

# Project1

---

통신네트워크

## ■ Introduction

- 목적
- 제출
- rdt1.0 TCP/IP에서 사용되는 주요 함수

## ■ rdt3.0 TCP/IP Client(Sender)

- Client (Sender)
- client.c 에서 사용되는 주요 함수
- mySender.c

## ■ rdt3.0 TCP/IP Server(Receiver)

- Server (Receiver)
- server.c 에서 사용되는 주요 함수
- myReceiver.c

## ■ Simulation

- 컴파일 및 실행
- 실행 결과 예시

# Introduction

---

## ■ 목적

- rdt1.0 TCP/IP에 reliable data transmission을 위한 기능들을 추가하여 rdt3.0 TCP/IP를 구현해보자
  - **rdt1.0 TCP/IP: Project0의 TCP/IP**
    - Reliable Channel을 가정하였다. (Channel에 의한 packet loss와 error 둘 다 고려 x)
  - **rdt3.0 TCP/IP: Project1의 TCP/IP**
    - channel에 의한 packet loss 고려하여 이를 극복하기 위한 기능들이 추가되었다.
    - Channel에 의한 packet error는 sender 부분은 고려하지 않고 receiver 부분에서만 고려하였다.
    - Stop-and-Wait protocol로써 수업시간에 배운 rdt3.0의 기능들이 구현되어 있다.
      - » ARQ
      - » Indicator(수업시간에 배운 sequence number의 역할)
      - » Timeout
  - mySender.c, myReceiver.c 파일의 ? 부분을 알맞게 채워 넣으면 rdt3.0 TCP/IP 구현 가능

## - 참고 사항

- Project1의 rdt3.0 TCP/IP는 표준에 맞게 모든 기능이 구현된(교수님께서 수업시간에 다루신) TCP가 아닌 rdt3.0 기능을 최대한 간단하게 구현하기 위한 교육용 TCP 소켓 프로그래밍입니다.
- rdt3.0 TCP/IP의 구현은 수업시간에 배운 rdt3.0과는 구현 상에서 차이가 있습니다.
- Project0에서는 server.h, client.h 파일이 존재했지만, Project1에서는 해당 파일 안의 코드들을 다른 파일들에 구현해서 server.h, client.h 파일 없이 컴파일 및 실행됩니다.

## ■ 제출

- 제출 파일 : 압축폴더 (Project1\_학번\_이름.zip)로 제출
  - PDF (Project1\_학번\_이름.pdf)
    - 실행 화면 첨부
      - » Data packet이 제대로 전송되고 ACK도 제대로 전송된 경우
      - » Data packet이 Loss 되어 Timeout 이후 재전송된 경우
      - » Data packet이 제대로 전송되었지만 ACK이 Loss되어 Data packet이 Timeout 이후 재전송되고 재전송 역시 Loss없이 제대로 전송되어 수신단에서 Data packet이 Duplicate 된 경우
      - » Data packet이 제대로 전송되었지만 ACK error가 발생하여 잘못된 indicator를 가진 ACK이 전송되어 잘못된 ACK으로 판단하여 Data packet에 대한 ACK을 기다리다 Data packet이 Timeout 이후 재전송되고 재전송 역시 Loss 없이 제대로 전송되어 수신단에서 Data packet이 Duplicate된 경우
    - 실행 화면들에 대한 간단한 설명
    - 이 protocol에 적용된 stop-and-wait protocol의 특징들에 대해 간단한 설명
    - rdt 1.0에 비해 어떤 점이 추가되었는지 간단한 설명
  - Client 소스 파일 : cmain.c, client.c, mySender.c
    - mySender는 뒤에 설명하는 동작과 부합하도록 완성되어야 함
  - Server 소스 파일 : smain.c, server.c, myReceiver.c
    - myReceiver는 뒤에 설명하는 동작과 부합하도록 완성되어야 함
  - 실행파일 : s, c
  - 사진파일 : src.jpg, dst.jpg
- 제출 기한 : 2022.11.2 (수) 23:59까지

# Introduction

## ■ rdt1.0 TCP/IP에서 사용되는 주요 함수

### – server.c

#### • serverOpen

- **socket**: Socket Creation
- **bind**: Socket Binding
- **listen**: Socket Listen
- **accept**: Socket Accept
- **rcvPacket**

