PA HW2

21400217 김태은, 21400286 박대솔

1. TCP의 경우

<파일 이름을 전송하기>

TCP의 경우 클라이언트가 보낸 메시지가 독립적으로 전달되는 것이 아니라 바이트스트림으로 전달된다. 따라서 이를 수신하는 서버에서는 파일의 이름을 전체 메시지에서 구분할 방법이 필요하다. 이를 위해 클라이언트에서 처음 파일의 이름을 메시지로 보낼 때 "이름길이 이름"과 같은 형식으로 파일 이름의 길이, 공백, 그리고 실제 파일 이름을 전송하게 하였다. 이에 서버에서는 처음 read를 할 때 공백을 기준으로 strtok()을 해서 첫번째 토큰, 즉 파일 이름의 길이를 알아낸다. 그 후 처음 read()했던 데이터 안에 파일 이름이 다 들어있는지 파악하고, 필요시 read를 추가적으로 진행해서 파일 이름을 마저 받아온다.

<데이터 받아오기>

먼저 받아온 파일 이름으로 파일을 열어야 한다. TCP에서는 상대가 close()했을 때 read()가 0을 반환한다. 따라서 서버는 read()의 결과가 0이 될 때까지 반복적으로 read()를 하며 읽어온 데이터를 fwrite()으로 파일에 써준다. 만약 데이터가 모두 전송되고 클라이언트가 close했다면 서버도 해당 소켓을 close하고 다시 accept를 하며 다음 클라이언트를 기다린다.

<실험>

```
[s21400217@localhost TCP]$ ./TCPclient 10.1.0.1 55553 oneMegaFile.dat
I sent the title as: 15 oneMegaFile.dat
All file sent successfully
[s21400217@localhost TCP]$ wc oneMegaFile.dat
      0       0 1048576 oneMegaFile.dat
```

(TCP 클라이언트 화면)

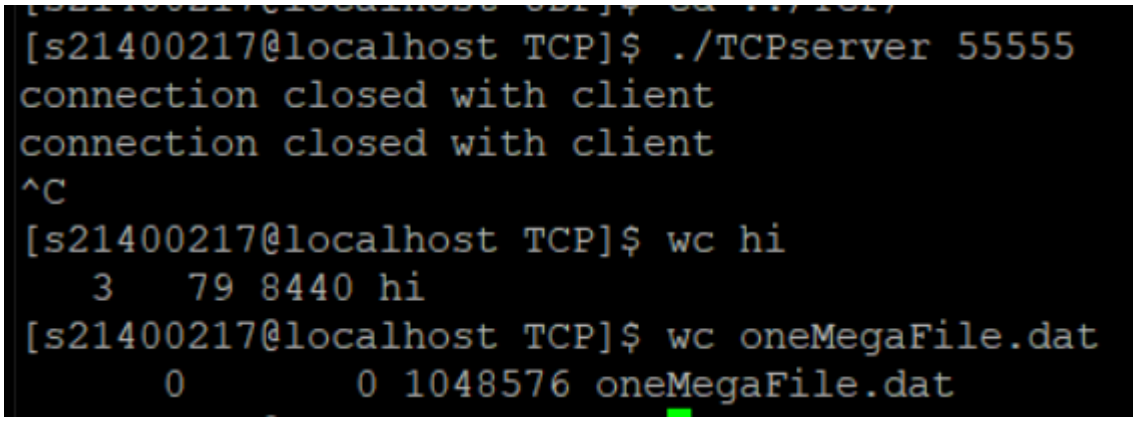
```
[s21400217@localhost TCP]$ ./TCPserver 55553
connection closed with client
^C
[s21400217@localhost TCP]$ wc oneMegaFile.dat
      0       0 1048576 oneMegaFile.dat
```

(TCP 서버 화면)

