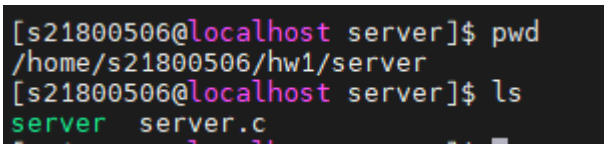
1. 팀원
    - 이동규(21800506)
    - 한승화(22000786)

2. 파일 위치
    a. server (ip 주소: 203.252.112.26)

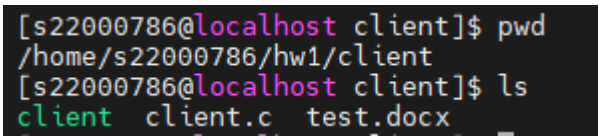


        - 소스파일

          /home/s21800506/hw1/server/server.c
        - 실행파일

          /home/s21800506/hw1/server/server

    b. client (ip 주소: 203.252.112.25)



        - 소스파일

          /home/s22000786/hw1/client/client.c
        - 실행파일

          /home/s22000786/hw1/client/client
        - 전송에 사용한 파일

          /home/s22000786/hw1/client/test.docx

3. 설계한 protocol 설명

a. 전송하는 packet 및 header 구조

seq(4 bytes)	msg_type(4 bytes)	payload(4 bytes)
data(512 bytes)		

우선 저희는 client 에서 server 로 전송하는 packet 의 구조를 위의 그림과 같이 정의하였습니다. 파란색으로 표시된 부분이 header, 노란색으로 표시된 부분이 data 입니다.

저희가 header 에 포함한 것은 seq, msg_type, payload 로 각 필드의 의미는 다음과 같습니다. (저희가 구현한 프로그램에서는 ack 이 양방향으로 이동하는 것이 아니라 server 에서 client 방향으로만 일방적으로 전송되는 정보이기 때문에, 이를 패킷의 헤더 구조에 포함할 필요 없이 integer type 의 데이터를 주고 받는 형식으로만 구현해도 된다고 판단하였고, ack 을 패킷의 헤더에 포함하지 않았습니다.)

〈각 헤더 필드의 의미〉

- seq: 전송하는 패킷의 sequence number (0 부터 시작)
- msg_type: 전송하는 패킷의 메시지 타입
  - 0 : 첫 번째 패킷, 즉 파일의 이름을 의미
  - 1 : (파일 제목, 마지막 파일 데이터를 제외한)일반적인 파일 데이터
  - 2: 마지막 파일 데이터. 즉, 마지막 패킷
- payload: data 에 들어있는 값 중 몇 byte 가 실제 데이터인지

〈헤더를 위와 같이 구성한 이유〉

- seq
  : server 가 받은 패킷이 몇번째 패킷인지를 확인하고, ack 를 몇번으로 해서 전송해야 할지 등을 결정하기 위해서 seq number 와 관련된 필드가 필요하다고 생각했습니다.

- msg_type

  : 주어진 문제 조건에서, 첫 번째로 전송하는 패킷은 파일 이름이어야 하고 서버는 해당 이름으로 파일을 만들 수 있어야 하며, 파일 전송이 완료되었다는 것을 server 에 알릴 수 있어야 했습니다. 파일 이름을 데이터로 포함하는 패킷, 일반적인 파일 데이터를 담고 있는 패킷, 마지막 파일 데이터를 담고 있는 패킷, 총 3 가지 종류로 패킷을 구분하여 각 종류별로 다른 메시지 타입 값을 가지게 함으로써 주어진 문제 조건에 만족하는 동작이 가능하도록 하였습니다.

- payload

  : 해당 필드는 전송한 데이터 중 실제 데이터가 얼마인지를 알려주기 위해 포함했습니다. 단순히 수신한 데이터 중에서 데이터 영역의 크기인 512 bytes 를 모두 읽어서 파일을 쓰도록 하는 경우에는 실제 전송한 데이터가 512 bytes 보다 적은 경우 의미 없는 데이터가 파일에 작성된다는 문제점이 있었습니다. 또, 수신한 데이터 중에서 실제 데이터의 크기를 strlen() 함수를 통해 측정하는 경우에는, binary data 를 ascii 로 변환하는 과정 등에서 데이터의 중간 부분에 null character 가 포함될 수 있어서 정확한 실제 데이터 크기의 측정이 불가하다는 문제점이 있었습니다. 따라서 모든 경우를 고려하여, 항상 수신한 패킷에서 실제 데이터에 해당하는 데이터만큼을 읽고 파일에 쓰기 위해서 payload 라는 필드를 헤더에 포함시켜 전달했습니다.

b. 사용한 timer 와 동작

i. 사용한 timer

- 저희는 high resolution timer 를 사용하였습니다. 또한, 전체적인 program 동작에서 각 packet 별로 timer 를 두는 것이 아니라 하나의 timer 를 사용하였습니다. timeout 값은 timer 를 생성할 때 해당 값이 인자로 전달될 수 있도록 하였고, timeout 이 발생할 경우, handler 에서 적절한 동작이 이루어질 수 있도록 하였습니다.