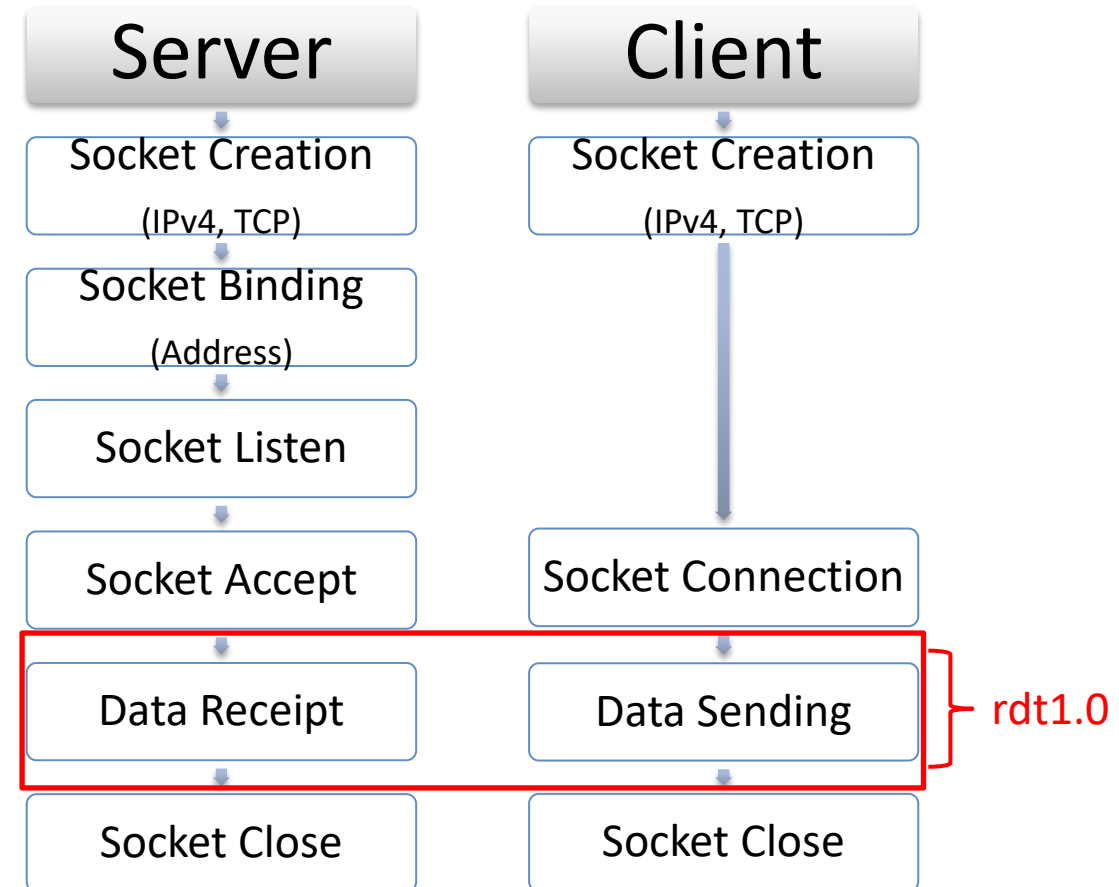
- reliable한 data 전송 위한  
어떤 기능도 구현 x
- » **read**: Data Receipt
  - » **toApp**: Send received packet to Application layer
  - **close**: Socket close

### – client.c

#### • clientOpen

- **socket**: Socket Creation
- **connect**: Socket Connection
- **write**: Data sending
- **close**: Socket close

reliable한 data 전송 위한  
어떤 기능도 구현 x



## **rdt3.0 TCP/IP Client (Sender)**

---

## ■ Client (Sender)

- 주어진 src.jpg 파일을 4096 Bytes 단위로 쪼개서 각 Data packet을 server로 전송
- Data Packet 구성 요소
  - 4096 Bytes 크기의 버퍼
  - 실질적인 데이터의 크기
  - Data packet의 sequence 번호
  - 송신 indicator (0 or 1)
  - 마지막 전송 패킷인지를 나타내는 flag
- 확률적으로 전송되는 Data Packet이 Loss 되는 상황 고려
- Timeout이 발생시 재전송
- Receiver가 보내는 ACK packet을 수신
  - 송신한 Data Packet에 대한 ACK packet인지는 indicator로 판단
- 다음 Data packet은 1 증가된 sequence 번호와 토글( $0 \leftrightarrow 1$ )된 indicator 값을 가짐
- 마지막 Data packet (4096 Bytes 이하) 전송 시, 마지막 Data packet임을 명시
- 마지막 Data packet에 대한 ACK을 수신하면 종료