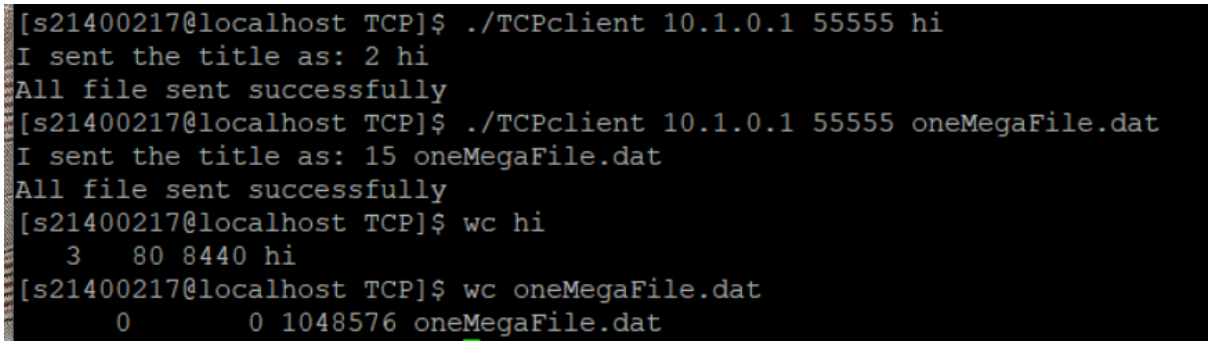
<실험 해석>

실행 후 wc 명령어로 각 파일의 길이를 확인해보니 동일했다. 이 뜻은 loss없이 전체 파일이 성공적으로 전송되었다는 것이다. 다만 loss를 복구를 처리하기 위해 delay가 상당히 많이 발생함을 알 수 있었다.

<Iterative Server 검증>



(두 클라이언트를 연속으로 처리한 서버 화면)

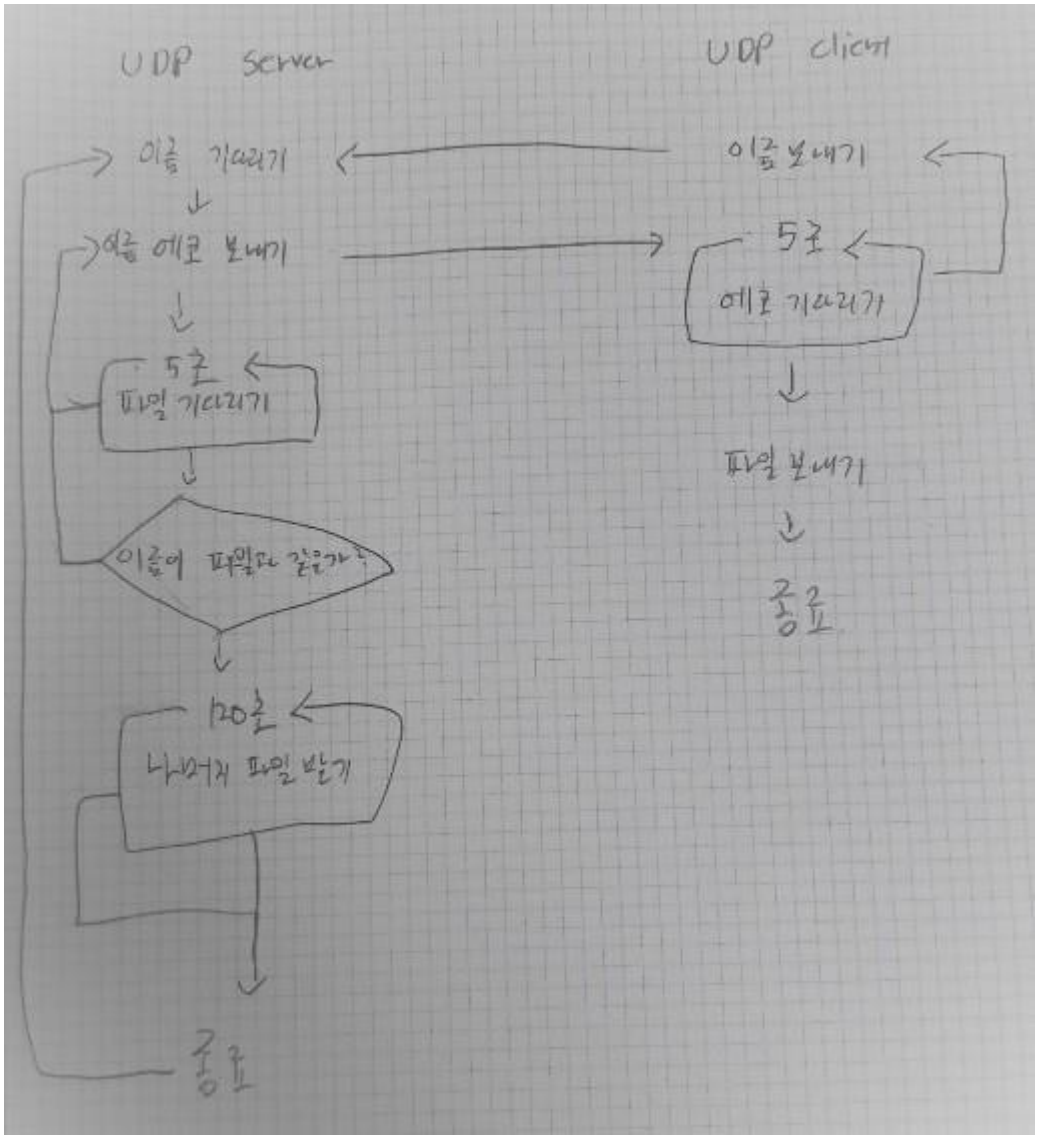


(서버에 두 번 파일을 전송한 클라이언트 화면)

