

컴퓨터 네트워크 실습 과제

2017029916

양동해

1. 주요 코드에 대한 설명

이 소켓 프로그램은 클라이언트가 서버에게 메시지를 보내면, 서버는 클라이언트에게서 받은 메시지를 echo로 답장해주는 프로그램이다. 주어진 과제 요구사항에 나와있듯 멀티 프로세스를 이용하여 구현했기 때문에, 여러 클라이언트가 한 서버에 접속할 수 있다. 프로그램은 C를 사용해 구현했으며, 서버는 local host를 사용한다. Port 번호는 define에서 8888로 정의했다.

- socket_server.c

```
16 // 서버 소켓 정의
17 int server_sockfd;
18 struct sockaddr_in server_addr;
19
20 // 서버 소켓 생성 후 addr 값 세팅
21 server_sockfd = socket(AF_INET, SOCK_STREAM, 0);
22 server_addr.sin_family = AF_INET;
23 server_addr.sin_addr.s_addr = htonl(INADDR_ANY);
24 server_addr.sin_port = htons(PORTNUM);
```

서버 소켓을 server_sockfd로 정의하고 server_addr을 만들어 주었다. 그 뒤에 서버 소켓을 socket()함수로 연결해주었는데, 인자로 들어간 값들은 TCP 프로토콜을 의미한다. 그리고 나서 server_addr에 소켓 주소를 세팅해주었다.

```
26 bind(server_sockfd, (struct sockaddr*)&server_addr, sizeof(server_addr)); // bind를 통해 소켓과 addr 묶어줌
27 listen(server_sockfd, 5); // listen으로 대기
```

서버 소켓을 생성한 뒤에는 bind()함수를 사용해 소켓과 addr를 묶어주었으며, listen() 함수를 통해 클라이언트를 기다리도록 설정했다.

```
29 printf("Accepting...\n");
30 while(1){
31     // 클라이언트 소켓 정의
32     int client_sockfd;
33     struct sockaddr_in client_addr;
34     int client_len = sizeof(client_addr);
35
36     // 클라이언트 소켓 accept
37     client_sockfd = accept(server_sockfd, (struct sockaddr*)&client_addr, (socklen_t *)&client_len);
38     printf("Accept success! And wait for another accept...\n");
39     clientIdx++;
```

listen() 후에는 "Accepting..."이라는 메시지와 함께 while문에 들어가면서, accept를 기다리는 상태로 돌입한다. while문 안에서는 클라이언트 소켓 변수를 만들어두고, accept() 함수를 통해 client를 받아드린다. 클라이언트가 성공적으로 accept 되면 메시지와 함께 clientIdx를 1 증가 시킨다.

```
41 // 멀티 프로세스 (자식 프로세스 생성)
42 int pid = fork();
43 if(pid == 0){
44     // 서버에서 클라이언트가 접속했다는 정보를 것을 출력
45     printf("[Client %d] is online.\n", clientIdx);
46
47     // 버퍼 생성 및 초기화
48     char buf[MAXLINE];
49     memset(buf, 0x00, MAXLINE);
50 }
```

클라이언트가 접속했기 때문에 멀티 프로세싱(fork()함수 사용)을 사용해 자식 프로세스를 생성한다. pid가 0인 것은 자식 프로세스이므로 아래 if문을 수행하지만, 부모 프로세스는 다시 while 문 처음으로 돌아가 새로 accept를 기다린다. 자식 프로세스로 만든 서버에선 클라이언트와의 연결이 이루어졌기 때문에, 서버에 클라이언트가 접속했다는 정보를 출력해준다.

```

51         while(1){
52             read(client_sockfd, buf, sizeof(buf)); // 클라이언트로부터 입력 받아옴
53             write(client_sockfd, buf, sizeof(buf)); // 클라이언트에게 echo
54             if(strcmp(buf, "exit") == 0){ // 받은 문자열이 exit이면 클라이언트 연결종료
55                 // 서버에서 정상적으로 종료가 되었다는 것을 출력
56                 printf("[Client %d] is closed\n", clientIdx);
57                 close(client_sockfd);
58                 goto end;
59             }
60             printf("[Client %d]: %s\n", clientIdx, buf); // 받아온 문자열 출력
61         }
62     }
63 }
64 end:
65     close(server_sockfd);
66
67     return 0;

```

그 뒤 클라이언트에서 보내는 메시지를 받고 이를 화면에 출력해주며, 만약 클라이언트에서 "exit" 메시지를 보냈다면 클라이언트와의 연결을 종료시킨다. 서버에서 정상적으로 종료가 되었다는 메시지를 출력하고 클라이언트 소켓을 닫은 뒤, goto 문을 통해 자식 프로세스로 만든 서버의 소켓도 닫는다.

