Programming HW 1

21400217 김태은, 21400286 박대솔

1. becho_clinet.c 와 becho_server.c에서 송수신 방식과 동일하게 TCP program을 작성하여 실행하시오. TCP와 UDP 실행 결과에 어떤 차이가 있는 가를 비교하고, 그 이유를 설명하시오.

<UDP 실행 시 클라이언트 화면>

```
[s21400217@localhost Client]$ ./b_client 203.252.112.25 55556
0th message from the server : Good
1th message from the server : Evening
2th message from the server : Everybody!
```

<UDP 실행 시 서버 화면>

```
[s21400217@localhost Server]$ ./becho_server 55556
received number: 0
received message from client : Good
 received number: 1
received message from client : Evening
 received number: 2
received message from client : Everybody!
```

<TCP 실행 시 클라이언트 화면>

```
[s21400217@localhost Client]$ ./b_client_tcp 203.252.112.25 55558
0th message from the server : Good Evening Everybody!
```

<TCP 실행 시 서버 화면>

```
[s21400217@localhost Server]$ ./b_server_tcp 55558
received number: 0
recieved message : Good Evening Everybody!
```

실행 결과를 보면 UDP 통신을 사용할 때 클라이언트가 세번에 나눠서 보낸 메시지를 서버가 세번에 나눠서 읽게 된다. 하지만 TCP에서는 클라이언트가 세번에 나눠서 보낸 메시지를 서버는 한번에 읽게 된다.

그 이유는 다음과 같다. UDP는 프로세스에서 보낸 데이터 단위대로 데이터그램이 구분되어 전송되기에 한쪽에서 세 번에 나눠서 보내면 세 개로 구분되는 데이터가 전송된다. 반면에 TCP는 데이터 간의 경계가 없이 하나의 스트림으로 데이터가 전송되기 때문에 프로세스에서 데이터를 세번에 나누어 보내도 한 스트림으로 흘러가서 받는 쪽에서는 한 데이터처럼 받게 된다. 특히 현재 코드에서는 서버가 sleep을 하는 동안 세번의 데이터가 모두 도착해 서버가 한 번에 다 읽을 수 있게 된다.

2. becho_server.c 에서 BUFSIZE를 5로 (#define BUFSIZE 5)로 변경한 후, server와 client를 실행하시오. 수정하지 않았을 경우와 수정하였을 때에 어떤 차이가 있는 가를 비교하고, 그 이유를 설명하시오.

<UDP 실행 시 클라이언트 화면>

```
[s21400217@localhost Client]$ ./b_client 203.252.112.25 55555
0th message from the server : Good
1th message from the server : Eveni
2th message from the server : Every
```

<UDP 실행 시 서버 화면>

```
[s21400217@localhost Server]$ ./b_server 55555
received number: 0
received message from client : Good
 received number: 1
received message from client : Eveni
 received number: 2
received message from client : Every
```

실행 결과를 보면 각 메시지가 5 글자를 넘어갈 때 나머지 데이터가 누락되는 현상이 나타났다. 수정하기 전에는 클라이언트가 한 번에 보내온 메시지 전체를 읽을 정도로 BUFSIZE가 컸지만 이제는 작아진 BUFSIZE로 인한 제한이 생겨 한 번에 5글자만을 읽을 수 있게 되었다.

그 이유는 다음과 같다. UDP의 경우 프로세스에서 보낸 각 데이터가 BUFSIZE로 인한 글자 수를 초과하는 경우 나머지가 누락되게 된다. UDP는 프로세스 상에서의 데이터 경계를 고려하기에 이 5글자 제한은 각 메시지에 따로따로 적용되게 된다. 따라서 각 메시지의 앞 5글자씩만 받게 되는 것이다.

3. becho_clinet.c 와 becho_server.c에서 송수신 방식과 동일하게 작성한 TCP program에서 server의 BUFSIZE를 5로 (#define BUFSIZE 5)로 변경한 후, server와 client를 실행하시오. 2번에서의 UDP의 경우와 어떤 차이가 있는 가를 비교하고, 그 이유를 설명하시오.

