

Programming Assignment #2

2020. 12. 01.

Jihun Kim (jihunkim@hanyang.ac.kr)

Development Environment

- **macOS Big Sur** 11.0.1 20B29 x86_64
- **java 12.0.2** 2019-07-16

Build and Run

```
$ cd /path/to/code/  
$ javac FTPServer.java  
$ javac FTPClient.java  
$ java FTPServer &  
$ java FTPClient
```

SR Implementation

FTP server와 client에 SR (selective repeat) 알고리즘을 구현한다.

Client

윈도우

윈도우 내에는 완성된 패킷들이 들어가 있고, 후술할 타이머가 이들을 반복해서 전송한다. 물론 첫 전송 때도 활용한다.

윈도우는 실제로 한 칸씩 slide하지 않고, 윈도우의 첫 부분을 가리키는 포인터만 한 칸 이동시킴으로써 구현했다. 이를 관리하기 위해 몇 가지 변수들을 도입했는데, 각각 아래와 같다. 길이 1짜리 배열로 구현한 것은 다른 thread 에서도 값을 볼 수 있게 하기 위해서다.

```
int[] winBase = {0};           // Index of firstly sent chunk in the window.  
int[] numBuffered = {0};       // Number of buffered chunks in the window.  
byte[] firstSeqNo = {0};       // First sequence number in the window.  
byte nextSeqNo = 0;            // Next sequence number, in range of [0, maxSeqNo].
```

타이머

패킷이 보내질 때 별도 쓰레드로 작동하는 타이머를 작동시킨다. 이 타이머는 timeout 시간이 될 때마다 반복해서 윈도우 내의 패킷을 보내는 역할을 한다. ACK가 와서 중단되기 전까지 해당 패킷을 계속해서 전송한다.

```
Timer[] timers = new Timer[DataChunkC2S.winSize];
```

ACK 처리

별도의 ACKListener 쓰레드가 동작하면서, ACK이 올 때마다 신속하게 윈도우에 내용을 반영할 수 있도록 한다. 구체적으로는 ACK 값을 보고 window의 어떤 위치의 타이머를 disable 시킬 지 결정해 무효화시키고, 타이머 배열이 null을 가리키게 한다. window는 타이머 배열을 확인해 이 값이 null이면 ACK가 온 것으로 판단하고, 윈도우를 slide해 다음 패킷들이 전송될 수 있게 한다.

Packet Drop

타이머만 작동시키고, 처음에 패킷을 보내지 않는 식으로 구현했다. 타이머가 돌면서 해당 해킷을 timeout 시간 뒤에 재전송하게 된다. 다만 표준출력에는 마치 패킷을 보낸 것처럼 출력된다.

Bit Error

처음에 패킷을 보낼 때 checksum 값을 수정하여 전송한다. 전송한 다음에는 윈도우에 있는 패킷의 checksum 값을 다시 되돌려 놓음으로써, 재전송 시에는 bit error가 없는 패킷이 정상적으로 전송될 수 있게 한다.

Timeout

이것도 구현을 위해 별도의 타이머 쓰레드가 필요했다. 일단 처음에 전송을 하지 않고, 별도의 타이머 쓰레드에게 전송을 맡긴다. 이 쓰레드는 2 * timeout 시간 동안 대기하다가 나중에 패킷을 전송한다. 이 시간이 timeout 시간보다 더 길기 때문에, 대기하는 도중에 타이머 쓰레드가 timeout을 감지하고 패킷을 전송한다. 나중에 보내진 패킷은 server에서 무시한다. 이것도 packet drop과 마찬가지로, 실제로는 나중에 전송함에도 불구하고 표준출력에는 마치 패킷을 지금 보낸 것처럼 출력한다.

Server

윈도우

Client와 달리, 서버에서 파일을 받을 때는 여러 개의 쓰레드가 필요하지 않았다. 그냥 ACK를 받고, 윈도우를 slide할 수 있으면 slide한 다음, 다시 새로운 ACK를 받는다. 전송받은 파일의 저장은 윈도우 slide 시 이루어진다.

단, ACK를 받았을 때 그 sequence number가 [rcvbase-N,rcvbase-1] 범위 안에 있는 경우에는 따로 버퍼링하지 않고, ACK만 보내 준다. 이는 ACK이 유실됐을 경우를 대비한 것으로, 과제에서는 일어나지 않지만 그냥 구현했다.

