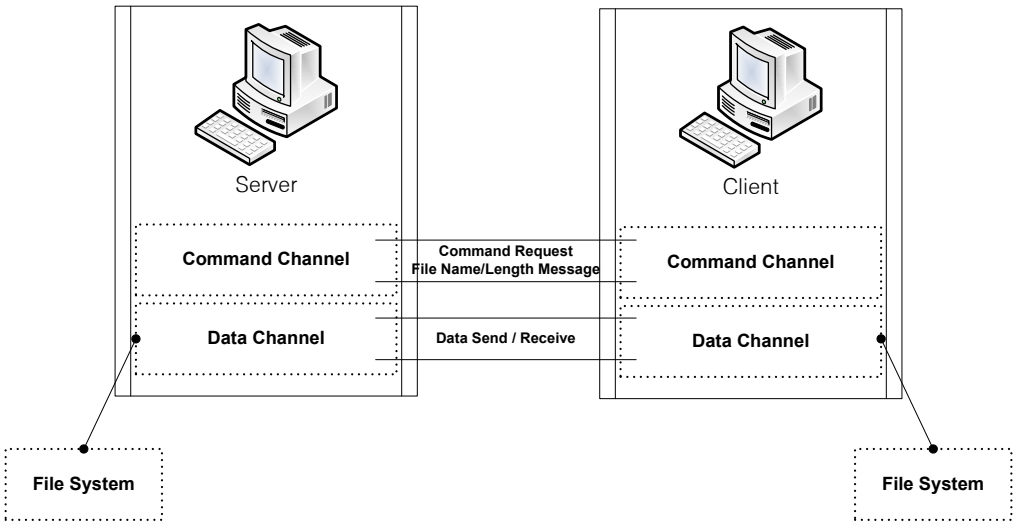


그림 1. FTP (File Transfer Protocol) Model

그림 1은 서버와 클라이언트 간에 command channel과 data channel을 통해 파일 송수신을 하는 기본 FTP model을 나타낸 것이다. 이를 기반으로 하여 아래에서 설명하는 과제 1, 2를 수행한다.

- **Assignment #1: FTP 기본 기능 구현** – 아래의 FTP 프로토콜 서버 및 클라이언트 프로그램을 Java 네트워킹 클래스를 사용하여 구현한다. 클라이언트와 서버는 명령어 채널과 데이터 채널을 명령어와 파일을 교환하는 FTP 프로토콜의 기본 기능을 구현한다. 이때 명령어/데이터 채널은 비트에러, 지연, 패킷 유실 등의 문제가 없는 이상적인 전송 수단으로 가정한다.
- **Assignment #2: SR 프로토콜 구현** – 과제 1의 구현을 바탕으로 데이터 채널에 SR(Selective Repeat) 기능을 구현한다. 즉, 이제 데이터 채널에는 비트에러, 전송지연, 패킷유실 등의 문제가 발생할 수 있어 SR 프로토콜이 필요하게 된다. SR은 클라이언트에서 서버로 전송하는

방향에 대해서만, 즉 PUT 명령어에 대해서만 구현하면 된다. 추가적으로 SR을 반대 방향에 대해서도 구현하는 경우에는 bonus 점수가 주어진다.

2. 세부 사항

2.1 명령어 (LIST, GET, PUT, CD, QUIT)

Command 채널은 전송 에러나 지연이 발생하지 않지만, 2번 과제에서는 데이터 채널에 전송 에러 및 패킷 유실이 발생할 가능성이 있다고 가정한다. (Assignment #1에서는 전송 에러 및 패킷 유실을 고려하지 않는다.) Command channel과 data channel을 위해 server machine에서 사용하는 포트 번호는 각각 2020과 2121번이다. 즉, 클라이언트는 서버의 2020 포트에 command channel 생성을 요청하고, 데이터 채널이 필요하게 되면 서버의 2121 포트 번호를 지정한다.