<TCP 실행 시 클라이언트 화면>

```
[s21400217@localhost Client]$ ./b_client_tcp 203.252.112.25 55559
0th message from the server : Good
1th message from the server : Eveni
2th message from the server : ng Ev
```

<TCP 실행 시 서버 화면>

```
[s21400217@localhost Server]$ ./b_server_tcp 55559
received number: 0
recieved message : Good
received number: 1
recieved message : Eveni
received number: 2
recieved message : ng Ev
received number: 3
recieved message : erybo
```

현재 코드에서 클라이언트 쪽은 for문으로 read()를 세번 하게 해 두어서 세 번의 에코만 왔지만, 서버 쪽에서는 while문으로 클라이언트가 close할 때까지 계속 read를 하게 해 두었다. 그 결과 비록 BUFSIZE는 5로 설정되어 한 번에 받는 데이터가 5글자로 제한되었지만, 클라이언트가 close하기 전까지는 서버가 데이터를 계속 읽을 수 있음을 알 수 있다. 지금 같은 경우에는 서버가 4번째(3번) 데이터까지 읽었을 때 클라이언트가 close하기 때문에 통신이 종료되는 상황이다. 만약 클라이언트가 sleep 등으로 close를 연기한다면 서버는 클라이언트가 보낸 모든 데이터를 읽어올 수 있을 것이다.

그 이유는 다음과 같다. TCP는 프로세스에서의 데이터 경계를 고려하지 않는 bytestream 통신을 사용하기 때문에 BUFSIZE로 인한 데이터 길이 제한이 각 메시지에 적용되지 않았고 도달한 데이터 전체에 적용된다. 즉, 각 메시지를 독립적인 데이터로 보지 않고 경계가 없는 하나의 데이터로 처리한다는 뜻이다. 따라서 한 번에 읽을/받을 수 있는 데이터는 5글자이지만 데이터의 나머지 부분이 버려지지 않고 stream을 통해 연결되어 있기 때문에 다음 5글자 만큼의 데이터를 받을 때, 마저 받아올 수 있는 것이다. 반면 UDP는 한번의 전송이 하나의 데이터그램을 보내고 각 데이터그램은 독립적으로 처리되기 때문에 전송이 이루어졌을 때 BUFSIZE가 제한되어 못 받은 데이터는 결국 누락되는 것이다.

4. 1번에서 구성한 TCP program에서 server 코드에서 sleep() 코드를 모두 제거하고 program을 실행하시오. 실행 결과를 sleep()이 있는 1번 TCP의 경우와 어떤 차이가 있는 가를 비교하고, 그 이유를 설명하시오.

<TCP 실행 시 클라이언트 화면>

```
[s21400217@localhost Client]$ ./b_client_tcp 203.252.112.25 55559
0th message from the server : Good
1th message from the server : Evening Everybody!
```

<TCP 실행 시 서버 화면>

```
[s21400217@localhost Server]$ ./b_server_tcp 55559
received number: 0
recieved message : Good
received number: 1
recieved message : Evening Everybody!
```

실행 결과를 보면 1번의 실행결과와는 다르게 server에서 총 두번의 read가 실행됨을 볼 수 있다. 그 중 첫번째 read로 읽어온 데이터는 client가 보낸 첫번째 데이터이다. 그러나 두번째 read로 읽어온 데이터는 client가 보낸 두번째와 세번째 데이터이다.

1번에서의 코드가 실행될 때는 server가 sleep을 했기 때문에 sleep하는 동안 client가 세번의 데이터를 모두 보낼 수 있었다. 유의할 점은 accept가 되기 전에 sleep을 해도 client에서는 server에게 데이터를 전송할 수 있다는 점이다. 그래서 server가 sleep하는동안 세 번의 데이터 전송이 모두 이루어져 server가 read를 할 때 그 데이터를 한꺼번에 읽어 올 수 있다. 물론 BUFSIZE가 허락하는 만큼만 읽어올 수 있다.

