

# 컴퓨터네트워크 - Programming Assignment 2

2019019016 서시연

## <프로그램 설명>

1. CD, LIST, QUIT는 assignment1과 동일하게 작동한다. 또한 PUT과 GET의 기본적인 동작은 assignment1과 같다.
2. 입력받은 command를 “ ”를 기준으로 split했을 때 0번째 값이 DROP, TIMEOUT, BITERROR 중 하나라면 split한 결과의 1번째 값을 “,”을 기준으로 split해서 String 배열인 tmp에 저장한다. 그 후 drop, timeout, biterror 중 해당하는 것에 tmp의 길이만큼의 int형 배열을 할당해주고, for문을 이용해 substring(1)으로 tmp에 저장되어있는 (Rn) 형식의 값에서 숫자부분(n) 값만을 가져와 int형으로 바꿔서 drop, timeout, biterror에 저장한다.
3. initDrop, initTimeout, initBiterror
  - drop, timeout, biterror가 null이 아니라면 입력받은 cntChunk값이 drop, timeout, biterror에 있는지 확인한다. 있다면 true, 없거나 drop, timeout, biterror가 null이라면 false를 리턴한다.
4. isExistInRcvWindow(int seqNum) : seqNum이 receiver의 receivebase내부에 있는지 확인한다. 있다면 true, 없다면 false를 리턴한다.
5. isLostedAck(int seqNum) : receiver가 이미 ack을 보낸 seqNum에 대해 sender가 재전송한 경우 (rcvbase-N~rcvbase-1)에 해당하면 true, 아니면 false를 리턴한다.

## [ControlSend class]

- sender를 관리하기 위한 inner class Sendbase가 있다.
- Sendbase는 int type의 sendbase와 크기 16인 boolean 배열 isAacked, 크기 16인 Timer 배열 timers, 1005 바이트 크기의 byte 배열을 16개 갖는 send\_buffer를 가진다.
- isAacked는 해당하는 seqNum의 ack을 받았을 때 true가 되며, 그 seqNum가 sendbase를 업데이트 할 때 사용되면 false가 된다.
  - ex) sendbase가 1일 때 1에 문제가 생겨 2,3,4에 대한 ack을 먼저 받은 후 재전송된 1에 대한 ack을 받은 경우, ack을 받은 뒤 isAacked[2], isAacked[3], isAacked[4] 값은 true였다가, 1에 대한 ack을 받은 뒤 sendbase를 5로 업데이트할 때 false가 된다.
- Thread를 통해 ack을 받고, isAacked, sendbase, timers, send\_buffer를 수정하므로, 일관성을 위해 이에 접근하는 하기 위한 메소드들을 synchronized로 선언한다.
- 메소드
  - 1) initIsAacked() : Arrays.fill 메소드를 이용해 isAacked의 모든 element를 false로 만들어준다.
  - 2) setIsAackedTure(int seqNum) : isAacked[seqNum]을 true로 만들어준다.
  - 3) checkIsAacked() : isAacked의 상태를 확인해 isAacked가 true인 곳까지 sendbase를 증가시켜 준다. (true였던 isAacked 값은 false로 바꿔준다.)
  - 4) isExistInsSendWindow(int seqNum) : seqNum이 윈도우 내부에 있는지 확인한다. 내부에 있다면 true, 아니라면 false를 리턴한다.
  - 5) startTimer(int seqNum, TimerTask timerTask) : timers[seqNum].schedule(timerTask,1000)을 실행시킨다.
  - 6) cancelTimer(int seqNum) : timers[seqNum].cancel을 실행한다.
  - 7) initSendBuffer() : send\_buffer의 값들을 null로 초기화한다.
  - 8) setSend\_buffer(int seqNum, byte[] sendData) : send\_buffer[seqNum]에 sendData를 저장한다.
  - 9) getSend\_buffer(int seqNum) : send\_buffer[seqNum]을 리턴한다.