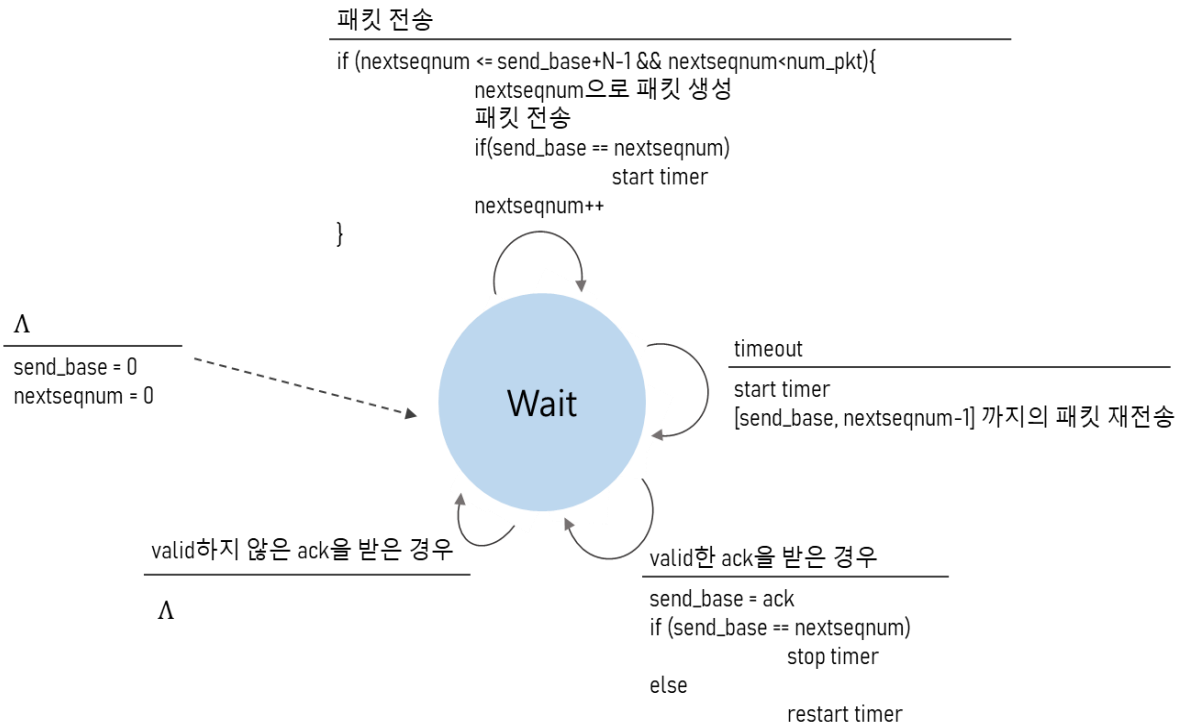
ii. 동작

- 저희가 사용한 timer 는 전송되었지만 아직 ack 을 받지 못한 packet 중에서 가장 전송된지 오래된 packet 에 대한 timer 로 생각할 수 있습니다.
- ACK 이 도착하였을 때, 전송되었지만 아직 ACK 을 받지 못한 packet 이 존재하면, timer 가 restart 되어 해당 packet 에 대한 timer 로서 동작할 수 있도록 하였고, 만약 전송되었지만 ACK 을 아직 받지 못한 packet 이 없다면 timer 를 stop 하도록 하였습니다.

4. 프로그램 동작 상태 및 결과

(저희는 전체적인 프로그램 구현에 cumulative ack 와 nextexpected seqnum 를 ack number 로 사용하는 방식을 채택하였습니다)

a. sender(client) 동작



(구현한 sender 의 동작 state diagram)