- 클라이언트가 파일 송수신을 위해서 "LIST", "GET", "PUT", "CD"와 같은 명령어를 요청할 수 있다. 아래 명령어 수행의 예와 유사하게 수행 결과는 클라이언트가 화면에 출력하여 사용자가 확인할 수 있도록 한다.
    - LIST: 파일 서버의 특정 디렉토리의 파일을 확인하는 명령어
    - GET: 파일 서버로부터 파일을 download 하는 명령어
    - PUT: 파일 서버에 파일을 upload 하는 명령어
    - CD: 파일 서버에서의 현 디렉토리 위치를 변경하는 명령어
    - QUIT: 명령어 채널을 해제하고 클라이언트의 수행을 종료하는 명령어
- 데이터 채널을 통해 전송되는 클라이언트-to-서버 방향의 데이터 메시지의 포맷은 {SeqNo(1byte), CHKsum(2bytes), Size(1byte), 데이터청크(1000bytes)}로 구성된다. 단, CHKsum 필드는 자리만 확보되어 있고, 전송 데이터에 대해 실제로 사용하지 않고 저장된 값만 보고 에러 발생 여부를 판단하는 방식으로 이용된다. 서버-to-클라이언트 방향의 제어 메시지는 {SeqNo(1byte), CHKsum(2bytes)}와 같이 구성된다. 마찬가지로 CHKsum 은 따로 계산하지 않고 0x0000 이면 비트 에러가 없는 상황이고 0xFFFF 이면 전송 중에 비트 에러가 발생하였다고 생각한다.
- 파일 전송 명령어의 파라미터로 사용되는 파일 또는 디렉토리로는 아래의 네 가지 종류를 지원해야 한다.
    - (절대경로)
    - (상대경로)
    - "." (마침표 - 현재 위치)
    - ".." (마침표 2 개 - 상위 디렉토리)
- 제출 과제의 동작 시험을 위해 timeout 데이터 채널에 네트워크 에러 상황을 모의하기 위해서 아래의 명령어도 지원한다 **(Assignment #2 에만 해당)**
    - DROP ChkNo: 지정된 청크번호(ChkNo)의 패킷을 클라이언트가 drop 한다.
    - TIMEOUT ChkNo: ChkNo 에 해당하는 패킷에 대한 타임아웃 이벤트가 발생하도록 의도적으로 전송을 지연시킨다.
    - BITERROR ChkNo: CHKsum 필드는 0x0000 으로 초기화되는데 에러 상황을 나타내기 위해 전송 전에 필드 값을 0xFFFF 로 변경한다.

- 아래의 샘플 명령어 수행 결과의 status code 는 예시로 제공된 것이며 실제 구현 시는 적절한 코드를 정의하여 사용하도록 한다.
- 예시에서의 SR 에 대한 내용은 assignment #2 에 해당한다. PUT/GET 명령에 의한 파일 전송 시 과제 1 에서는 청크 전송 시 '#'를 한번 화면에 출력하고, 과제 2 에서는 청크의 SeqNo 를 화면에 출력하여 ftp 진행이 이루어지고 있음을 알 수 있도록 한다.

CD (디렉토리명)	• 서버에서의 '현재 디렉토리 위치'를 변경할 때 사용되는 명령어이다. 기본 디렉토리는 서버 프로그램이 수행된 폴더가 된다. • 이에 대해 서버는 새로운 디렉토리 위치로 응답한다. 단, 응답은 절대 경로이며 클라이언트 화면에 출력되어야 한다. • 만일 파라미터가 디렉토리가 아닌 경우 에러 메시지를 출력하도록 한다.
참고사항	• File 클래스의 exists(), isDirectory(), getCanonicalPath() 메소드 • Path 클래스의 isAbsolute() 메소드 • PrintWriter, BufferedReader 클래스 등
클라이언트 화면	CD /home/username/ftp/ // ., .., 상대경로도 지원해야 한다 /home/username/ftp
명령어 채널 전송 (클라이언트->서버) (서버->클라이언트)	 Line 1: CD /home/username/ftp/ Line 1: 200 Moved to /home/username/ftp // status code & phrase
서버 화면	Request: CD /home/username/ftp/ Response: 200 Moved to /home/username/ftp
클라이언트 화면	CD // no argument의 경우 현 디렉토리 위치를 리턴한다 /home/username/ftp
명령어 채널 전송 (클라이언트->서버) (서버->클라이언트)	 Line 1: CD Line 1: 200 Moved to /home/username/ftp // status code & phrase
서버 화면	Request: CD Response: 200 Moved to /home/username/ftp
클라이언트 화면	CD temp Failed – directory name is invalid
명령어 채널 전송 (클라이언트->서버) (서버->클라이언트)	 Line 1: CD temp Line 1: 501 Failed – directory name is invalid // status code & phrase
서버 화면	Request: CD temp Response: 501 Failed – directory name is invalid

