

이름: 김수빈

학번: 20192218

0. 제출:

- Sender: sender_20192218_김수빈.py
- Receiver: receiver_20192218_김수빈.py
- 사용하는 function: function_zip.py
- Function block description: function_block_desc_김수빈 20192218.doc /
function_block_desc_김수빈 20192218.pdf
(혹시 모를 호환 문제를 위해 두개다 첨부합니다.)
- 실행환경: MAC-OS

1. 입출력

- 입력: 없음, sender 파일의 input_stream 변수에 할당되어 있음

input_stream= '1111000011110000'

```
if __name__ == '__main__':  
    print("I'm Sender\n Let's Go Communication")  
    input_stream='1111000011110000'
```

- 출력:

성공 시, Application layer 는 SUCCESS 메세지를 받게 되고 다음과 같이 출력합니다.

[최종적으로 Sender가 받은 msg: SUCCESS]

2. 실행방법: 자신의 파이썬 버전에 맞게 python 명령어를 사용해주세요
터미널 두개를 이용하여
python(3) receiver_20192218_김수빈.py
python(3) sender_20192218_김수빈.py
코드를 실행한다.

이때, receiver 를 먼저 실행시킨 뒤 sender 를 실행한다.



3. 코드 소개

(1) 기본 환경

- 모든 layer 의 시작과 끝에 표시를 해준다. 처음 Scenario 1 은 layer 앞에 (S1)을 붙여주고, Scenario 2 는 (S2)를 붙임
- 파이프의 오브젝트는 *_conn 형식을 유지합니다.
- Layer 를 제외한 모든 function 은 function_zip.py 파일에 있습니다.
- Pipe 를 이용해 다른 layer 로 data 가 이동할 때, sending layer -> Receive layer 형식을 출력한다.
- Sender 는 p1~p6 총 6 개의 pipe 를 생성한다.
- Receiver 는 R1~R5 총 5 개의 pipe 를 생성한다.

① 사용 모듈

- import socket: sender 와 receiver 의 socket 통신을 위해 사용
- import time: Transport layer 에서 Stop-And-Wait 을 위해 사용
- import random: Transport layer 에서 Stop-And-Wait 을 위해 사용
- from multiprocessing import Process, Pipe
 - : 각 layer 를 process 로 여기면서 통신을 하기 위하여 사용, scenario1 과 2 에서 각 layer 는 양방향 통신을 하기 때문에 pipe 를 사용

<user-defined module>

- import function_zip as f: MLT-3, bit-stuffing/unstuffing, CSMA_CD, making_bitstream 을 위한 function 을 모두 저장

② socket 통신: sender 와 receiver 간의 소켓통신을 위하여 HOST,PORT 를 전역변수로 할당

(sender 와 receiver 는 같은 변수값을 가짐)

- HOST = '127.0.0.1'
- PORT = 9999

(2) Sender

① main : `if __name__ == '__main__'`

```

if __name__ == '__main__':
    print("I'm Sender\n Let's Go Communication")
    input_stream='1111000011110000'
    Sender_conn, App_conn = Pipe()
    print("\n\nUser -> Application")

    Sender_conn.send(input_stream)
    p1 = Process(target=Application_layer, args=(App_conn,))
    p1.start()
    out_stream=Sender_conn.recv()
    print("[최종적으로 Sender가 받은 msg: {}]".format(out_stream))

```

- *p1: Sender(main) 와 Application_layer 를 연결하는 pipe*
- 프로그램 시작 시, input stream 을 Application layer 에 보내고, 통신 종료 시, 성공 메세지 “Success”를 받음

[실행결과]

Scenario 1)

```

[→ 기말_20192218_김수빈 git:(main) ✘ python3 sender_20192218_김수빈.py
I'm Sender
Let's Go Communication

```

Scenario 2)

```

[최종적으로 Sender가 받은 msg: SUCCESS]
[→ 기말_20192218_김수빈 git:(main) ✘ ]

```

② Application layer: def Application_layer(conn)

```

def Application_layer(conn):
    print("(S1)<Start: Application_layer>")
    stream=conn.recv()
    #conn.close()
    #checking
    print("input_stream: ",stream)

    #Application -> Transport
    App_conn, Trans_conn = Pipe()
    print("Application -> Transport")

    print("Sending Stream [{}] to Transport_layer".format(stream))
    App_conn.send(stream)
    p2 = Process(target=Transport_layer, args=(Trans_conn,))
    p2.start()
    #p2.join()
    print("(S1)<End: Application_layer>\n\n")

    inv_stream=App_conn.recv()
    print("(S2)<Start: Application_layer>")

    if(inv_stream=="1111000011110000"): last_stream="SUCCESS"
    print("Receiving Stream [{} from Transport_layer".format(last_stream))
    print("Application -> user")
    conn.send(last_stream)
    print("(S2)<End: Application_layer>\n\n")

    conn.close()