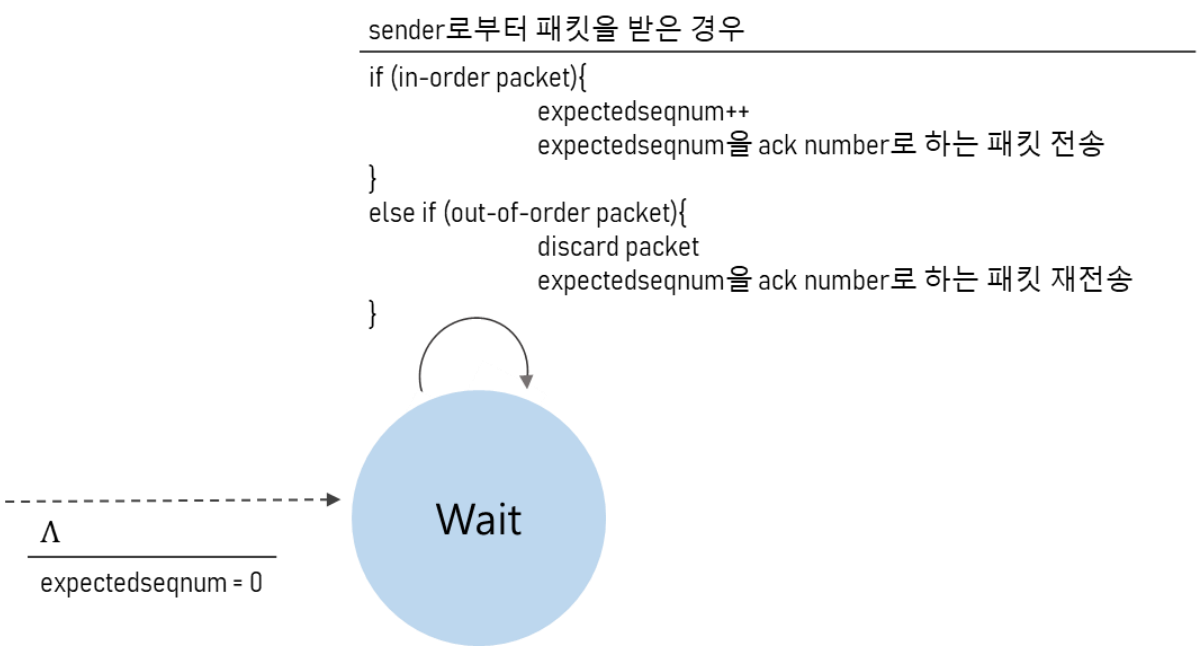
- 저희가 구현한 sender 의 동작은 위의 state diagram 으로 표현될 수 있습니다.

i) 보내야 하는 데이터가 남아있고 패킷이 전송 가능한 경우에는 nextseqnum 로 패킷을 생성하여 전송한 후(이때 만약, send_base 와 nextseqnum 가 같다면 새롭게 timer 를 시작) nextseqnum 를 1 증가

ii) valid 한 ack 을 받은 경우에는 cumulative ack 를 적용하여 send_base 를 ack number 로 변경하고 send_base 와 nextseqnum 가 같다면 timer 를 멈추고 아니라면 timer 를 재시작

iii) timeout 이 발생한 경우, timer 를 재시작하고 send_base 부터 next_seqnum-1 까지의 패킷을 재전송

b. receiver(server) 동작



(구현한 receiver 의 동작 state diagram)

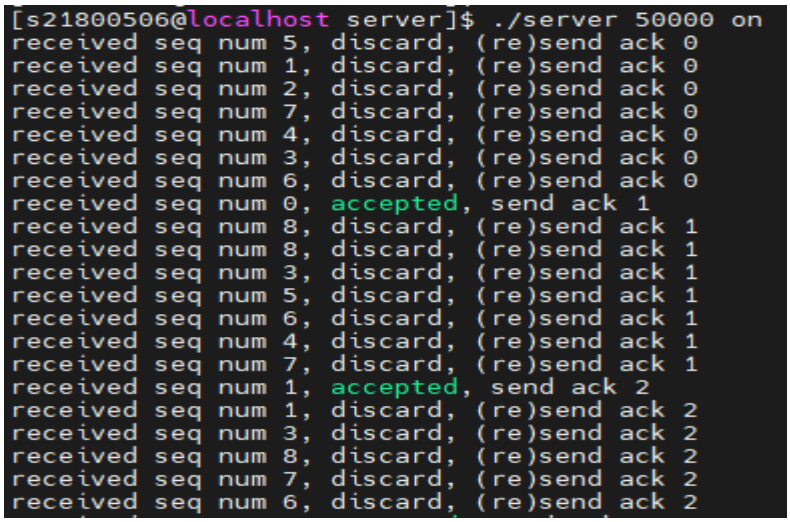
i) 패킷을 받은 경우

- 기대하고 있던 seq num 를 가진 packet 을 받았다면 expectedseqnum 를 증가시키고 expectedseqnum 를 ack number 로 하는 패킷 전송
    - 이때 받은 패킷의 msg_type 이 0 이라면 받은 데이터가 파일 이름이므로 해당 이름으로 파일 생성

