Computer Network – Socket Programming

컴퓨터소프트웨어학부 2019054957 이용우

1. Environment

- Language : C
- Operating System : Ubuntu 18.04

2. Procedure

- Server socket 생성
- Client connect 요청
- Client socket 생성
- Server accept
- Client의 data 전송 / Server측에서 data 수신
- "quit" 메시지 전송 시, client socket close

다음의 절차를 따라서 TCP 통신이 진행된다. 서버는 이 과정을 멀티 프로세스로 구현하여, 새로운 client가 connect할 때마다 새로운 프로세스를 만든다.

3. client.c

[1] Socket create

```
// socket create
int fd = socket(AF_INET, SOCK_STREAM, 0);
if(fd < 0)  {
    printf("Wrong socket fd\n");
    return 1;
}
```

과제는 INET과 tcp 프로토콜을 이용할 것이므로 socket 함수의 parameter를 다음과 같이 넘겨준다. Socket이 정상적으로 생성될 경우 fd는 양의 값을 가진다. 만

약 음수라면 socket이 생성되지 않은 것이므로 예외처리를 해준다.

[2] Connect to Server

```
// connect to server
struct sockaddr_in sockaddr;
sockaddr.sin_family = AF_INET;
sockaddr.sin_addr.s_addr = inet_addr("127.0.0.1");
sockaddr.sin_port = htons(8888);

if(connect(fd, (struct sockaddr*)&sockaddr, sizeof(sockaddr)) < 0) {
    printf("Connect Error\n");
    return 2;
}

printf("Connect Success\n");
```

통신을 위한 주소 정보를 sockaddr_in 구조체인 sockaddr에 넣어주고(본 프로그램은 로컬에서 테스트를 진행할 것이기 때문에, 주소를 127.0.0.1로 설정했다), connect 함수를 이용하여 서버와 연결한다. 만약 접속에 문제가 생기면 예외처리하여 프로그램을 종료한다.

[3] Send data

```
// send data
char buf[MAXLINE];
for(;;) {
    memset(buf, 0, MAXLINE);
    read(0, buf, MAXLINE - 1);
    write(fd, buf, strlen(buf));

    // if user types "quit", socket closes
    if(strstr(buf, "quit") != NULL) {
        break;
    }

    memset(buf, 0, MAXLINE);
    read(fd, buf, MAXLINE - 1);
}
```

전달할 내용을 buf에 저장하여 서버에 송신한다. 본 과제는 서버에 client가 접속한 정보와 종료된 정보를 출력만 하면 됨으로, client에서 타이핑한 메시지를 server에 띄우는 것만 구현했다. 메시지에 "quit"이 들어가면 통신을 종료하며

socket이 close된다.

[4] Connection closed

```
// Connection closed
printf("Connect closed\n");
close(fd);
return 0;
```

Client가 "quit"을 전송한 경우, loop를 빠져나와 socket을 close하고 프로그램을 종료한다.

4. server.c

[1] Socket create

```
// socket create
int sockfd = socket(AF_INET, SOCK_STREAM, 0);
if(sockfd < 0) {
    printf("Wrong socket fd\n");
    exit(0);
}
```

서버 socket을 생성한다. client.c의 [1]과 동일하게 작동한다.

[2] Bind

```
// bind
struct sockaddr_in addr;
addr.sin_family = AF_INET;
addr.sin_addr.s_addr = INADDR_ANY;
addr.sin_port = htons(8888);

if(bind(sockfd, (struct sockaddr*)&addr, sizeof(addr)) < 0) {
    printf("Bind Error\n");
    exit(0);
}
```

서버 socket을 주소에 묶어준다. sockaddr_in 구조체인 addr에 정보를 넣어주고, bind 함수를 이용하여 서버 socket을 커널에 등록한다. 이 과정을 통해 서버는 다른 시스템과 통신할 수 있는 상태가 된다. bind가 처리되지 않으면 예외처리하여 종료한다.

[3] Listen

```
    // listen
    if(listen(sockfd, 5) < 0) {
        printf("Listen Error\n");
        exit(0);
    }
```

Client의 접속 요청에 대한 수신 대기열을 만든다. listen 함수를 호출하면 client의 접속 요청이 올 때까지 대기한다. Listen에 에러가 생기면 예외처리한다.