2. UDP의 경우

<파일 이름을 확실히 전달하기>

UDP의 경우 파일 이름이 하나의 독립적인 데이터그램으로 전송되기 때문에 서버측 에서 이를 따로 자르거나 구분할 필요는 없다. 다만 중간에 loss가 되더라도 UDP 레벨에서 따로 복구를 해주지 않으므로 어플리케이션 프로그램 단위에서 따로 이를 복구하는 작업이 필요하다. 이를 위해 프로그램이 따라야 할 흐름을 다이어그램으로 나타내면 다음과 같다.



다이어그램을 설명하자면 다음과 같다.

우선, 서버의 흐름을 설명하겠다. 서버는 파일 이름을 기다리다가 받게 되면 그대로 에코를 클라이언트에게 보내준다. 그 후 파일의 데이터를 기다리게 된다. 이 때 setsockopt() 함수를 통해 소켓에 타이머를 5초로 걸고 데이터를 기다리게 된다. 만약 5초 이내에 데이터가 도착하지 않는다면 서버가 클라이언트에게 보낸 에코가 중간에 loss가 되었다 판단하여 에코를 다시 보낸다. 만약 5초 이내에 데이터가 도착해도 우선 처음에 받았던 파일 이름과 같은 데이터인지 확인해 본다. 왜냐하면 클라이언트에서도 파일 이름을 다시 보낼 가능성이 있기 때문이다. 만약 파일 이름이 다시 온다면 역시 에코를 다시 클라이언트에게 보내 준다.

성공적으로 파일의 데이터그램을 하나 받게 된다면 소켓의 타이머를 120초로 설정하고 나머지 데이터를 마저 받게 된다. 이 때 타이머를 120초로 설정한 것은 UDP의 경우 서버가 클라이언트의 상황을 알기 어렵기 때문에 일정 시간동안 연결이 없으면 클라이언트가 데이터를 모두 보냈다고 판단하기 위해서이다.

이 때 Iterative Server이기 때문에 서버는 다시 새로운 클라이언트로부터의 이름을 기다린다. 물론 이때는 소켓의 타이머를 해제해주어야 한다.

클라이언트의 흐름을 설명하겠다. 클라이언트는 우선 서버에게 파일 이름을 보낸다. 그 후 소켓의 타이머를 5초로 설정한 후 서버에게서 오는 에코를 기다린다. 5초 안에 에코가 오지 않는다면 파일 이름을 다시 보내준다. 만약 에코가 온다면 파일 이름이 전송된 것이 확인되었기 때문에 나머지 파일을 보내주고 종료하게 된다.

<실험>

전송될 파일은 대략 1 메가바이트의 oneMegaFile.dat으로 wc 결과는 1048576이다.

<UDP 클라이언트 결과>

```
[s21400217@localhost UDP]$ ./UDPclient 10.1.0.1 55554 oneMegaFile.dat
no ack received from the server. Resent title
Successfully sent the title as: oneMegaFile.dat
All work done
[s21400217@localhost UDP]$ wc oneMegaFile.dat
      0       0 1048576 oneMegaFile.dat
```

(서버로부터 에코가 한 번에 오지 않은 경우)

```
[s21400217@localhost UDP]$ ./UDPclient 10.1.0.1 55555 oneMegaFile.dat
Successfully sent the title as: oneMegaFile.dat
All work done
```

(처음부터 성공적으로 전송된 경우)

<UDP 서버 결과>

```
[s21400217@localhost UDP]$ ./UDPserver 55554
title recieved from client: oneMegaFile.dat
Timeout waiting on data. Resend ack
Title recieved again!! Resend ack
finish one client
^C
[s21400217@localhost UDP]$ wc oneMegaFile.dat
     0      0 906240 oneMegaFile.dat
```

(파일을 기다리다 Timeout 발생한 경우 & 파일 이름이 다시 온 경우)

```
[s21400217@localhost UDP]$ ./UDPserver 55554
title recieved from client: oneMegaFile.dat
Title recieved again!! Resend ack
finish one client
^C
[s21400217@localhost UDP]$ wc oneMegaFile.dat
     0      0 879616 oneMegaFile.dat
```