```

- *p2: Application_layer 와 Transport_layer 를 연결하는 pipe*
- p2 를 시작한 뒤, Transport_layer 로 부터 bit stream 을 받으면, 그 bit stream 0| “1111000011110000”(“ACK”를 ASCII 로 변환한 뒤 binary 로 변환한 결과) 이면 성공이라고 판단하고 main 에 “SUCCESS” 전송

[실행결과]

Scenario 1)

```

user -> Application
(S1)<Start: Application_layer>
input_stream: 1111000011110000
Application -> Transport
Sending Stream [1111000011110000] to Transport_layer
(S1)<End: Application_layer>

```

Scenario 2)

```

(S2)<Start: Application_layer>
Receiving Stream [SUCCESS] from Transport_layer
Application -> user
(S2)<End: Application_layer>

```

③ Transport layer

Stop-and-wait protocol 을 이용한다.

```
def Transport_layer(conn):
    NOW_STATE=True
    #print("Application -> Transport")
    print("(S1)<Start: Transport_layer>")
    stream=conn.recv()

    #checking
    print("Transport - stream: ",stream)

    #Transport -> Network
    Trans_conn,Net_conn=Pipe()

    print("Transport -> Network")
    print("Sending Stream [{} ] to Network_layer by Stop and Wait".format(stream))

    #stop and wait
    #try:
    while True:
        try:
            if(random.randint(1,10))>8:
                print("LOST from SENDER")
                time.sleep(2)
                raise socket.timeout
                continue

        except socket.timeout:
            print("timeout! \n Resending a MSG")

        Trans_conn.send(stream)
        NOW_ACK='ACK'
        p3 = Process(target=Network_layer, args=(Net_conn,))
        p3.start()
        print("SENDING SUCCESS")
        print("WAITING ACK....")
        NOW_STATE=False #waiting 상태
        print("(S1)<End: Transport_layer>\n\n")

        inv_stream=Trans_conn.recv()
        if inv_stream=='1111000011110000': NOW_STATE=True

        print("(S2)<Start: Transport_layer>")
        print("Receiving Stream [{} ] from Network_layer".format(inv_stream))
        if NOW_STATE: #ACK라는 것을 받았으면 성공한 것
            print("SUCCESS TO RECEIVE ACK MSG")
            print("Transport -> Application")
            conn.send(inv_stream)
            print("(S2)<End: Transport_layer>\n\n")
            conn.close()
            break
```

Stop-and-wait protocol

- *p3: Transport_layer 와 Network_layer 를 연결하는 pipe*

<stop and wait>

random.randint 함수를 이용하여, 1/5 확률로 lost 가 발생하도록 구성한다. 만약 Lost 가 발생했을 시, timeout 을 발생시키고 다시 random.randint 함수를 다시 시행한다.

Scenario 1 을 통해 Network_layer 로 msg 를 보내고 나서는 다시 Scenario 2 에서 Network_layer 로 부터 ACK 를 받기 전까지 NOW_STATE 를 FALSE 로 놔두고 wait 합니다.

[실행결과]

Scenario 1)

```
(S1)<Start: Transport_layer>
Transport - stream: 1111000011110000
Transport -> Network
Sending Stream [1111000011110000] to Network_layer by Stop and Wait
SENDING SUCCESS
WAITING ACK....
(S1)<End: Transport_layer>
```

Scenario 2)

```
(S2)<Start: Transport_layer>
Receiving Stream [1111000011110000] from Network_layer
SUCCESS TO RECEIVE ACK MSG
Transport -> Application
(S2)<End: Transport_layer>
```

*

```
<Start: Transport_layer>
Transport - stream: 1111000011110000
Transport -> Network
Sending Stream [1111000011110000] by Stop and Wait
[ LOST from SENDER
  timeout!
  Resending a MSG ] ← LOST 가 발생하는 경우
SENDING SUCCESS
WAITING ACK....
<End: Transport_layer>
```