[4] Create client socket & Accept

```
// create client socket & accept
    signal(SIGCHLD, SIG_IGN);
    for(;;) {
        clientLen = sizeof(clientAddr);
        clientSockfd = accept(sockfd, (struct sockaddr*)&clientAddr, &clientLe
n);

        if(clientSockfd < 0) {
            printf("Accept Error\n");
            exit(0);
        }
        printf("Client %d Connect\nIP Address: %s\n", clientSockfd, inet_ntoa(
clientAddr.sin_addr));
```

accept 함수를 통해 접속 요청이 허락되면, 커널에서 client socket을 생성한다. socket이 제대로 생성되지 않으면 예외처리한다. 제대로 생성된 경우에는 client가 서버에 접속한 것으로, 접속했다는 정보를 출력한다. 출력되는 정보에는 client socket의 fd와 IP주소가 있다. (fd로 접속한 clients를 구분한다.)

[5] Multiprocess

```
        // multiprocess
        int pid = fork();
```

접속된 client와 통신하는 child 프로세스를 생성한다. Parent 프로세스는 다시 loop의 처음으로 돌아가서 다음 client의 접속을 위한 작업을 수행한다.

[6] Receive data & Print

```
// for child process, receive data & print
if(pid == 0) {
    for(;;) {
        memset(buf, 0, MAXLINE);
        if(read(clientSockfd, buf, MAXLINE - 1) < 0) {
            close(clientSockfd);
            exit(0);
        }
        else {
            // if data contains "quit", close client socket
            if(strstr(buf, "quit") != NULL) {
                printf("client %d close\n", clientSockfd);
                close(clientSockfd);
                exit(0);
            }

            printf("%d> %s", clientSockfd, buf);
            write(clientSockfd, buf, strlen(buf));
        }
    }
}
```

Child 프로세스에서 client와의 통신을 진행한다. Client가 송신한 데이터를 그대로 출력하며, 송신한 client를 구분하기 위해 fd를 표시한다. 만약 Client의 데이터를 수신할 수 없으면 socket을 close하고 child process를 종료한다. 또한, client의 메시지 속에 "quit"이 포함되면 client가 통신을 끊은 것으로 인식하여 socket을 close하고 child process를 종료한다.

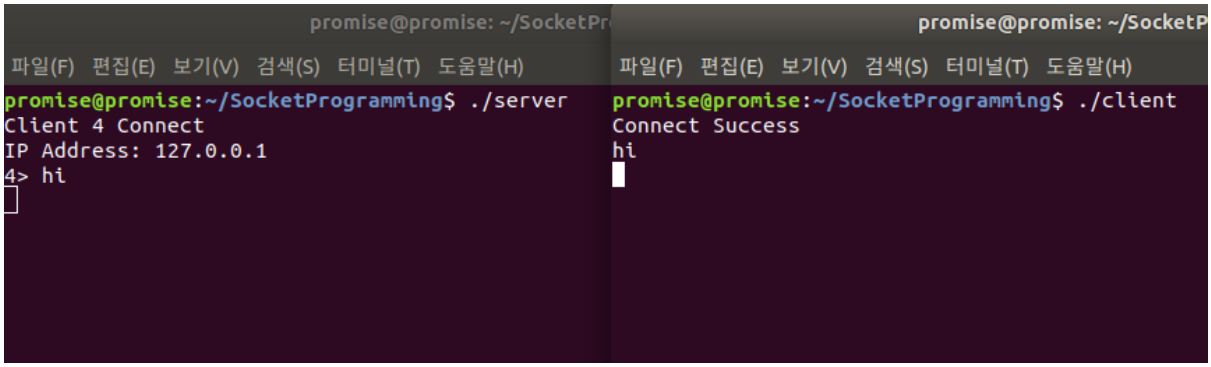
[7] Exception for fork error

```
// Exception for fork error
if(pid == -1) {
    printf("Fork Error\n");
    return 1;
}
```