LIST (디렉토리명)	• 서버가 유지하는 파일 리스트를 받아올 때 사용한다. • 서버는 해당 디렉토리 내의 결과를 "(파일명),(파일크기)₩n"로 응답한다. 이 응답은 클라이언트 화면에 출력되어야 한다. • 디렉토리의 크기는 '-'로 표기한다. • 파일의 크기는 byte 단위로 표기한다. • 만일 파라미터가 디렉토리가 아닌 경우 에러 메시지를 출력하도록 한다.
참고사항	• File 클래스의 listFiles() 메소드
클라이언트 화면	LIST /home/username/ftp/ subdir1,- subdir2,- subdir3,- test.txt,13 test.jpg,71514
명령어 채널 전송 (클라이언트->서버) (서버->클라이언트)	 Line 1: LIST /home/username/ftp/ Line 1: 200 Comprising 5 entries // status code & phrase Line 2: subdir1,-,subdir2,-,subdir3,-,test.txt,13,test.jpg,71514
서버 화면	Request: LIST /home/username/ftp/ Response: 200 Comprising 5 entries
	아래의 상대 경로도 위의 예시와 유사하게 동작해야 한다. LIST . // 현 디렉토리 LIST .. // 상위 디렉토리 LIST subdir1 // 상대 경로
클라이언트 화면	LIST nodir Failed – directory name is invalid
명령어 채널 전송 (클라이언트->서버) (서버->클라이언트)	 Line 1: LIST nodir Line 1: 501 Failed - Directory name is invalid // status code & phrase
서버 화면	Request: LIST nodir Response: 501 Failed - Directory name is invalid

GET (파일명)	• 서버로부터 지정된 파일을 수신하고자 할 때 사용한다. 파일은 클라이언트의 현 디렉토리 위치에 저장된다. • 클라이언트가 서버로 'GET 파일명' 명령어를 전송하면, 서버는 응답으로 해당 파일의 크기를 보내준다. 클라이언트는 데이터 채널을 열어서 파일 다운로드를 시작한다. 청크의 크기는 1000 바이트로 설정되는데, 한 청크를 수신할 때마다, 과제 1 에서는 '#'를, 과제 2 에서는 SeqNo 를 화면에 출력하여 파일 다운로드의 진행 상황을 화면 출력을 통해 알 수 있도록 한다.
참고사항	• File 클래스의 length() 메소드 • FileInputStream, FileOutputStream, InputStream, OutputStream 등
클라이언트 화면	GET /home/username/ftp/subdir1/a.txt Received a.txt / 8700bytes # # # # # # # # Completed… // 과제 1은 #를, 과제 2는 SeqNo를 출력
명령어 채널 전송 (클라이언트->서버) (서버->클라이언트)	Line 1: GET /home/username/ftp/subdir1/a.txt Line 1: 200 Containing 8700 bytes in total // status code & phrase
서버 화면	Request: GET /home/username/ftp/subdir1/a.txt Response: 200 Containing 8700 bytes in total
데이터 채널 전송	데이터 채널을 열어 SR 프로토콜에 따라 청크가 전송된다.
클라이언트 화면	GET nofile.txt Failed – Such file does not exist!
명령어 채널 전송 (클라이언트->서버) (서버->클라이언트)	Line 1: GET /home/username/ftp/subdir1/a.txt Line 1: 401 Failed – No such file exists // status code & phrase
서버 화면	Request: GET /home/username/ftp/subdir1/a.txt Response: 401 Failed – No such file exists

PUT (파일명)	• 서버로 지정된 파일을 송신하고자 할 때 사용한다. 파일은 서버의 현 디렉토리를 기준으로 저장된다. • 'PUT 파일명' 명령어가 주어지면 클라이언트는 서버로 해당 파일의 이름과 크기를 보내준다. 클라이언트는 서버와 데이터 채널을 열어서 파일 업로드를 시작한다. 청크의 크기는 1000 바이트로 설정되는데, 한 청크를 전송할 때마다 화면에 '#' 혹은 SeqNo 를 한번 출력하여 파일 업로드의 진행 상황을 화면 출력을 통해 알 수 있도록 한다.