④ Network layer : `def Network_layer(conn)`
별 다른 기능을 하지 않고 **bypass** 한다.

```

def Network_layer(conn):
#bypass

    print("(S1)<Start: Network_layer>")
    stream=conn.recv()
    #checking
    print("Network - stream: ",stream)

    #Network -> Datalink
    Net_conn, Data_conn = Pipe()
    print("Network -> Datalink")

    print("Sending Stream [{}] to Datalink_layer".format(stream))
    Net_conn.send(stream)

    p4 = Process(target=Datalink_layer, args=(Data_conn,))
    p4.start()
    #p4.join()
    #conn.close()
    print("(S1)<End: Network_layer>\n\n")

    inv_stream=Net_conn.recv()

    print("(S2)<Start: Network_layer>")
    print("Receiving Stream [{}] from Datalink_layer".format(inv_stream))
    print("Network -> Transport")
    conn.send(inv_stream)
    print("(S2)<End: Network_layer>\n\n")

    conn.close()

```

- *p4: Application_layer 외 Transport_layer 를 연결하는 pipe*

[실행결과]

Scenario 1)

```

(S1)<Start: Network_layer>
Network - stream: 1111000011110000
Network -> Datalink
Sending Stream [1111000011110000] to Datalink_layer
(S1)<End: Network_layer>

```

Scenario 2)

```

(S2)<Start: Network_layer>
Receiving Stream [1111000011110000] from Datalink_layer
Network -> Transport
(S2)<End: Network_layer>

```

⑤ Datalink layer: `def Datalink_layer(conn)`

- ◆ DLC sub layer: Message bit stuffing
- ◆ Simple protocol 을 이용해 DLC 에서 MAC sub layer 로 보낸다.
*ideal 한 환경을 위해 무조건 전송하게 되는 방법을 선택하였습니다.
- ◆ Mac sub layer 에서 Physical layer 로 CSMA/CD 를 이용해 전송한다.
이때, ideal 한 환경을 위해 CSMA/CD 의 limit 을 100 으로 한다.

```

def Datalink_layer(conn):
    #print("Network -> Datalink")
    print("(S1)<Start: Datalink_layer>")
    stream=conn.recv()

    #checking
    print("Datalink - stream: ",stream)

    #bitstuffing
    aft_bitstuff_stream=f.bitstuffing(stream)
    print("DLC sub layer -> Mac sub layer (simple protocol) stream)",aft_bitstuff_stream)

    print("Sending to MAC sub layer using simple protocol")
    #Datalink(DLC-> MAC)
    print("DLC sub layer -> MAC sub layer")
    mac_stream=aft_bitstuff_stream

    #Datalink -> Physical
    Data_conn, Phy_conn = Pipe()
    print("Datalink -> Physical")

    print("Sending Stream [{} ] to Physical_layer".format(mac_stream))
    Data_conn.send(f.CSMA_CD(mac_stream))

    p5 = Process(target=Physical_layer, args=(Phy_conn,))
    p5.start()
    print("(S1)<End: Datalink_layer>\n\n")

    inv_stream=Data_conn.recv()

    print("(S2)<Start: Datalink_layer>")
    print("Receiving Stream [{} ] from Physical_layer".format(inv_stream))
    #bitunstuffing
    aft_bitunstuff_stream=f.bit_unstuffing_fun(stream)
    print("Aft Bit unstuffing:[{} ]".format(aft_bitunstuff_stream))

    print("Datalink -> Network by simple protocol")
    conn.send(aft_bitunstuff_stream)
    print("(S2)<End: Datalink_layer>\n\n")

    conn.close()

```

- *p5: Datalink_layer 와 Physical_layer 를 연결하는 pipe*
- Scenario 1 에서는 **bit_stuffing, simple_protocol,CSMA-CD** 를 실행하고
Scenario 2 에서는 **bit_unstuffing** 후 **simple_protocol** 을 통해
Network_layer 로 전송
[실행결과]
Scenario 1)

```
(S1)<Start: Datalink_layer>
Datalink - stream: 1111000011110000
Start bit_stuffing
Aft Bit stuffing:[1111000011110000]
Sending to MAC sub layer using simple protocol
DLC sub layer -> MAC sub layer
Datalink -> Physical
Sending Stream [1111000011110000] to Physical_layer
Start CSMA_CD
Data packet was sent, waiting for acknowledgement ...
Attemp 1 :
Transmission done...
The attemp was successful.
(S1)<End: Datalink_layer>
```