- socket_client.c

```

14 // 클라이언트 소켓 정의
15 int client_sockfd;
16 struct sockaddr_in client_addr;
17
18 // 클라이언트 소켓 생성 후 addr 값 세팅
19 client_sockfd = socket(AF_INET, SOCK_STREAM, 0);
20 client_addr.sin_family = AF_INET;
21 client_addr.sin_addr.s_addr = inet_addr("127.0.0.1"); // local host로 접속
22 client_addr.sin_port = htons(PORTNUM);

```

클라이언트 소켓을 client_sockfd로 정의하고 client_addr을 만들어 주었다. 그 뒤에 클라이언트 소켓을 socket()함수로 연결해주었는데, 인자로 들어간 값들은 TCP 프로토콜을 의미한다. 그리고 나서 client_addr에 소켓 주소를 세팅해주었다.

```

24 connect(client_sockfd, (struct sockaddr*)&client_addr, sizeof(client_addr)); // 서버와 연결

```

클라이언트 소켓을 생성한 뒤에는 connect()함수를 사용해 서버에 접속한다.

```

26 while(1){
27     // 버퍼 생성 및 초기화
28     char buf[MAXLINE];
29     memset(buf, 0x00, sizeof(buf));
30
31     // 입력
32     printf(">>> ");
33     fgets(buf, sizeof(buf), stdin);
34     buf[strlen(buf) - 1] = '\0';
35
36     write(client_sockfd, buf, sizeof(buf)); // 서버에게 입력 전달
37     read(client_sockfd, buf, sizeof(buf)); // 서버로부터 echo 받아옴
38     if(strcmp(buf, "exit") == 0){ // 입력한 문자열이 exit이면 클라이언트 연결종료
39         // 클라이언트에서 정상적으로 종료가 되었다는 것을 출력
40         printf("Successfully terminated!\n");
41         close(client_sockfd);
42         break;
43     }
44     printf("Server(echo): %s \n", buf); // 받아온 문자열 출력
45 }

```

서버에 접속을 성공한 뒤에는, 서버에게 메시지를 보낼 수 있으며, "exit"을 입력하면 클라이언트가 종료된다. 접속이 종료되면, 클라이언트에서 정상적으로 종료가 되었다는 메시지를 출력하고 클라이언트 소켓을 닫아준다.

2. 실행 화면

본 프로그램은 mac os(Big sur 11.0.1 / clang(gcc) 버전: 12.0.0) 환경과 linux(Ubuntu 18.04 / gcc 버전: 7.5.0) 환경에서 진행했다.

- 컴파일 및 프로그램 실행

```
eastsea@EastSeai-iMac socket_programming % ls
Makefile      socket_client.c socket_server.c
eastsea@EastSeai-iMac socket_programming % make
gcc -o server socket_server.c
gcc -o client socket_client.c
eastsea@EastSeai-iMac socket_programming % ls
Makefile      client      server      socket_client.c socket_server.c
```

우선 압축파일을 풀게 되면, 보고서 파일을 제외한 3개의 파일이 존재한다. 여기서 make 명령어를 사용해 실행파일을 만들어준다.

```
eastsea@EastSeai-iMac socket_programming % ./server
Accepting...
```

[server 화면]

```
eastsea@EastSeai-iMac socket_programming % ./client
>> █
```

[client 화면]

이제 각각의 터미널에서 서버와 클라이언트 프로그램을 실행하면 되는데, 서버는 "./server"로 실행하고, 클라이언트는 "./client"로 실행한다.