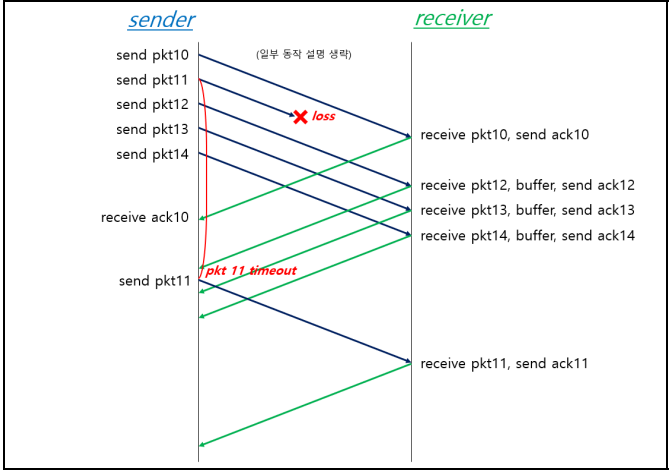
클라이언트 화면	PUT a.txt a.txt transferred / 8700 bytes ######## Completed… // 과제 1은 #를, 과제 2는 SeqNo를 출력
명령어 채널 전송 (클라이언트->서버) (서버->클라이언트)	 Line 1: PUT a.txt Line 2: 8700 // 파일 크기 Line 1: 200 Ready to receive
서버 화면	Request: PUT a.txt Request: 8700 Response: 200 Ready to receive // status code & phrase
데이터 채널 전송	데이터 채널을 열어 SR 프로토콜에 따라 청크가 전송된다.
클라이언트 화면	PUT b.jpg Failed for unknown reason
명령어 채널 전송 (클라이언트->서버) (서버->클라이언트)	 Line 1: PUT a.txt Line 2: 8700 // 파일크기 Line 1: 501 Failed for unknown reason // status code & phrase
서버 화면	Request: PUT a.txt Request: 8700 Response: 501 Failed for unknown reason

2.2 이벤트 (DROP, TIMEOUT, BITERROR)

SR 프로토콜의 동작을 모의하기 위해 DROP, TIMEOUT, BITERROR 이벤트를 다음과 같이 구현하여야 한다. 예를 들어 "DROP R1,R2" 명령어는 후속 PUT 명령어를 수행할 때 파일을 구성하는 첫 번째와 두 번째 청크를 클라이언트에서 제외시켜 전송 과정 중에서의 packet loss를 모의할 수 있도록 한다. "TIMEOUT R1"은 후속 PUT 명령어 수행 결과 발생하는 첫 번째 메시지를 클라이언트에서 전송 전에 현 timeout 값의 2배로 지연시켜 특정 패킷에 대한 전송 지연 상황을 연출할 수 있도록 한다. 단, 동일 파일의 다른 패킷은 지연없이 정상적으로 전송됨을 유의한다. 마지막으로 CHKsum 필드는 0x0000으로 초기화되는데, BITERROR 이벤트 명령어가 주어지면 에러 상황을 나타내기 위해 인위적으로 값을 0xFFFF로 변경한다. 다음은 DROP과 TIMEOUT 이벤트 수행 결과의 예시이다.

DROP

DROP R(청크번호),R(청크번호)...



$ DROP R2 // 두 번째 패킷을 drop

$ PUT a.txt

a.txt transferred 8700 bytes

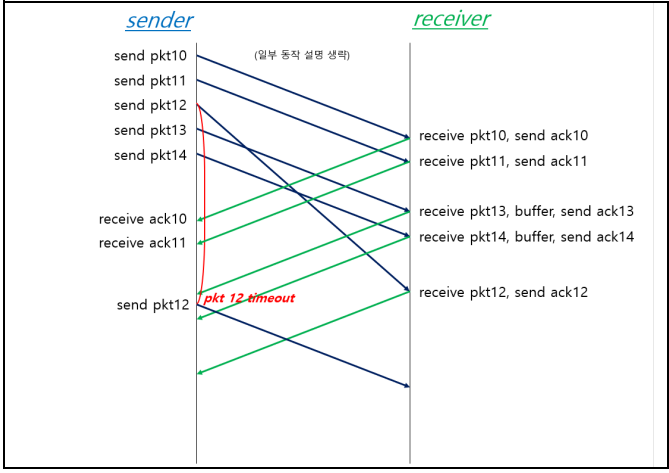
10 11 12 13 14 // 5 개의 패킷이 전송되지만 두 번째인 11 번이 유실됨

10 acked, 12 acked, 13 acked, 14 acked

11 timed out & retransmitted

TIMEOUT

TIMEOUT R(청크번호),R(청크번호)...