Scenario 2)

```
(S2)<Start: Datalink_layer>
Receiving Stream [100000110000111001011] from Physical_layer
Start bit_unstuffing
Aft Bit unstuffing:[1111000011110000]
Datalink -> Network by simple protocol
(S2)<End: Datalink_layer>
```

⑥ Physical layer: `def Physical_layer(conn)`

MLT-3/Reverse MLT-3 를 시행한다.

```

def Physical_layer(conn):
    #print("Datalink -> Physical")

    print("(S1)<Start: Physical_layer>")
    stream=conn.recv()

    #checking
    print("Physical - stream: ",stream)

    //MLT 3
    sending_stream=f.EN_MLT_3(stream)
    print("After MLT-3: {}".format(sending_stream))
    #Physical -> aft

    Phy_conn, send_conn = Pipe()
    print("Physical -> aft")

    print("Sending Stream {} to binder".format(sending_stream))
    Phy_conn.send(sending_stream)

    p6 = Process(target=binder, args=(send_conn,))
    p6.start()

    print("(S1)<End: Physical_layer>\n\n")

    inv_stream=Phy_conn.recv()

    print("(S2)<Start: Physical_layer>")
    print("Receiving Stream {} from binder".format(inv_stream))

    #Re-MLT-3
    sending_stream=f.DE_MLT_3(inv_stream)
    print("Aft Reverse MLT-3: {}".format(sending_stream))
    print("Physical -> Datalink")
    conn.send(sending_stream)
    print("(S2)<End: Physical_layer>\n\n")
    conn.close()

```

- *p6: Physical_layer 와 sender_layer(binder fun)를 연결하는 pipe*
- *실행 예시:*
Ex) bit_stream="1111000011110000"
After MLT-3 ="+0-00000+0-00000"

[실행결과]

Scenario 1)

```

(S1)<Start: Physical_layer>
Physical - stream: 1111000011110000
Start MLT-3 scheme
After MLT-3: [+0-00000+0-0000]
Physical -> aft
Sending Stream [+0-00000+0-0000] to binder
(S1)<End: Physical_layer>

Scenario 2)

(S2)<Start: Physical_layer>
Receiving Stream [++++++0-----0+000--0+] from binder
Start Reverse MLT-3 scheme
Aft Reverse MLT-3:[100000110000111001011]
Physical -> Datalink
(S2)<End: Physical_layer>

```

⑦ binder 함수: `def binder(conn)`

Sender ⇔ Receiver 간에 **소켓 통신**을 도와주는 함수이다.

```

def binder(conn):
    #print("Physical -> binder")
    print("(S1)<Start: binder>")
    stream=conn.recv()
    #checking
    print("binder - stream: ",stream)

    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    # connect함수로 접속을 한다.
    client_socket.connect((HOST, PORT))
    print("\nStart Socket\n")
    data = stream.encode();
    # 메시지 길이를 구한다.
    length = len(data)

    # server로 리틀 엔디언 형식으로 데이터 길이를 전송한다.
    client_socket.sendall(length.to_bytes(4, byteorder="little"))
    # 데이터를 전송한다.
    client_socket.sendall(data)

    print("\n\n===== start Scenario 2=====\\n\\n")

    data = client_socket.recv(4)
    # 데이터 길이는 리틀 엔디언 형식으로 int를 변환한다.
    length = int.from_bytes(data, "little")
    # 데이터 길이를 받는다.
    data = client_socket.recv(length)
    # 데이터를 수신한다.
    inv_stream = data.decode()
    print("(S2)-SENDING_TO_RECEIVER~")

    #inv_stream='010000010100001101001011'
    print("Receive from RECEIVER: ",inv_stream)
    conn.send(inv_stream)
    print("(S2)<End: binder>")

    conn.close()

```

소켓 통신 부분

- 주소를 연결하고 receiver로 data를 보내고 receiver로 부터 data를 받는 모든 소켓 기능을 담당한다.

- Scenario 2에서 stream 을 받은 뒤, Physical_layer 로 bit_stuffing 을 보내줌

[실행결과]

Scenario 1)

```
(S1)<Start: binder>
binder - stream: +0-00000+0-00000
```

Start Socket

Scenario 2)

```
===== start Scenario 2=====
```

```
(S2)<SENDING TO RECEIVER>
Receive from RECEIVER: ++++++0-----0+000--0+
(S2)<End: binder>
```

(3) Receiver

```
① main : if __name__ == '__main__'
```

Sender ⇔ Receiver 간에 **소켓 통신**을 담당한다.