- 프로그램 수행 스크린샷

```
eastsea@EastSeai-iMac socket_programming % ./server
Accepting...
Accept success! And wait for another accept..
[Client 1] is online.
```

[server 화면]

1개의 클라이언트가 접속하게 되면, 서버에선 Accept 됐다는 메시지와, 연결된 클라이언트의 정보를 출력해준다.

```
eastsea@EastSeai-iMac socket_programming % ./client
>> Hi, server!
Sever(echo): Hi, server!
>> █
```

[client1 화면]

```
eastsea@EastSeai-iMac socket_programming % ./server
Accepting...
Accept success! And wait for another accept..
[Client 1] is online.
[Client 1]: Hi, server!
```

[server 화면]

클라이언트에서 메시지를 입력하면, 서버에서 echo 메시지로 응답한다. 서버 측에서도 클라이언트의 메시지를 받으면 화면에 출력해준다.

```
eastsea@EastSeai-iMac socket_programming % ./client
>> █
```

[client2 화면]

```
eastsea@EastSeai-iMac socket_programming % ./server
Accepting...
Accept success! And wait for another accept..
[Client 1] is online.
[Client 1]: Hi, server!
Accept success! And wait for another accept..
[Client 2] is online.
```

[server 화면]

다른 클라이언트가 새로 접속하면, 서버에서도 새로 연결된 클라이언트의 정보를 출력해준다.

```
eastsea@EastSeau-iMac socket_programming % ./client
>> Hi, server!
Sever(echo): Hi, server!
>> I'm client 1.
Sever(echo): I'm client 1.
>> I like banana~
Sever(echo): I like banana~
>> WANNABE
Sever(echo): WANNABE
>> █
```

[client1 화면]

```
eastsea@EastSeau-iMac socket_programming % ./client
>> Hi, I'm client 2.
Sever(echo): Hi, I'm client 2.
>> My favorite music is
Sever(echo): My favorite music is
>> █
```

[client2 화면]

```
eastsea@EastSeau-iMac socket_programming % ./server
Accepting...
Accept success! And wait for another accept..
[Client 1] is online.
[Client 1]: Hi, server!
Accept success! And wait for another accept..
[Client 2] is online.
[Client 1]: I'm client 1.
[Client 1]: I like banana~
[Client 2]: Hi, I'm client 2.
[Client 2]: My favorite music is
[Client 1]: WANNABE
█
```

[server 화면]

위의 사진처럼 클라이언트에서 메시지를 보내면 서버에서 출력해주고, 클라이언트 측은 서버에서 echo 메시지를 받게 된다.

```
eastsea@EastSeau-iMac socket_programming % ./client
>> I'm client 3!!!!!!
Sever(echo): I'm client 3!!!!!!
>> █
```

[client3 화면]

```
eastsea@EastSeau-iMac socket_programming % ./server
Accepting...
Accept success! And wait for another accept..
[Client 1] is online.
[Client 1]: Hi, server!
Accept success! And wait for another accept..
[Client 2] is online.
[Client 1]: I'm client 1.
[Client 1]: I like banana~
[Client 2]: Hi, I'm client 2.
[Client 2]: My favorite music is
[Client 1]: WANNABE
Accept success! And wait for another accept..
[Client 3] is online.
[Client 3]: I'm client 3!!!!!!
█
```

[server 화면]

세 개의 클라이언트가 서버에 접속한 사진이다.

```
>> Hi, I'm client 2.
Sever(echo): Hi, I'm client 2.
>> My favorite music is
Sever(echo): My favorite music is
>> exit
Successfully terminated!
eastsea@EastSeau-iMac socket_programming % █
```

[client2 화면]

위 사진은 두번째 클라이언트가 서버와의 연결을 종료한 사진이다. 서버와 연결을 종료하면 다음과 같이 클라이언트에서 메시지가 출력되고 프로그램이 종료된다.

```
[Client 2]: My favorite music is
[Client 1]: WANNABE
Accept success! And wait for another accept..
[Client 3] is online.
[Client 3]: I'm client 3!!!!!!
[Client 2] is closed
```

[server 화면]

서버에서도 클라이언트 2의 종료를 알 수 있는 메시지가 출력된다.

```
[Client 2]: My favorite music is
[Client 1]: WANNABE
Accept success! And wait for another accept..
[Client 3] is online.
[Client 3]: I'm client 3!!!!!!
[Client 2] is closed
[Client 1]: But I'm still alive~~
[Client 3]: Me too
```

[server 화면]

두 번째 클라이언트의 접속이 종료되더라도, 클라이언트 1과 3은 접속을 유지하고 있으므로 메시지를 보낼 수 있다.