$ TIMEOUT R3 // 세 번째 패킷을 지연

$ PUT a.txt

a.txt transferred 8700 bytes

10 11 12 13 14

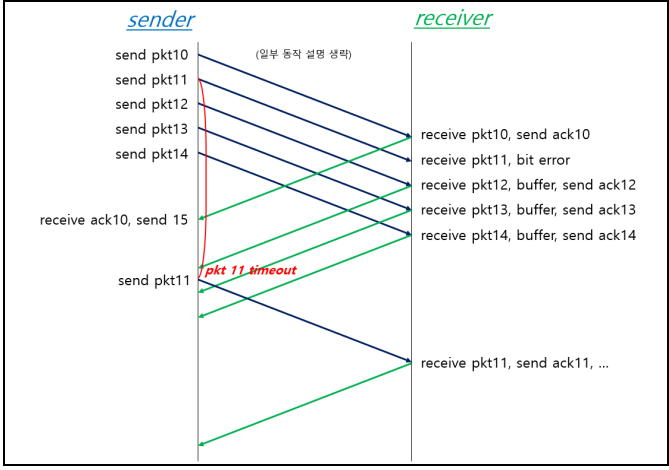
// 5 개의 패킷이 전송되지만 세 번째인 12 번에 대한 timeout 발생

10 acked, 11 acked, 13 acked, 14 acked

12 timed out & retransmitted

BITERROR

BITERROR R(청크번호),R(청크번호)...



$ BITERROR R2 // 두 번째 패킷에 비트 에러 발생을 연출함

$ PUT a.txt

a.txt transferred 8700 bytes

10 11 12 13 14

// 5 개의 패킷이 전송되지만 11 번에 대한 비트 에러 발생

10 acked, 12 acked, 13 acked, 14 acked

11 timed out & retransmitted

3. 유의 사항

SR 프로토콜 구현 파라미터로 command & data channel 에서는 **seqNo range 0..15, window size 5, sender timeout 1 초**로 설정한다. 데이터 채널로 전송되는 파일은 1000 바이트 단위 청크로 분할되는데 GET/PUT 수행 시 마지막 청크는 부분적으로 채워져 있을 수도 있음을 유의한다. 또한 클라이언트 화면에 매 청크를 송신 혹은 수신하게 되면 화면에 '#' 혹은 SeqNo 를 출력하여 전송이 진행되고 있음을 알 수 있도록 한다.

- 프로그래밍 언어는 Java 만 허용한다.
- 제출 소스 코드에서의 서버 및 클라이언트의 default IP 주소는 "127.0.0.1", 서버의 기본 포트 번호는 "2020"과 "2121"을 사용한다. 즉, 클라이언트와 서버 수행 시 command line argument 로 ftp server host, control/data port no 가 주어지지 않으면 기본 호스트 IP 와 포트 번호를 사용하도록 한다.
- 단, 프로그램 실행 시 포트 번호를 파라미터로 지정할 수 있도록 한다.
  - Java FTPServer <control port no> <data port no>
  - Java FTPClient <ftp server host> <control port no> <data port no>

4. 과제 제출

소스 및 설계 문서 파일을 포함하여 jar/zip 형태로 압축하여 파일명을 "이름-assignment.jar/zip"으로 하여 블랙 보드를 통해 제출한다. 이 압축 파일에는 컴파일 된 class file 이나 executable 파일이 포함되지 않도록 유의한다. 제출하는 설계 문서에서는 프로그램의 설계와 구조의 기술 외에도 제출된 프로그램을 컴파일 및 실행하는 방법과 클라이언트-서버 동작 절차도 단계적으로 정확하게 기술하여야 한다. 설계 문서에는 버퍼, 타이머 등 프로토콜의 구성 요소가 어떻게 구현되었는지에 대한 설명도 포함되도록 한다. 또한 소스 코드를 단순히 이 문서에 포함시키는 불필요한 일은 하지 않도록 한다. 프로그램 test 시 기술된 절차를 그대로 따름에도 불구하고 프로그램이 수행되지 않는 경우에는 제출 프로그램이 동작하지 않는 것으로 간주한다.

- 1 차 과제에 대한 제출 시한은 **11 월 17 일 화요일 23:59** 까지이다.
- 2 차 과제에 대한 제출 시한은 **12 월 1 일 화요일 23:59** 까지이다.

제출 시한을 지나 24 시간 이내까지 제출한 경우 50%를, 48 시간 이내까지는 25%를 인정하며, 그 이후 제출은 인정하지 않는다.

마지막으로 과제 수행 및 제출 방법에 대한 문의는 이효식 조교(lhs9394@naver.com)에게 이메일로 문의할 수 있다.