실행 화면

CD, LIST

CD 명령어와 LIST 명령어 모두 정상적으로 작동하는 것을 확인할 수 있다. 절대경로와 상대경로 모두 지원된다.

```
~/codes/ene4019/src  main+ • 43%
(base) → src git:(main) x java FTPServer
Running..
Connection established: /127.0.0.1
Response: 220 Hello
Request: cd
Response: 212 /Users/starlett/codes/ene4019/src
Request: cd ..
Response: 200 Moved to /Users/starlett/codes/ene4019
Request: list .
Response: 200 Comprising 15 entries
Request: list /Users/starlett/codes/ene4019/test
Response: 200 Comprising 2 entries
Request: cd /Users/starlett/codes/ene4019/test
Response: 200 Moved to /Users/starlett/codes/ene4019/test
Request: list .
Response: 200 Comprising 2 entries
```

```
~/codes/ene4019/src  main+ • 29%
(base) → src git:(main) x java FTPClient
Connection established: /127.0.0.1
Server responded: Hello
> cd
Server responded: /Users/starlett/codes/ene4019/src
> cd ..
Server responded: Moved to /Users/starlett/codes/ene4019
> list .
Server responded: Comprising 15 entries
test4.txt, 1001
test5.txt, 45001
.DS_Store, 14340
test1.txt, 1000
test, -
test2.txt, 1
test3.txt, 0
out, -
README.md, 255
.gitignore, 79
ene4019.iml, 423
doc, -
.git, -
.idea, -
src, -
> list /Users/starlett/codes/ene4019/test
Server responded: Comprising 2 entries
test5.txt, 45001
test1.txt, 1000
> cd /Users/starlett/codes/ene4019/test
Server responded: Moved to /Users/starlett/codes/ene4019/test
> list .
Server responded: Comprising 2 entries
test5.txt, 45001
test1.txt, 1000
```

Drop

3번 패킷을 의도적으로 drop시켰을 때의 화면이다. 표준출력 상으로는 3번이 전송된 것으로 출력되지만, 실제로는 drop되어 전송되지 않았다.

3번 패킷에 대한 ack이 오지 않았더라도 0~2번 패킷에 대해 ack이 왔기 때문에 윈도우는 추가로 전진하여 5~7번 패킷을 보낸다. 이후 3번 패킷에 대한 ack이 올 때까지 기다린다.

timeout이 되고 나면 3번 패킷이 재전송되며, 그에 따른 ack이 도착한다. 윈도우는 이제 계속해서 전진하여 남은 패킷들을 전송한다.

```
~/codes/ene4019/src  main+ •
Request: put ../test5.txt
Response: 200 Ready to receive
Request: 45001 bytes
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 0 1 2 3 4 5 6 7 8 9
10 11 12 13 14 15 0 1 2 3 4 5 6 7 8 9 10 11 12 13 Done.
```


```
~/c/ene4019/src  main+ •
> drop r3
> put ../test5.txt
Server responded: Ready to receive
Sent: 0 --> Server
Sent: 1 --> Server
ACKed: 0 <-- Server
Sent: 2 --> Server
ACKed: 1 <-- Server
ACKed: 2 <-- Server
Sent: 3 --> Server
Sent: 4 --> Server
ACKed: 4 <-- Server
Sent: 5 --> Server
ACKed: 5 <-- Server
Sent: 6 --> Server
ACKed: 6 <-- Server
Sent: 7 --> Server
ACKed: 7 <-- Server
Timeout, resent: 3
ACKed: 3 <-- Server
Sent: 8 --> Server
ACKed: 8 <-- Server
Sent: 9 --> Server
ACKed: 9 <-- Server
Sent: 10 --> Server
```

Timeout

timeout을 패킷 3번에 걸었을 때의 화면이다. drop과 거의 동일하게 작동하는 것을 확인할 수 있다. 첫 번째로 보내진 패

킷이 매우 느리게 가므로, timeout이 발생하게 되고 재전송된다.