- msg_type 이 1 혹은 2 라면 데이터이므로 파일에 작성 (2 인 경우에는 마지막 데이터이므로 작성 후 완료되었다는 메세지 출력)
- 기대하고 있던 seq num 가 아닌 packet(순서에 맞지 않는)을 받았다면, packet 을 discard 하고 expectedseqnum 을 ack number 로 하는 패킷을 재전송

c. 파일 전송 후 결과 분석

- 아래 사진은 server 실행 결과의 일부를 캡처한 것으로,
  - expectedseqnum 에 해당하는 packet 이 오지 않은 경우 discard 한 후 ack 를 다시 전송하고,
  - expectedseqnum 에 해당하는 packet 이 온 경우 해당 패킷을 accpet 하고 다음 expectedseqnum 를 ack number 로 하는 ack 을 전송하는 것을 볼 수 있습니다.



- 아래 사진은 client 실행 결과의 일부를 캡처한 것으로,
  - 데이터를 보낼 수 있는 경우에는 데이터를 보내고,
  - timeout 이 발생한 경우에는 send_base 부터 nextseqnum-1 까지의 패킷을 재전송하며,
  - duplicate ACK 이 도착하는 경우에는 이를 ignore 하고,
  - valid 한 ack 이 들어왔을 때만 변수의 값을 적절히 설정해 window 를 sliding 하면서 패킷이 전송되는 것을 확인할 수 있습니다.

```
[s22000786@localhost client]$ ./client 10.1.0.2 50000 8 on
total file size:  105174, file name: test.docx
-------------------------------------------------
start timer [seq 0]
send [seg 0] [msg type 0] [payload 9]
send [seg 1] [msg type 1] [payload 512]
send [seg 2] [msg type 1] [payload 512]
send [seg 3] [msg type 1] [payload 512]
send [seg 4] [msg type 1] [payload 512]
send [seg 5] [msg type 1] [payload 512]
send [seg 6] [msg type 1] [payload 512]
send [seg 7] [msg type 1] [payload 512]
[ack 1] received
start timer [seq 1]
send [seg 8] [msg type 1] [payload 512]
ignore duplicate ack 1
Time-out [seq 1], recovery & start timer [seq 1]
(re)send [seg 1] [msg type 1] [payload 512]
(re)send [seg 2] [msg type 1] [payload 512]
(re)send [seg 3] [msg type 1] [payload 512]
(re)send [seg 4] [msg type 1] [payload 512]
(re)send [seg 5] [msg type 1] [payload 512]
(re)send [seg 6] [msg type 1] [payload 512]
(re)send [seg 7] [msg type 1] [payload 512]
(re)send [seg 8] [msg type 1] [payload 512]
ignore duplicate ack 1
ignore duplicate ack 1
ignore duplicate ack 1
ignore duplicate ack 1
ignore duplicate ack 1
Time-out [seq 1], recovery & start timer [seq 1]
(re)send [seg 1] [msg type 1] [payload 512]
(re)send [seg 2] [msg type 1] [payload 512]
(re)send [seg 3] [msg type 1] [payload 512]
(re)send [seg 4] [msg type 1] [payload 512]
(re)send [seg 5] [msg type 1] [payload 512]
(re)send [seg 6] [msg type 1] [payload 512]
(re)send [seg 7] [msg type 1] [payload 512]
(re)send [seg 8] [msg type 1] [payload 512]
[ack 2] received
start timer [seq 2]
```

- 파일 전송이 완료된 후 wc command 를 통해 확인한 결과 글자 수, 라인 수, 바이트 수가 같음을 알 수 있습니다. 이를 통해 전송한 파일이 정상적으로 서버에 도착하였음을 확인할 수 있습니다. 또한, 전송한 파일의 이름과 같은 이름의 파일이 생성된 것을 확인할 수 있습니다.

```
[s21800506@localhost server]$ ls
server  server.c  test.docx
[s21800506@localhost server]$ wc test.docx
   369   2149 105174 test.docx
[s21800506@localhost server]$
```

(server)

```
[s22000786@localhost client]$ ls
client  client.c  test.docx
[s22000786@localhost client]$ wc test.docx
   369   2149 105174 test.docx
[s22000786@localhost client]$
```

(client)

- 또한, diff 명령어를 사용해, 수신하여 새로 저장한 파일과 원본 파일의 내용이 일치하는 것을 확인하였습니다. (origin.docx 파일은 원본 파일을 옮겨서 저장한 파일입니다.)

```
[s21800506@localhost server]$ diff ./test.docx ../origin.docx
[s21800506@localhost server]$
```