## [PUT - Client]

1. cntChunk라는 변수를 뒤서, while문이 한 번 돌 때마다(청크를 하나 보낼 때 마다) 값을 증가시킨다.  
boolean 값인 isDrop, isTimeout, isBitererror를 각각 initIsDrop(cntChunk), initIsTimeout(cntChunk), initIsBitererror(cntChunk)를 통해 초기화한다. (해당 청크에 drop, timeout, bitererror가 발생하는지를 체크)
2. 파일에서 data로 읽어온 길이 length를 header의 3,4번 element에 저장하고, checksum값, 즉, header의 1,2번 element는 0으로 초기화한 뒤 System.arraycopy를 통해 header와 data의 내용을 send\_data에 카피해서 합친다.
3. seqNum에 header[0] 값을 저장하고, send\_data.clone()과 ControlSend.setSend\_buffer를 이용해 ControlSend의 send\_buffer[seqNum]에 send\_data값을 저장한다.
4. ack을 받는 용도의 Thread인 acceptAck을 만든다.
  - 크기 3인 byte array ack을 만들고, seq를 선언한다.
  - while문에서 dataInputStream.read()를 이용해 server로부터 ack을 읽어온 뒤 ack[0]을 seq에 저장하고, ControlSend.isExistInSendWindow(seq)가 true일 때 (seq가 sendwindow내부에 있을 때) while을 빠져나온다.
  - seq acked를 출력해 ack을 받았음을 표시한다. (과제 명세서의 예시에는 seqNo들이 먼저 쭉 출력되고 'seqNo acked, '가 쭉 이어서 출력되는 형식으로 되어있는데, 실제 seqNo에 해당하는 청크를 전송할 때마다 seqNo를 출력하고, seqNo에 대한 ack을 받을 때마다 'seqNo acked'를 출력하면 둘이 섞여서 출력되므로 가독성이 좋지 않아 seqNo를 출력할 때나, 'seqNo acked'를 출력할 때 매번 한 줄씩 띄우도록 했다.)
  - ControlSend.setSend\_buffer를 이용해 send\_buffer[seq]를 null로 만들고, ControlSend.cancelTimer(seq)를 호출하고, ControlSend.setIsAackedTrue(seq)를 호출해 ControlSend의 isAacked[seq]를 true로 만들어준다.
  - total은 전송하는 파일의 전체 크기를 1000으로 나눈 것의 ceiling을 한 값이다. ControlSend.getSendbase() 값과 total%16이 같고, total이 cntChunk값과 같다면 모든 파일이 전송되었고, 모든 ack을 받은 것이다.
    - 따라서 drop, timeout, bitererror를 null로 초기화해주고, 서버와의 데이터 전송을 위한 input, output stream, socket을 close해주고 'Completed...'를 출력해준다.
5. acceptAck.start()를 이용해 ack을 받을 Thread를 실행시킨다.
6. while문에서 ControlSend.isExistInSendWindow(seqNum)이 true일 때 빠져나간다. (sendbase가 증가되지 않아, 윈도우 내에 있지 않은 경우, 윈도우 내로 들어갈 때까지 대기)
7. 전송할 데이터의 seqNum을 출력하고 ControlSend.startTimer(seqNum,timerTask)을 실행해서, 1초 후 타임아웃이 발생해 재전송이 일어나도록한다.
  - timerTask에서는 ControlSend의 send\_buffer[seqNum]이 null이 아니라면 'seqNum timed out & retransmitted'를 출력하고, send\_buffer[seqNum]을 서버로 전송한 뒤, ControlSend.startTimer(seqNum,restartTimerTask)를 이용해 타이머를 재시작한다.
8. isDrop이 false인 경우 isBitererror가 true이면 send\_data[1], send\_data[2]를 0xFF로 설정한다.
  - 8-1. 만약 isTimeout이 true라면 Timer와 TimerTask를 이용해 send\_data를 2초 뒤에 서버로 전송하게 한다.
  - 8-2. isTimeout이 false라면 send\_data를 서버로 바로 전송한다.
9. header[0]을 1증가시키고, header[0]이 16이되면 0으로 만들어준다.
10. 파일에서 data로 읽어온 길이 length가 -1이 될 때까지 1~9과정을 반복한다.
11. 만약 보내고자 하는 파일의 길이가 0이라면 while문을 그냥 통과하기 때문에 4의 모든 파일 내용을 전송했고 ack을 다 받았을 때와 같은 행동을 한다.