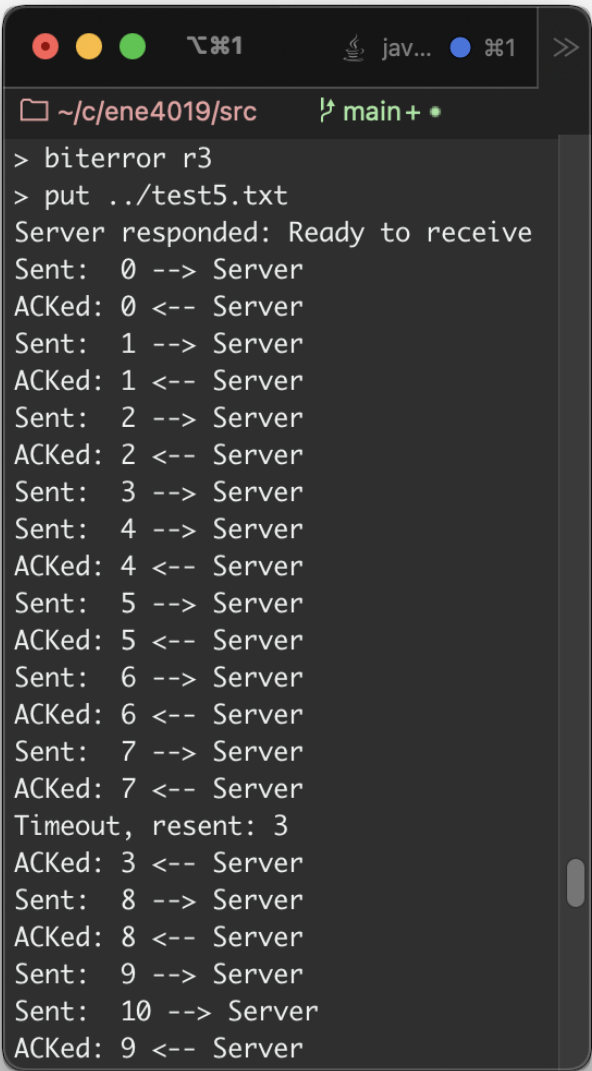
다만 packet drop과 다른 점은, 3번 패킷이 아예 사라진 것은 아니기 때문에 나중에 뜬금없는 타이밍에 3번에 대한 ACK이 온다는 점이다.



```
> timeout r3
> put ../test5.txt
Server responded: Ready to receive
Sent: 0 --> Server
Sent: 1 --> Server
ACKed: 0 <-- Server
Sent: 2 --> Server
ACKed: 1 <-- Server
ACKed: 2 <-- Server
Sent: 3 --> Server
Sent: 4 --> Server
ACKed: 4 <-- Server
Sent: 5 --> Server
ACKed: 5 <-- Server
Sent: 6 --> Server
ACKed: 6 <-- Server
Sent: 7 --> Server
ACKed: 7 <-- Server
Timeout, resent: 3
ACKed: 3 <-- Server
Sent: 8 --> Server
ACKed: 8 <-- Server
Sent: 9 --> Server
Sent: 10 --> Server
ACKed: 9 <-- Server
```

Bit error

Bit error가 발생하면 서버는 그냥 ACK을 하지 않는다. 따라서 client 입장에서는 bit error가 발생한 것은 drop이 발생한 것과 차이가 없어 drop과 동일하게 작동한다.



```
> biterror r3
> put ../test5.txt
Server responded: Ready to receive
Sent: 0 --> Server
ACKed: 0 <-- Server
Sent: 1 --> Server
ACKed: 1 <-- Server
Sent: 2 --> Server
ACKed: 2 <-- Server
Sent: 3 --> Server
Sent: 4 --> Server
ACKed: 4 <-- Server
Sent: 5 --> Server
ACKed: 5 <-- Server
Sent: 6 --> Server
ACKed: 6 <-- Server
Sent: 7 --> Server
ACKed: 7 <-- Server
Timeout, resent: 3
ACKed: 3 <-- Server
Sent: 8 --> Server
ACKed: 8 <-- Server
Sent: 9 --> Server
Sent: 10 --> Server
ACKed: 9 <-- Server
```

기타

주석을 Javadoc 스타일에 맞춰 나름 세밀하게 작성했는데, 이를 보고서에 다 옮기는 것은 지면 낭비인 것 같아 그렇게 하지 않았다. 혹시 미흡한 부분이 있다면 코드의 주석을 확인하면 되겠다.