Receiver 의 시작과 끝이 되는 함수이다.

```

if __name__ == '__main__':
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    # 소켓 레벨과 데이터 형태를 설정한다.
    server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    # 서버는 복수 ip를 사용하는 pc의 경우는 ip를 지정하고 그렇지 않으면 None이 아닌 ''로 설정한다.
    # 포트는 pc내에서 비어있는 포트를 사용한다. cmd에서 netstat -an | find "LISTEN"으로 확인할 수 있다.

    server_socket.bind(('', PORT))
    # server 설정이 완료되면 listen()를 시작한다.
    server_socket.listen()
    try:
        client_socket, addr = server_socket.accept()
        print("I'm RECEIVER! LETs GO!\n\n")
        # socket의 recv함수는 연결된 소켓으로부터 데이터를 받을 대기하는 함수입니다. 최초 4바이트를 대기합니다.
        data = client_socket.recv(4)
        # 최초 4바이트는 전송할 데이터의 크기이다. 그 크기는 little 엔디언으로 byte에서 int형식으로 변환한다.
        length = int.from_bytes(data, "little")
        # 다시 데이터를 수신한다.
        data = client_socket.recv(length)
        # 수신된 데이터를 str형식으로 decode한다.
        input_stream = data.decode()
        """
        여기서 데이터를 전송하는 곳입니다.
        """

```

소켓으로 Sender로 부터 bit stream 받기

```

print("Received stream: ",input_stream)

# Receiver -> Physical
Rec_conn, Phy_conn = Pipe()
print("\n\n(S1) Receiver Start!")
print("Receiver -> Physical")

print("Sending Stream [{}] to Physical".format(input_stream))
Rec_conn.send(input_stream)

R1 = Process(target=Physical_layer, args=(Phy_conn,))
R1.start()

=====
```

```

inv_stream=Rec_conn.recv()
print("\n\n<SENDING TO SENDER>")
print("{} stream을 sender에게 전송합니다.".format(inv_stream))
data = inv_stream.encode()
# 메시지 길이를 구한다.
length = len(data)
# server로 리틀 엔디언 형식으로 데이터 길이를 전송한다.
client_socket.sendall(length.to_bytes(4, byteorder="little"))
# 데이터를 전송한다.
client_socket.sendall(data)
print("(S2) Receiver End\n\n")
```

```

except:
    print("server")
finally:
    # 에러가 발생하면 서버 소켓을 닫는다.
    server_socket.close()
```

- R1: main 과 Physical_layer 를 연결하는 pipe

[실행결과]

Scenario 1)

↳ READER LINE

→ 기말_20192218_김수빈 git:(main) ✘ python3 receiver_20192218_김수빈.py
I'm RECEIVER! LETs GO!

Received stream: +0-00000+0-00000

Scenario 2)

```

<SENDING TO SENDER>
+++++0-----0+000--0+ stream을 sender에게 전송합니다 .
(S2) Receiver End

```

② Physical layer: `def Physical_layer(conn)`

MLT-3/Reverse MLT-3 를 시행한다.

```

def Physical_layer(conn):
    #print("Receiver(main) -> Physical")

    print("(S1)<Start: Physical_layer>")
    stream=conn.recv()

    #checking
    print("Physical - stream: ", stream)

    #Re-MLT-3
    sending_stream=f.DE_MLT_3(stream)
    print("Aft Reverse MLT-3:[{}]".format(sending_stream))
    #Physical -> aft
    Phy_conn, Data_conn = Pipe()
    print("Physical -> Network")

    print("Sending Stream [{}] to Datalink_layer".format(sending_stream))

    Phy_conn.send(sending_stream)

    R2 = Process(target=Datalink_layer, args=(Data_conn,))
    R2.start()
    #p4.join()
    #conn.close()
    print("(S1)<End: Physical_layer>\n\n")

    inv_stream=Phy_conn.recv()

    print("(S2)<Start: Physical_layer>")
    print("Receiving Stream [{}] from Datalink_layer".format(inv_stream))
    print("Physical -> binder")

    #MLT-3
    sending_stream=f.EN_MLT_3(inv_stream)
    print("Aft MLT-3:[{}]".format(sending_stream))

    conn.send(sending_stream)
    print("(S2)<End: Physical_layer>\n\n")
    conn.close()

```

- *R2: Physical_layer 와 Datalink_layer 를 연결하는 pipe*

[실행결과]