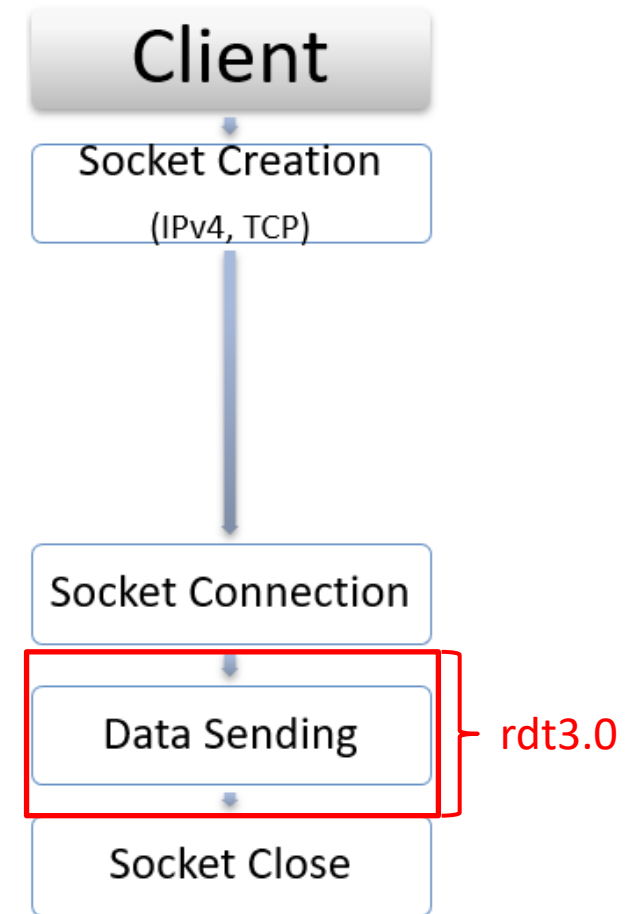


# rdt3.0 TCP/IP Client (Sender)

## ■ client.c 에서 사용되는 주요 함수

### – clientOpen

- **socket**: Socket Creation
- **connect**: Socket Connection
- **mySender**: mySender.c에 구현되어 있다.
  - **rdtSend**
    - » **udtSend**
      - **write**: Data Sending
    - » **isACK**
      - **read**: ACK Receipt
  - **close**: Socket Close



# rdt3.0 TCP/IP Client (Sender)

## ■ mySender.c

### – mySender

- 주어진 파일을 4096 bytes 단위 (BUFSIZE) 로 쪼개어 Data packet을 서버로 전송
- 사용 변수들
  - fSize : 주어진 파일 중 Data packet들을 보내고 남은 Data packet들의 전체 크기
    - » fSize를 이용하여 보내야 할 Data packet이 남았는지 판단
    - » fSize를 이용하여 현재 보낼 Data packet이 마지막 Data packet인지 판단
      - Hint. 기존 Data packet의 크기와 비교
  - sentSize : 현재까지 보낸 Data packet들의 전체 크기
  - dataSeq : 현재 보내는 Data packet의 sequence number
  - Indicator : 현재 보내는 Data packet의 indicator

```
// initialize sndPkt.
sndPkt.dataSeq = ?
sndPkt.indicator = ?
sndPkt.isFinish = ?

// send the file to the server.
while ( ? )
{
    // less than or equal to maximum length (BUFSIZE)
    sndPkt.dataSize = ?

    // if packet to send is last one, set flag of finish.
    sndPkt.isFinish = ?

    // read data from file to packet data.
    fread(sndPkt.data, sndPkt.dataSize, 1, fp);

    // send packet by reliable data transmission.
    rdtSend(sock, sndPkt);

    fSize = ?;
    sentSize = ?;
    sndPkt.dataSeq = ?;
    sndPkt.indicator = ?;
    usleep(500000);
}
```

# rdt3.0 TCP/IP Client (Sender)

## ■ mySender.c

### – rdtSend

- udtSend함수로 Data packet을 전송
- isACK 함수로 ACK이 수신되었는지 여부 판단
- Timeout이 될 때까지 ACK을 수신하지 못하면, 보낸 Packet이 Loss 되었다 판단하고 재전송
  - Timeout은 isACK 함수에서 구현

```
void rdtSend(int sock, Packet pkt)
{
    udtSend(sock, pkt);
    // retransmit packet when Timeout occurs

    while( ? )
    {
        printf("TIMEOUT => resent ");
        ?
    }

    usleep(500000);
    printf("ACK received\n\n");
}
```