(파일 이름이 다시 온 경우)

```
[s21400217@localhost UDP]$ ./UDPserver 55555
title recieved from client: oneMegaFile.dat
Timeout waiting on data. Resend ack
finish one client
^C
[s21400217@localhost UDP]$ vi UDPserver.c
[s21400217@localhost UDP]$ wc oneMegaFile.dat
     0      0 893952 oneMegaFile.dat
```

(파일을 기다리다 Timeout이 온 경우)

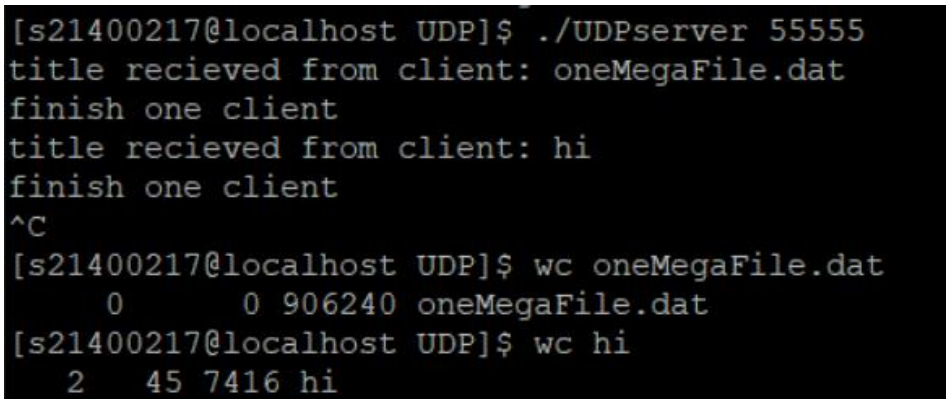
```
[s21400217@localhost UDP]$ ./UDPserver 55555
title recieved from client: oneMegaFile.dat
finish one client
^C
[s21400217@localhost UDP]$ wc oneMegaFile.dat
     0      0 880640 oneMegaFile.dat
```

(아무 문제 없이 전송이 이루어진 경우)

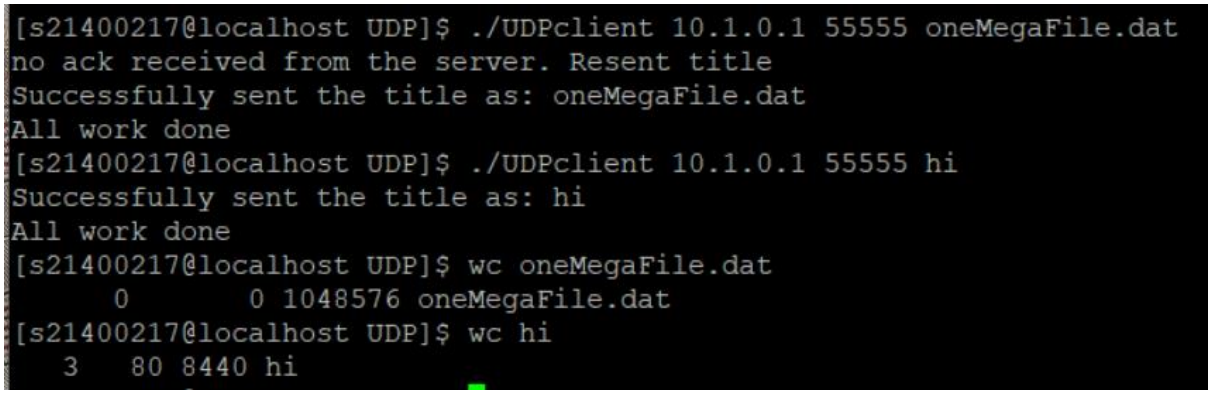
<실험 해석>

결과를 보면 중간에 문제가 발생하여도 이를 복구하여 성공적으로 전송이 이루어지는 것을 볼 수 있다. 그러나 UDP의 특성상 중간에 loss가 일어나기 때문에 따로 방지를 한 파일 이름을 제외한 데이터는 loss가 발생했다. 원본 파일과 실제 전송된 파일의 크기 차이의 평균을 구해 loss rate을 대략적으로 추측하면 15%정도 된다는 것을 알 수 있다.

<Iterative Server 검증>



(두 클라이언트를 처리한 서버 화면)



(서버에 두 번 파일을 전송한 클라이언트 화면)