Scenario 1)

```

(S1)<Start: Physical_layer>
Physical - stream: +0-00000+0-00000
Start Reverse MLT-3 scheme
Aft Reverse MLT-3:[1111000011110000]
Physical -> Network
Sending Stream [1111000011110000] to Datalink_layer
(S1)<End: Physical_layer>

```

Scenario 2)

```
(S2)<Start: Physical_layer>
Receiving Stream [100000110000111001011] from Datalink_layer
Physical -> Receiver
Start MLT-3 scheme
Aft MLT-3:[+++++0-----0+000--0+]
(S2)<End: Physical_layer>
```

③ Datalink layer : `def Datalink_layer(conn)`

- ◆ DLC sub layer: Message bit stuffing
- ◆ Simple protocol 을 이용해 DLC 에서 MAC sub layer 로 보낸다.
*ideal 한 환경을 위해 무조건 전송하게 되는 방법을 선택하였습니다.
- ◆ Mac sub layer 에서 Physical layer 로 CSMA/CD 를 이용해 전송한다.
이때, ideal 한 환경을 위해 CSMA/CD 의 limit 을 100 으로 한다.

```
def Datalink_layer(conn):
    #print("Physical -> Datalink")
    print("(S1)<Start: Datalink_layer>")
    stream=conn.recv()

    #checking
    print("Datalink - stream: ",stream)

    #bitunstuffing
    aft_bitunstuff_stream=f.bit_unstuffing_fun(stream)
    print("Aft Bit unstuffing:[{}]" .format(aft_bitunstuff_stream))

    #Datalink(DLC-> MAC)
    mac_stream=aft_bitunstuff_stream

    #Datalink -> Network
    Data_conn, Net_conn = Pipe()
    print("Network -> Datalink using simple protocol")

    print("Sending Stream [{} ] to Network_layer".format(mac_stream))
    Data_conn.send(mac_stream)

    R3 = Process(target=Network_layer, args=(Net_conn,))
    R3.start()
    print("(S1)<End: Datalink_layer>\n\n")

    inv_stream=Data_conn.recv()

    print("(S2)<Start: Datalink_layer>")
    print("Receiving Stream [{} ] from Network_layer".format(inv_stream))
    #bitstuffing
    aft_bitstuff_stream=f.bit_stuffing_fun(inv_stream)
    print("Aft Bit stuffing:[{}]" .format(aft_bitstuff_stream))

    print("Sending to MAC sub layer using simple protocol")
    #Datalink(DLC-> MAC)
    print("DLC sub layer -> MAC sub layer")
    mac_stream=aft_bitstuff_stream

    print("Datalink -> Physical")
    print("Sending Stream [{} ] to Physical_layer".format(mac_stream))
    conn.send(f.CSMA_CD(mac_stream))
    print("(S2)<End: Datalink_layer>\n\n")

    conn.close()
```

- R3: Datalink_layer 와 Network_layer 를 연결하는 pipe

- Scenario 1 에서는 **bit_unstuffing** 후 **simple_protocol** 을 통해 Network_layer 로 전송하고
- Scenario 2 에서는 **bit_stuffing, simple_protocol,CSMA-CD** 를 실행한다.

[실행결과]

Scenario 1)

```
(S1)<Start: Datalink_layer>
Datalink - stream: 1111000011110000
Start bit_unstuffing
Aft Bit unstuffing:[1111000011110000]
Network -> Datalink using simple protocol
Sending Stream [1111000011110000] to Network_layer
(S1)<End: Datalink_layer>
```

Scenario 2)

```
(S2)<Start: Datalink_layer>
Receiving Stream [100000110000111001011] from Network_layer
Start bit_stuffing
Aft Bit stuffing:[100000110000111001011]
Sending to MAC sub layer using simple protocol
DLC sub layer -> MAC sub layer
Datalink -> Physical
Sending Stream [100000110000111001011] to Physical_layer
Start CSMA_CD
Data packet was sent, waiting for acknowledgement ...
Attemp 1 :
Transmission done...
The attemp was successful.
(S2)<End: Datalink_layer>
```

④ Network layer : def Network_layer(conn)