# rdt3.0 TCP/IP Client (Sender)

## ■ mySender.c

### – isACK

- Receiver가 보낸 ACK이 수신 될 때까지 ACK을 계속 읽으려고 시도하면서 기다림
  - 본 함수에 구현되어 있는 timer 기능을 이용하여 구현
- ACK 수신 시, 지금 받은 ACK이 현재 보낸 Data packet에 대한 ACK이 맞으면 true 반환
  - indicator로 판단
  - 그렇지 않으면 false 반환

```
bool isACK(int sock, int idc)
{
    int timer_count = 0;
    Packet ACK;
    memset(&ACK, -1, sizeof(ACK));

    // timer loop
    while (timer_count++ < TIMEOUT)
    {
        ?; //Use read()

        if ( ? )
            return true;
    }

    return false;
}
```

# rdt3.0 TCP/IP Client (Sender)

## ■ mySender.c

### – udtSend

- Data packet을 실제로 전송하기 위한 함수
- Channel에 의한 packet loss를 의도적으로 구현
  - 현재 Sender와 Receiver 사이의 채널은 loss가 없는 채널(동일 Host)이므로 시뮬레이션을 위해 의도적으로 packet loss를 구현
- Packet loss가 발생한 경우, Packet을 server에 안 보냄으로써 packet loss 상황을 구현
- Packet loss가 없는 경우는 정상적으로 Packet을 전송

```
void udtSend(int sock, Packet pkt)
{
    // make intentional error
    int err_idc = rand() % ERR_PARAM;

    if (0 == err_idc)        // error occurs (the probability of the packet loss is 1/7)
    {
        printf("%i Bytes data (seq: %i, idc: %i) sent (LOSS)\n",
               pkt.dataSize, pkt.dataSeq, pkt.indicator);
    }
    else
    {
        ?                //Use write()
        printf("%i Bytes data (seq: %i, idc: %i) sent (SUCCESS)\n",
               pkt.dataSize, pkt.dataSeq, pkt.indicator);
    }
}
```

## **rdt3.0 TCP/IP Server (Receiver)**

---

## ■ Server (Receiver)

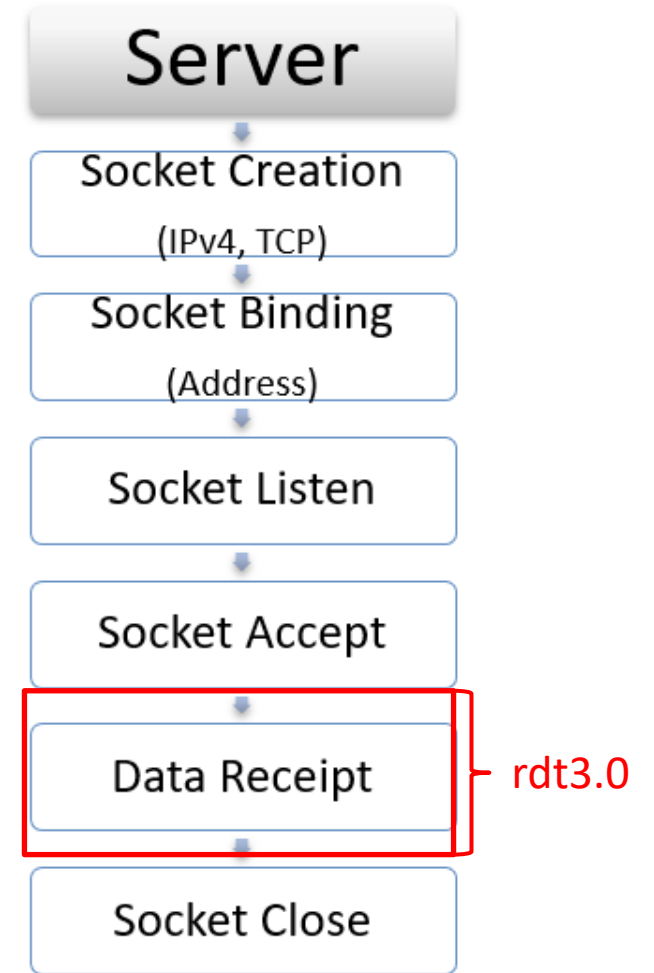
- Client로부터 Data packet 수신 받고 제대로 받았으면 ACK packet을 보내준다.
  - 제대로 받았는지 여부는 기다리는 Data packet의 indicator 값과 수신한 Data packet의 indicator 값을 비교
    - 기다리는 indicator 값이 0인데, 수신한 Data packet의 indicator 값이 0인 경우
      - » ACK 0 송신, 수신한 Data packet의 데이터 값을 dst.jpg에 쓰기
    - 기다리는 indicator 값이 1인데, 수신한 패킷의 indicator 값이 0인 경우
      - » ACK 0 송신, 수신한 Data packet의 데이터 사용 X
    - 기다리는 indicator 값이 0인데, 수신한 패킷의 indicator 값이 1인 경우
      - » ACK 1 송신, 수신한 Data packet의 데이터 사용 X
    - 기다리는 indicator 값이 1인데, 수신한 패킷의 indicator 값이 1인 경우
      - » ACK 1 송신, 수신한 Data packet의 데이터 값을 dst.jpg에 쓰기
- ACK packet이 전송되는 도중에 특정 상황 고려
  - ACK packet이 packet loss가 발생하는 상황
  - Packet error가 발생하여 실제 indicator와 반대의 indicator 값을 가지는 ACK packet 전송
    - Ex. 받은 data packet의 indicator는 1이지만 error가 발생하여 indicator가 0인 ACK packet을 전송한다.
    - Sender에서는 잘못된 ACK을 받았다고 판단하여 제대로 된 ACK을 받을 때까지 기다리고 timeout이 되면 data packet을 재전송한다.
- 마지막 Data packet을 수신하면 ACK packet을 전송하고 종료