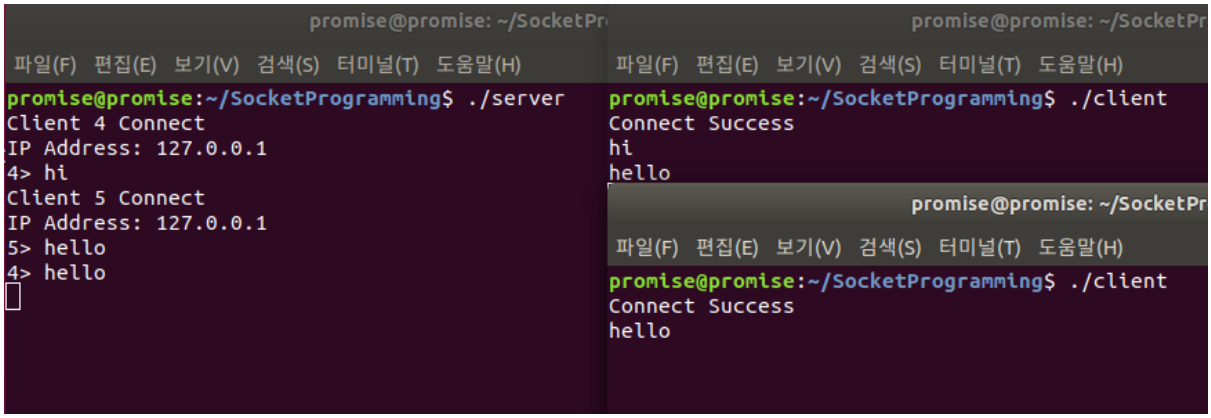
Child process를 생성할 수 없을 때 예외처리한다.

5. Execution

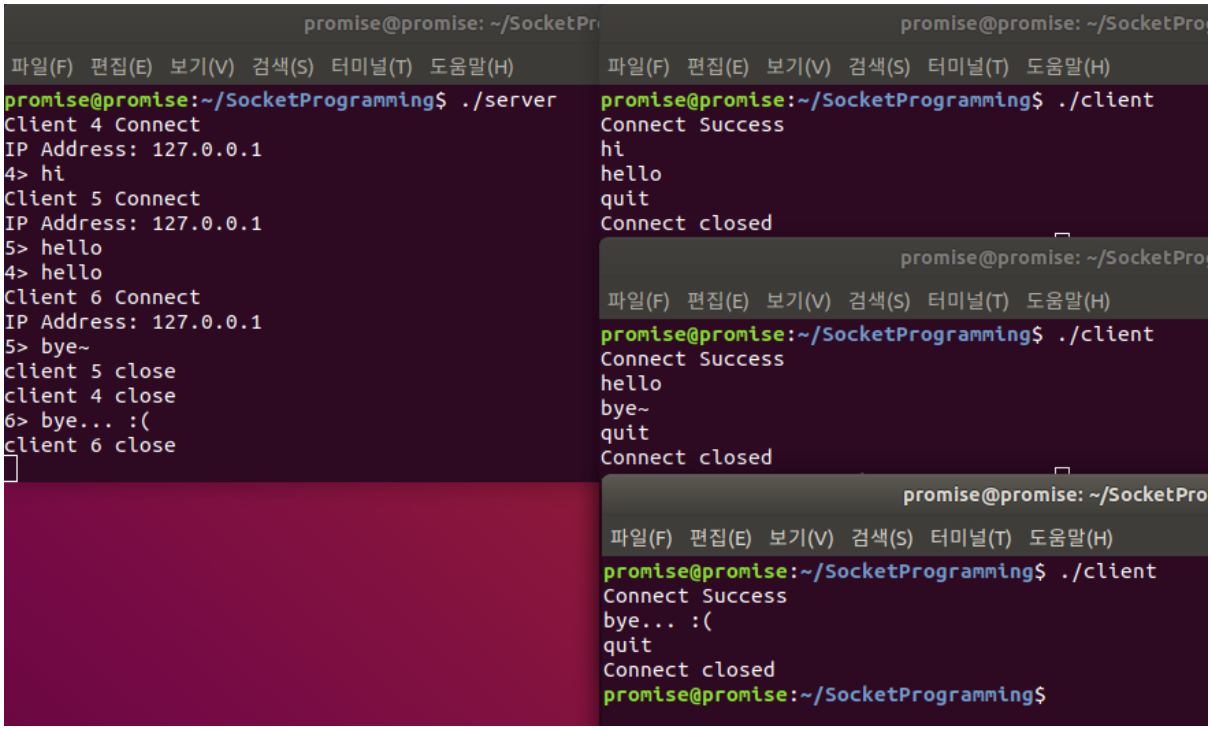
로컬에서 테스트를 진행한다. 터미널 창을 여러 개 띄워 각각 프로그램을 실행시킨다.



[1] server에 첫 번째 client 접속, client에서 "hi"라는 메시지 전송



[2] 두 번째 client 접속, 각 client에서 "hello"라는 메시지 전송



[3] 세 번째 client가 접속, 각각의 client에서 "quit" 전송

Server와 client에서 모두 socket이 닫히며 종료 메시지 출력