별 다른 기능을 하지 않고 **bypass** 한다.

```

def Network_layer(conn):
#bypass
    #print("Datalink -> Network")

    print("(S1)<Start: Network_layer>")
    stream=conn.recv()
    #checking
    print("Network - stream: ",stream)

    #Network -> Datalink
    Net_conn, Trans_conn = Pipe()
    print("Network -> Transport")

    print("Sending Stream [{}] to Transport_layer".format(stream))
    Net_conn.send(stream)

    R4 = Process(target=Transport_layer, args=(Trans_conn,))
    R4.start()

    print("(S1)<End: Network_layer>\n\n")

    inv_stream=Net_conn.recv()

    print("(S2)<Start: Network_layer>")
    print("Receiving Stream [{}] from Transport_layer".format(inv_stream))
    print("Network -> Datalink")
    conn.send(inv_stream)
    print("(S2)<End: Network_layer>\n\n")

    conn.close()

```

- R4: Network_layer 외 Transport_layer 를 연결하는 pipe

[실행결과]

Scenario 1)

```

(S1)<Start: Network_layer>
Network - stream: 1111000011110000
Network -> Transport
Sending Stream [1111000011110000] to Transport_layer
(S1)<End: Network_layer>

```

Scenario 2)

```

(S2)<Start: Network_layer>
Receiving Stream [100000110000111001011] from Transport_layer
Network -> Datalink
(S2)<End: Network_layer>

```

⑤ Transport layer:

Stop and wait 통신함수

```

def Transport_layer(conn):
    NOW_STATE=True
    #print("Network -> Transport")
    print("(S1)<Start: Transport_layer>")
    #try:
    while True:
        stream=conn.recv()

        #checking
        print("Transport - stream: ",stream)
        print("MAke ACK MSG")
        ACK_MSG='ACK'
        NOW_STATE=False
        #Transport -> App
        Trans_conn,App_conn=Pipe()
        print("Transport -> Application")
        print("Sending Stream [{} ] to Application_layer".format(stream))
        Trans_conn.send(stream)
        R5 = Process(target=Application_layer, args=(App_conn,))
        R5.start()
        print("(S1)<End: Transport_layer>\n\n")

        inv_stream=Trans_conn.recv()
        print("===== start Scenario 2 =====")
        print("(S2)<Start: Transport_layer>")

        #통신 성공하면 Application은 SUCCESS 라고 msg를 보내기로 함
        if inv_stream=="SUCCESS":
            NOW_STATE=False
            if (NOW_STATE==False) and (ACK_MSG=='ACK'):
                sending_stream=f.making_bitstream('ACK')
                print("Sending Stream [{} ] to Network_layer".format(sending_stream))
                print("Transport -> Network")
                conn.send(sending_stream)
                print("(S2)<END: Transport_layer>\n\n")
                NOW_STATE=True
                conn.close()
                break

```

- R5: Application_layer 와 Transport_layer 를 연결하는 pipe
- Scenario 2에서 통신에 성공하면 Network layer에 ACK를 ASCII 코드로 바꿔 생성한 bit stream을 보낸다. (making_bitstream(ascii_str) 함수 사용)

[실행결과]

Scenario 1)

```

(S1)<Start: Transport_layer>
Transport - stream: 1111000011110000
MAke ACK MSG
Transport -> Application
Sending Stream [1111000011110000] to Application_layer
(S1)<End: Transport_layer>

```

Scenario 2)

```

===== start Scenario 2 =====
(S2)<Start: Transport_layer>
Sending Stream [100000110000111001011] to Network_layer
Transport -> Network
(S2)<END: Transport_layer>

```

⑥ Application layer : def Application_layer(conn)

Bit-stream 이 성공적으로 보내졌는지 확인하고 수신 확인한다면 생성한 ACK 메세지를 보내는 Scenario 2 를 시작한다.

```
def Application_layer(conn):  
  
    print("(S1)<Start: Application_layer>")  
    stream=conn.recv()  
    #conn.close()  
    #checking  
    print("Received_stream: ",stream)  
  
    print("Check stream !=====")  
  
    if stream==right_stream:  
        print("SUCCESS")  
        conn.send("SUCCESS")  
    else: pass  
    print("(S1)<End: Application_layer>\n\n")  
    conn.close()
```

[실행결과]

Scenario 1)

```
(S1)<Start: Application_layer>  
Received_stream: 1111000011110000  
Check stream !=====  
SUCCESS  
(S1)<End: Application_layer>
```

(4) Function_zip : layer 를 제외한 함수들을 저장하기 위해 만든 모듈

① def bit_stuffing_fun(bit_stream): 문자열로 저장된 bit stream 을 bit stuffing 한다.

Ex) 1111000011110000 -> 1111000011110000

② def bit_unstuffing_fun(bit_stream): bit stuffing 된 bit stream 을 bit unstuffing 한다.