# rdt3.0 TCP/IP Server (Receiver)

## ■ server.c 에서 사용되는 주요 함수

### – serverOpen

- **socket**: Socket Creation
- **bind**: Socket Binding
- **listen**: Socket Listen
- **accept**: Socket Accept
- **myReceiver**: myReceiver.c에 구현되어 있다.
  - **read**: Data Receipt
  - **rdtACK**
    - » **sendACKwithLoss**
      - **write**: Ack Sending
    - » **toApp**: Send received packet to Application layer
- **close**: Socket Close





# rdt3.0 TCP/IP Server (Receiver)

## ■ myReceiver.c

### – myReceiver

- read를 통해 data packet을 읽는다.
- rdtACK을 통해 data packet에 대한 ACK 전송
- Data packet이 마지막 data packet이면 data packet의 수신을 멈춘다.
  - Hint. isFinish 변수 사용

```
void myReceiver(int sock)
{
    Packet rcvPkt;
    int idcRcv = 0;
    int rcvedSize = 0;

    while (1)
    {
        // read data transferred from client.
        assert(read(sock, &rcvPkt, sizeof(rcvPkt)));

        usleep(250000);

        printf("%i Bytes data (seq: %i, idc: %i) received\n",
               rcvPkt.dataSize, rcvPkt.dataSeq, rcvPkt.indicator);
        rdtACK( ? );

        // stop reception if finish flag is set.
        if ( ? )
            break;
    }
    printf("The file (%i Bytes) has been received.\n", rcvedSize);
}
```

# rdt3.0 TCP/IP Server (Receiver)

## ■ myReceiver.c

### – rdtACK

- 만약 현재 받은 Data packet이 기다리던 Data packet이면 sendACKwithLoss와 toAPP 실행
  - sendACKwithLoss 함수 : 받은 Data packet에 대한 ACK을 보낸다.
  - toApp 함수 : 받은 Data packet을 Application layer로 보낸다.
  - ACK 전송 뒤, 다음에 받을 Data packet의 indicator로 토글 (1 ↔ 0)한다.
- 그렇지 않으면 sendACKwithLoss만 실행하여 이전 indicator를 가진 ACK을 전송

```
void rdtACK(int sock, Packet pkt, int* idc, int* rcvedSize)
{
    // indicator for that server is waiting == indicator from client
    if ( ? )
    {
        printf("Acceptable Data\n");
        usleep(250000);

        // send ACK with indicator.
        sendACKwithLoss( ? );

        // deliver data to app layer.
        toApp(pkt);

        *rcvedSize = ? // accumulate
        *idc = ?      // toggle idc
    }
    else // different indicators
    {
        printf("Non-acceptable Data => Data will be thrown away.\n");
        usleep(250000);

        // send ACK with indicator
        sendACKwithLoss(?);
    }
}
```