## [GET - Client]


1. 3byte 크기의 byte 배열 ack을 선언하고, rcvbase를 0으로 초기화하고, received\_buffer의 모든 값을 null로 초기화한다.
2. while문에서 received\_data를 읽어오고, 읽어온 길이를 length에 저장한다.
3. received\_data[0]을 seqNum에 저장한다.
- 4-1. seqNum이 rcvWindow안에 있고 received\_data[1]과 received\_data[2]가 0이라면 (bit error가 없는 경우)
  - seqNum이 rcvbase와 같은 경우,
    - 받은 seqNum을 출력해준다.
    - received\_data를 header와 data로 쪼개고 header[3], header[4]에 저장되어 있는 값을 size에 저장한다.
    - 파일에 data에 있는 값을 써주고, cntSize에 size를 더해주고, rcvbase에 (rcvbase+1)%16을 저장한다.
    - temp에 rcvbase값을 저장한 뒤 for문을 이용해 receiver\_buffer[temp]가 null이 아닐 때까지 receiver 버퍼에 저장되어 있는 data들을 가져와서 파일에 써주고, receiver\_buffer[temp]를 null로 만들어준다. for문을 빠져나와서 rcvbase에 temp값을 저장한다.
  - seqNum이 rcvbase가 아닌 경우, receive\_buffer에 received\_data를 넣어준다.
  - 이후 받아온 received\_data의 seqNum에 대한 ack을 전송한다.
- 4-2. 만약 isLostedAck(seqNum)이 true이고 bit error가 없다면 받아온 데이터의 seqNum에 대한 ack을 전송한다.
5. length가 -1이고, cntSize가 fileSize와 같아질 때까지 2~4의 과정을 반복한다.
6. while문을 빠져나온 뒤 data를 받기위한 소켓, input,output stream을 close해준다.

## [GET & PUT - Server]

1. 서버의 GET은 클라이언트의 PUT과, 서버의 PUT은 클라이언트의 GET과 같은 과정을 거쳐 이루어진다.
2. 단, 클라이언트에서는 데이터를 송/수신할 때마다 seqNo을 출력하고, ack을 받을 때마다, 그리고 송/수신이 끝났을 때, retransmit할 때 메시지를 출력하는데 반해 서버에는 이런 정보를 출력하지 않는다.

## <컴파일 및 실행 방법>

### [컴파일]

1. *cd 디렉토리경로*
  - 위의 명령을 이용해 FTPServer.java와 FTPClient.java가 있는 디렉토리로 이동한다.
  - ex) 
2. *javac FTPClient.java*
  - 위의 명령을 실행하면 FTPClient.java가 컴파일 되어 .class파일들이 생성된다.
  - ex)

```
C:\Users\wtjtd\cnet>ls
FTPClient.java  FTPServer.java  t.txt  test.txt

C:\Users\wtjtd\cnet>javac  FTPClient.java

C:\Users\wtjtd\cnet>ls
FTPClient$1.class  FTPClient$3.class  FTPClient$ControlSend.class  FTPClient.java  t.txt
FTPClient$2.class  FTPClient$4.class  FTPClient.class              FTPServer.java  test.txt
```

### 3. *javac FTPServer.java*

- 위의 명령을 실행하면 FTPServer.java가 컴파일되어 .class파일들이 생성된다.
- ex)

```
C:\Users\wtjtd\cnet>ls
FTPClient$1.class  FTPClient$3.class  FTPClient$ControlSend.class  FTPClient.java  t.txt
FTPClient$2.class  FTPClient$4.class  FTPClient.class              FTPServer.java  test.txt

C:\Users\wtjtd\cnet>javac FTPServer.java

C:\Users\wtjtd\cnet>ls
FTPClient$1.class  FTPClient$ControlSend.class  FTPServer$2.class  FTPServer.class
FTPClient$2.class  FTPClient.class              FTPServer$3.class  FTPServer.java
FTPClient$3.class  FTPClient.java               FTPServer$4.class  t.txt
FTPClient$4.class  FTPServer$1.class            FTPServer$ControlSend.class  test.txt
```

## [실행]

### 1. *java FTPServer*

- 위 명령을 실행하면 FTPServer가 실행된다.

### 2. *java FTPClient*

- 위 명령을 실행하면 FTPClient가 실행된다.

### 3. 실행예시

#### 1) PUT

서버

```
C:\Users\wtjtd\cnet>java FTPServer
Request: CD ..
Response: 200 Moved to C:\Users\wtjtd

Request: TIMEOUT R1

Request: PUT t.txt
Request: 10903
Response: 200 Ready to receive
```

클라이언트

```
C:\Users\wtjtd\cnet>java FTPClient
CD ..
C:\Users\wtjtd
TIMEOUT R1
PUT t.txt
t.txt transferred / 10903bytes
0
1
2
3
4
2 acked
1 acked
4 acked
3 acked
0 timed out & retransmitted
0 acked
5
5 acked
6
6 acked
7
7 acked
8
9
8 acked
9 acked
10
10 acked
Completed...
```

### 3) GET

서버

```
Request: TIMEOUT R1  
Request: GET test.txt  
Response: 200 Containing 11259 bytes in total
```

클라이언트

```
0 1 2 3 4 5 6 7 8 9 10 11 Completed...  
TIMEOUT R1  
GET test.txt  
Received test.txt / 11259bytes  
1 2 3 4 0 5 6 7 8 9 10 11 Completed...
```