③ def EN_MLT_3(bit_stream): MLT-3 을 적용한다.

④ def DE_MLT_3(bit_stream): Reverse- MLT-3 를 한다.

⑤ def CSMA_CD(input_stream): CSMA-CD 를 실행한다.

⑥ def making_bitstream(ascii_str): String 을 받아 ascii 로 바꿔주는 함수이다.

4. 동작 과정

(1) 터미널 두개를 연다

(2) 한 쪽엔 python(3) receiver_20192218_김수빈.py

다른 한 쪽에는 python(3) sender_20192218_김수빈.py 를 입력한다.

(3) python(3) receiver_20192218_김수빈.py 를 입력한 터미널 먼저 실행하고 그 다음 sender 가 입력된 터미널을 실행한다.

[실행결과]

The screenshot shows two terminal windows side-by-side. Both windows have tabs at the top labeled ".192218_김수빈".

Left Terminal (Sender):

```
+ 기발_20192218_김수빈 git:(main)* python3 receiver_20192218_김수빈.py
I'm RECEIVER! LET's GO!
Received stream: +0-00000+0-00000
(S1) Receiver Start!
Receiver -> Physical
Sending Stream [+0-00000+0-00000] to Physical
(S1)<Start: Physical_layer>
Physical - stream: +0-00000+0-00000
Start Reverse MLT-3 scheme
Aft Reverse MLT-3: scheme
Physical -> Network
Sending Stream [111000011110000] to Datalink_layer
(S1)<End: Physical_layer>

(S1)<Start: Datalink_layer>
Datalink - stream: 111000011110000
Start bit_unstuffing:[111000011110000]
Network -> Datalink using simple protocol
Sending Stream [111000011110000] to Network_layer
(S1)<End: Datalink_layer>

(S1)<Start: Network_layer>
Network - stream: 111000011110000
Network -> Transport
Sending Stream [111000011110000] to Transport_layer
(S1)<End: Network_layer>

(S1)<Start: Transport_layer>
Transport - stream: 111000011110000
Make ACK MSG
Transport -> Application
Sending Stream [111000011110000] to Application_layer
(S1)<End: Transport_layer>

(S1)<Start: Application_layer>
Received stream: 111000011110000
Check stream !=====
SUCCESS
(S1)<End: Application_layer>

===== start Scenario 2 =====
(S2)<Start: Transport_layer>
Sending Stream [10000011000111001011] to Network_layer
Transport -> Physical
(S2)<End: Transport_layer>
(S2)<Start: Network_layer>
Receiving Stream [10000011000111001011] from Transport_layer
Network -> Datalink
(S2)<End: Network_layer>
```

Right Terminal (Receiver):

```
+ 기발_20192218_김수빈 git:(main)* python3 sender_20192218_김수빈.py
[최종적으로 Sender가 받은 msg: SUCCESS]
I'm Sender
Let's Go Communication
user -> Application
(S1)<Start: Application_layer>
input - stream: 111000011110000
Application -> Transport
Sending Stream [111000011110000] to Transport_layer
(S1)<End: Application_layer>

(S1)<Start: Transport_layer>
Transport - stream: 111000011110000
Transport -> Network
Sending Stream [111000011110000] to Network_layer by Stop and Wait
SENDING SUCCESS
WAITING ACK...
(S1)<End: Transport_layer>

(S1)<Start: Network_layer>
Network - stream: 111000011110000
Network -> Datalink
Sending Stream [111000011110000] to Datalink_layer
(S1)<End: Network_layer>

(S1)<Start: Datalink_layer>
Datalink - stream: 111000011110000
Start bit_stuffing:[111000011110000]
After Bit stuffing:[111000011110000]
Sending to MAC sub layer using simple protocol
DLC sub layer -> MAC sub layer
Datalink -> Physical
Sending Stream [111000011110000] to Physical_layer
Start CSMA_CD
Data packet was sent, waiting for acknowledgement ...
Attempt 1
Transmission done...
The attempt was successful.
(S1)<End: Datalink_layer>

(S1)<Start: Physical_layer>
Physical - stream: 111000011110000
Start MLT-3 scheme
After MLT-3: [+0-00000+0-00000]
Physical -> binder
Sending Stream [+0-00000+0-00000] to binder
(S1)<End: Physical_layer>

(S1)<Start: binder>
binder - stream: +0-00000+0-00000
Start Socket
```