# rdt3.0 TCP/IP Server (Receiver)

## ■ myReceiver.c

### – sendACKwithLoss

- ACK packet을 실제로 전송하기 위한 함수
- Channel에 의한 packet loss와 error를 의도적으로 구현
  - Sender의 udtSend에서 구현한 방식 참고
  - Packet loss가 발생했을 때는 ACK packet을 전송하지 않는다.
  - Error가 발생했을 때는 실제 indicator와 반대의 indicator 값을 가지는 ACK packet을 전송한다.
- rdtACK에서 지정한 indicator값을 실제 ACK packet의 indicator값으로 지정

```
void sendACKwithLoss(int sock, int idc)
{
    Packet ACKPkt;
    // make intentional error
    int err_idc = rand() % LOSS_PARAM;
    int err_idc2 = rand() % ERR_PARAM;

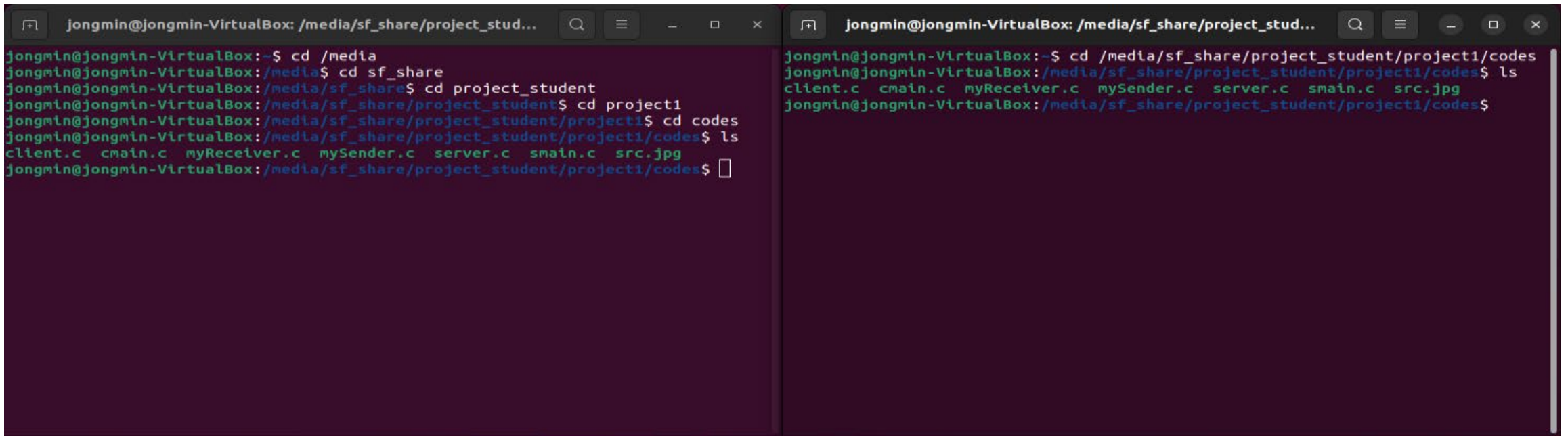
    if (0 == err_idc)
        //ACK is loss
        printf("ACK sent (LOSS)\n\n");
    else if (0 == err_idc2)
        //ACK is transmitted with wrong indicator
        {
            printf("ACK sent (ERROR)\n\n");
            strcpy(ACKPkt.data, "ACK");
            ACKPkt.indicator = ? //wrong indicator is sent
            ? //Use write().
        }
    else
        {
            printf("ACK sent (SUCCESS)\n\n");
            // send [ACK, indicator number] to client.
            strcpy(ACKPkt.data, "ACK");
            ACKPkt.indicator = ?
            ? //Use write().
        }
}
```

# Simulation

---

## ■ 컴파일 및 실행

- 터미널 2개 실행 후 각각 공유 폴더 내의 Project1 파일들 저장 위치로 이동



The image shows two terminal windows side-by-side. The left window shows a sequence of commands to navigate from the root of the media directory to the codes directory within project1. The right window shows the same path but then lists the files in the codes directory.

```
jongmin@jongmin-VirtualBox: /media/sf_share/project_stud...  
jongmin@jongmin-VirtualBox:~$ cd /media  
jongmin@jongmin-VirtualBox:/media$ cd sf_share  
jongmin@jongmin-VirtualBox:/media/sf_share$ cd project_student  
jongmin@jongmin-VirtualBox:/media/sf_share/project_student$ cd project1  
jongmin@jongmin-VirtualBox:/media/sf_share/project_student/project1$ cd codes  
jongmin@jongmin-VirtualBox:/media/sf_share/project_student/project1/codes$ ls  
client.c  cmain.c  myReceiver.c  mySender.c  server.c  smain.c  src.jpg  
jongmin@jongmin-VirtualBox:/media/sf_share/project_student/project1/codes$
```

```
jongmin@jongmin-VirtualBox: /media/sf_share/project_stud...  
jongmin@jongmin-VirtualBox:~$ cd /media/sf_share/project_student/project1/codes  
jongmin@jongmin-VirtualBox:/media/sf_share/project_student/project1/codes$ ls  
client.c  cmain.c  myReceiver.c  mySender.c  server.c  smain.c  src.jpg  
jongmin@jongmin-VirtualBox:/media/sf_share/project_student/project1/codes$
```