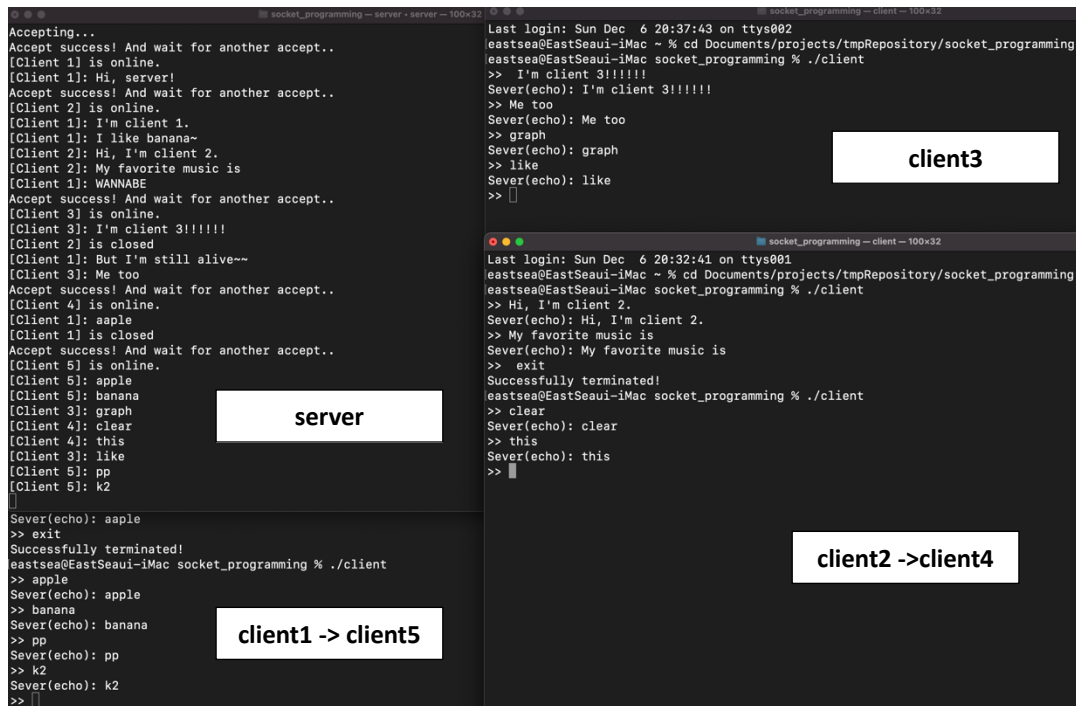
```
Successfully terminated!
eastsea@EastSeau-iMac socket_programming % ./client
>> 
```

[client4 화면]

```
[Client 2] is closed
[Client 1]: But I'm still alive~~
[Client 3]: Me too
Accept success! And wait for another accept..
[Client 4] is online.
```

[server 화면]

종료된 터미널에서도 새로운 클라이언트로 다시 접속하는 것이 가능하다.



```
Accepting...
Accept success! And wait for another accept..
[Client 1] is online.
[Client 1]: Hi, server!
Accept success! And wait for another accept..
[Client 2] is online.
[Client 1]: I'm client 1.
[Client 1]: I like banana~
[Client 2]: Hi, I'm client 2.
[Client 2]: My favorite music is
[Client 1]: WANNABE
Accept success! And wait for another accept..
[Client 3] is online.
[Client 3]: I'm client 3!!!!!!
[Client 2] is closed
[Client 1]: But I'm still alive~~
[Client 3]: Me too
Accept success! And wait for another accept..
[Client 4] is online.
[Client 1]: aapple
[Client 1] is closed
Accept success! And wait for another accept..
[Client 5] is online.
[Client 5]: aapple
[Client 5]: banana
[Client 3]: graph
[Client 4]: clear
[Client 4]: this
[Client 3]: like
[Client 5]: pp
[Client 5]: k2
[Client 5]: k2
Sever(echo): aapple
>> exit
Successfully terminated!
eastsea@EastSeau-iMac socket_programming % ./client
>> aapple
Sever(echo): aapple
>> banana
Sever(echo): banana
>> pp
Sever(echo): pp
>> k2
Sever(echo): k2
>> 
```

전체적인 실행 화면을 확인하면, 이런 식으로 클라이언트와 서버 간의 통신이 진행된다.