반면 sleep을 없앤 코드의 경우 server는 client의 데이터가 도착하는 즉시 읽게 된다. 이 때 첫번째 데이터는 바로 읽을 수 있지만 이를 read하는 동안 두번째, 세번째 데이터가 도착하게 된다. 그렇기 때문에 server가 두번째로 read를 할 때는 이미 누적 두 번의 데이터가 도착한 상황이다. 그렇기 때문에 두번째 read의 결과는 client에서 보낸 두번째와 세번째 데이터가 된다.

5. UDP에서 recvfrom() 함수에서 return 값이 0 이 될 수 있는가? 있다고 한다면 어떤 경우인가? 실험을 한 다음에 방법을 설명하시오. 위의 방법을 TCP에 적용시키면 어떤 결과가 나타나는가? UDP 경우와 어떤 차이가 있는 가를 비 교하고, 그 이유를 설명하시오.

UDP의 경우:

<UDP Client 코드>

```
char MSG_M[] = "";
sendto(sock, MSG_M, 0, 0, (struct sockaddr*)&serv_addr, sizeof(serv_addr)) ;
```

<UDP Server 코드>

```
str_len = recvfrom(serv_sock, message, BUFSIZE, 0, (struct sockaddr*)&clnt_addr, &clnt_addr_size);
printf("recvfrom returned %d\n", str_len);
```

<UDP 실행 시 서버 화면>

```
[s21400217@localhost Server]$ ./b_server 55558
recvfrom returned 0
```

UDP 에서는 0이 반환될 수 있다. 해당 경우는 상대로부터 길이가 0인 데이터를 받았을 때이다. UDP에서 recvfrom() 함수가 0을 반환하도록 하기 위해 다음과 같이 실험하였다. 우선 클라이언트가 길이가 0인 string을 서버에게 보내게 하였다. 그후 서버에서 해당 데이터를 recvfrom()으로 받고 그 반환값을 출력했다. 출력 결과는 0이었다. 이로 인해 보내는 데이터의 길이가 0일 때는 recvfrom()이 0을 반환한다는 것을 알 수 있다.

TCP의 경우:

<TCP Client 코드>

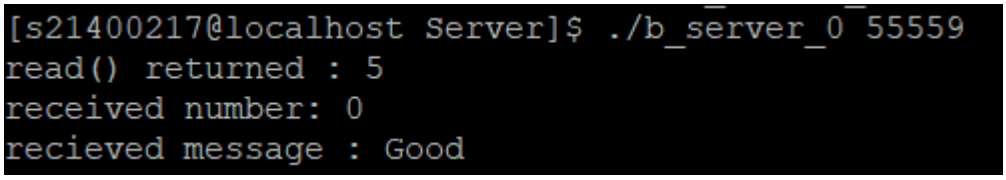
```
char MSG_M[] = "";
write(sock, MSG_M, 0);

write(sock, MSG1, strlen(MSG1) );
```

<TCP Server 코드>

```
str_len = read( clnt_sock, message, BUFSIZE) ;
printf("read() returned : %d\n", str_len);
message[str_len] = 0;
printf("received number: %d \nrecieved message : %s\n", num++, message);
```

<TCP 실행 시 서버 화면>



반면에 TCP에서는 길이가 0인 데이터가 전달이 안된다. 따라서 read의 return이 0이 되는 경우는 오직 상대방이 close했을 때이다.

TCP에서 write으로 길이가 0인 데이터를 보내는 실험을 해보았다. 우선 client에서 길이가 0인 데이터를 먼저 write하고 그 후 일반 데이터를 다시 write하였다. 이 때, server에서는 한번의 read만 실행하게 하였다. 그 후 server의 read 결과를 보면 client가 보낸 두번째 데이터만 읽었음을 알 수 있다. Client가 첫번째로 길이가 0인 데이터를 보내고 그 다음 두번째로 정상적인 데이터를 보냈는데 server는 client가 보낸 첫번째 데이터는 무시하고 두번째 데이터를 read한 것이다. 만약 길이가 0인 데이터만을 보낸다면 서버의 read에서 block이 발생하게 된다. 왜냐하면 아무것도 오지 않은 것으로 인식되기 때문이다. 따라서 TCP에서는 길이가 0인 데이터는 전달이 되지 않는 것을 확인할 수 있었다.