# Simulation

## ■ 컴파일 및 실행 (1)

### – 터미널 1

- `gcc -o c cmain.c client.c mySender.c`
  - c라는 이름의 실행 파일 생성
- `./c 127.0.0.1 99999`
  - ./실행파일명 Server의IP Server의Port번호

### – 터미널 2

- `gcc -o s smain.c server.c myReceiver.c`
  - s라는 이름의 실행 파일 생성
- `./s 99999`
  - ./실행파일명 Port번호

```
jongmin@jongmin-VirtualBox: /media/sf_share/project_stud...
jongmin@jongmin-VirtualBox:~$ cd /media
jongmin@jongmin-VirtualBox:/media$ cd sf_share
jongmin@jongmin-VirtualBox:/media/sf_share$ cd project_student
jongmin@jongmin-VirtualBox:/media/sf_share/project_student$ cd project1
jongmin@jongmin-VirtualBox:/media/sf_share/project_student/project1$ cd codes
jongmin@jongmin-VirtualBox:/media/sf_share/project_student/project1/codes$ ls
client.c cmain.c myReceiver.c mySender.c server.c smain.c src.jpg
jongmin@jongmin-VirtualBox:/media/sf_share/project_student/project1/codes$ gcc -o c cmain.c client.c mySender.c
jongmin@jongmin-VirtualBox:/media/sf_share/project_student/project1/codes$ ./c 127.0.0.1 99999
```

```
jongmin@jongmin-VirtualBox: /media/sf_share/project_stud...
jongmin@jongmin-VirtualBox:~$ cd /media/sf_share/project_student/project1/codes
jongmin@jongmin-VirtualBox:/media/sf_share/project_student/project1/codes$ ls
client.c cmain.c myReceiver.c mySender.c server.c smain.c src.jpg
jongmin@jongmin-VirtualBox:/media/sf_share/project_student/project1/codes$ gcc -o s smain.c server.c myReceiver.c
jongmin@jongmin-VirtualBox:/media/sf_share/project_student/project1/codes$ ./s 99999
```

## ■ 컴파일 및 실행 (2)

### – 터미널 1

- `gcc -o c cmain.c client.c mySender.c`
  - c라는 이름의 실행 파일 생성
- `./c 127.0.0.1 99999`
  - ./실행파일명 Server의IP Server의Port번호

### – 터미널 2

- `gcc -o s smain.c server.c myReceiver.c`
  - s라는 이름의 실행 파일 생성
- `./s 99999`
  - ./실행파일명 Port번호

```
jongmin@jongmin-VirtualBox: /media/sf_share/project_stud...  
Connect to (127.0.0.1:99999)? [Y/n]
```

```
jongmin@jongmin-VirtualBox: /media/sf_share/project_stud...  
Socket has been created.  
The server address information (ANYIP:99999) is bound.  
The listen queue has been created.  
Ready to receive TCP connection request.  
█
```

# Simulation

## ■ 실행 결과 예시 (1)

```
jongmin@jongmin-VirtualBox: /media/sf_share/project_stud...
4096 Bytes data (seq: 31, idc: 1) sent (LOSS)
TIMEOUT => resent 4096 Bytes data (seq: 31, idc: 1) sent (SUCCESS)
ACK received

4096 Bytes data (seq: 32, idc: 0) sent (SUCCESS)
ACK received

4096 Bytes data (seq: 33, idc: 1) sent (SUCCESS)
ACK received

4096 Bytes data (seq: 34, idc: 0) sent (SUCCESS)
ACK received

4096 Bytes data (seq: 35, idc: 1) sent (SUCCESS)
TIMEOUT => resent 4096 Bytes data (seq: 35, idc: 1) sent (SUCCESS)
TIMEOUT => resent 4096 Bytes data (seq: 35, idc: 1) sent (SUCCESS)
ACK received

4096 Bytes data (seq: 36, idc: 0) sent (SUCCESS)
ACK received

1791 Bytes data (seq: 37, idc: 1) sent (SUCCESS)
TIMEOUT => resent 1791 Bytes data (seq: 37, idc: 1) sent (SUCCESS)
jongmin@jongmin-VirtualBox: /media/sf_share/project_student/project1/codes$

jongmin@jongmin-VirtualBox: /media/sf_share/project_stud...
ACK sent (SUCCESS)

4096 Bytes data (seq: 35, idc: 1) received
Acceptable Data
ACK sent (ERROR)

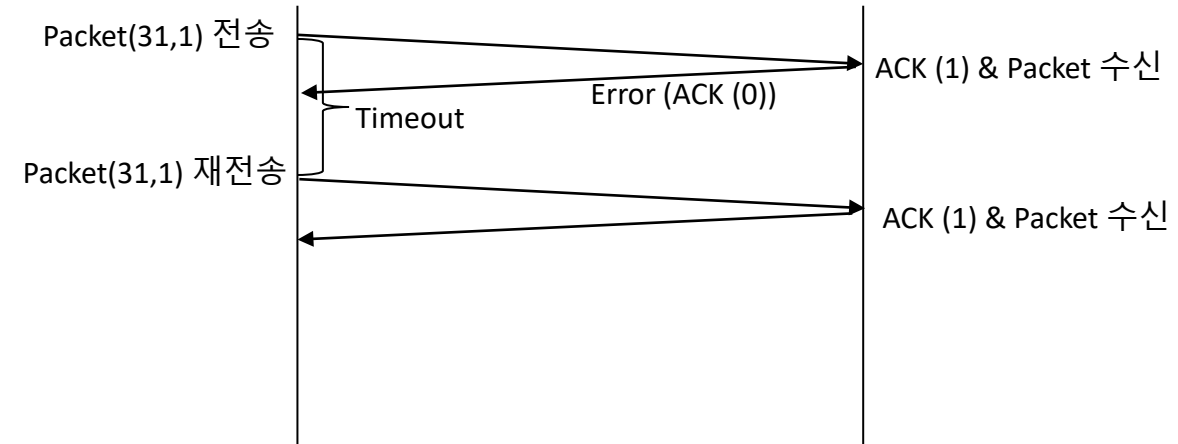
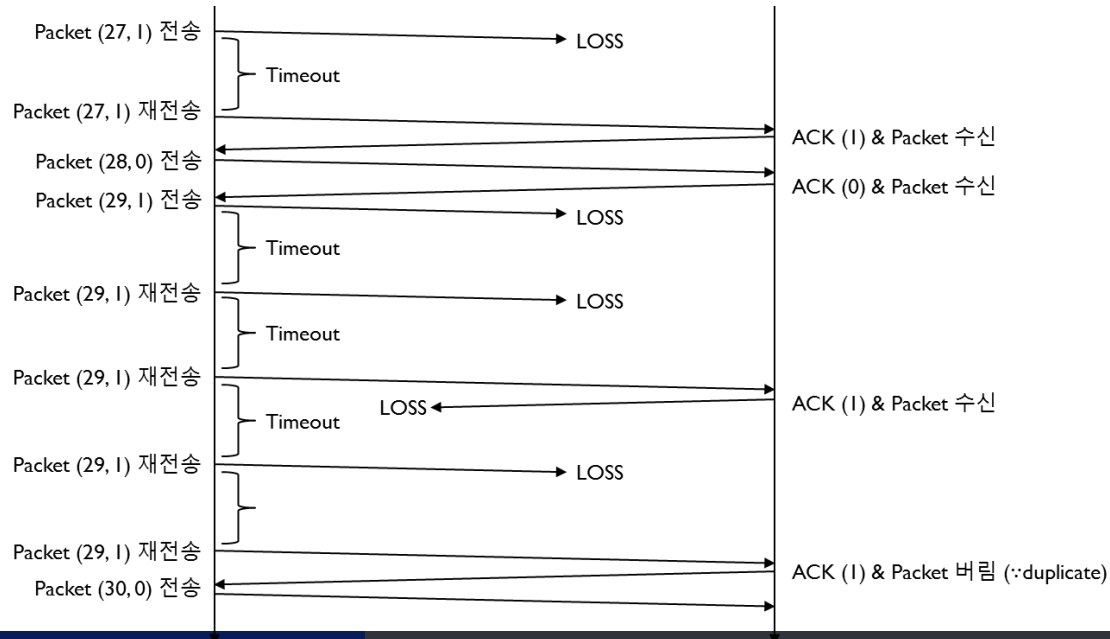
4096 Bytes data (seq: 35, idc: 1) received
Non-acceptable Data => Data will be thrown away.
ACK sent (LOSS)

4096 Bytes data (seq: 35, idc: 1) received
Non-acceptable Data => Data will be thrown away.
ACK sent (SUCCESS)

4096 Bytes data (seq: 36, idc: 0) received
Acceptable Data
ACK sent (SUCCESS)

1791 Bytes data (seq: 37, idc: 1) received
Acceptable Data
ACK sent (LOSS)

The file (153343 Bytes) has been received.
jongmin@jongmin-VirtualBox: /media/sf_share/project_student/project1/codes$
```





# Simulation

## ■ 실행 결과 예시 (2)

