

[리눅스 환경 C언어]

UDP, TCP 통신 프로그램(클라이언트, 서버 총 4개)

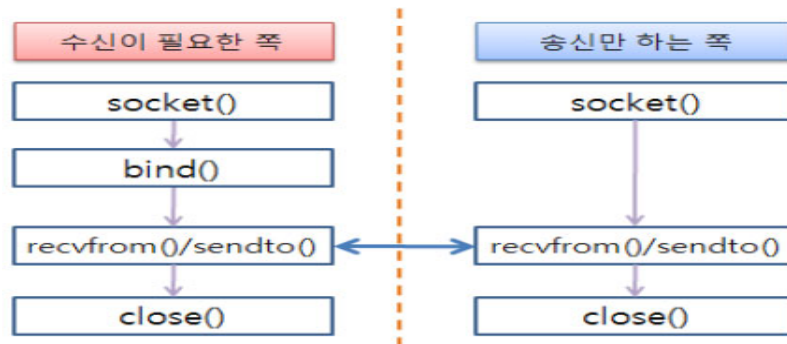
1. 프로그램 목록

- 1) UDP 서버 1개
 - 2) UDP 클라이언트 1개
 - 3) TCP 서버 1개
 - 4) TCP 클라이언트 1개
- 총 4개 프로그램(C언어 - 리눅스 환경)

2. 개념 정리

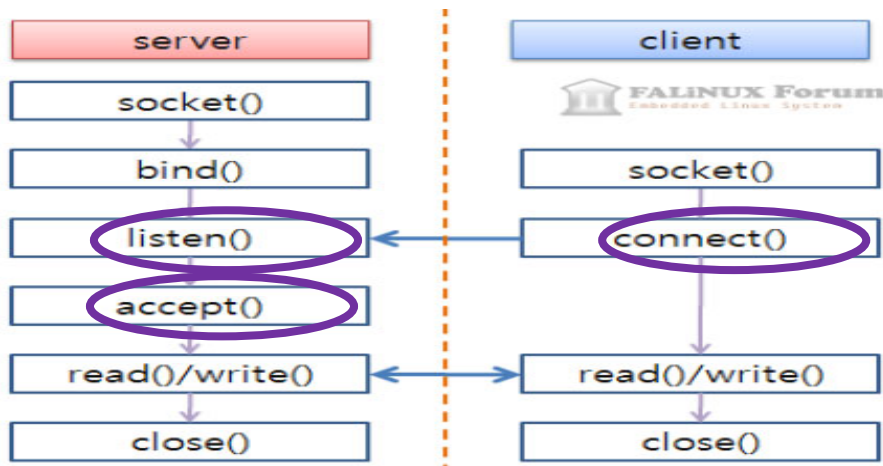
UDP(User Datagram Protocol)란?

: 인터넷에서 정보를 주고받을 때, 서로 주고받는 형식이 아닌 한쪽에서 일방적으로 보내는 방식의 통신 프로토콜. 비 연결지향 방식



TCP(Transmission Control Protocol)란?

: 서로 다른 운영체제를 쓰는 컴퓨터 간에도 데이터를 전송할 수 있어 인터넷에서 정보전송을 위한 표준 프로토콜로 사용됨. 연결지향 방식



3. 함수 설명

1) accept() 함수 : 소켓으로부터 연결을 받아들이는 함수로 아직 처리되지 않은 연결들이 대기하고 있는 큐에서 제일 처음 연결된 연결을 가져와서 새로운 소켓을 만든다. 연결에 성공하면 0보다 큰 파일 지정번호를 반환하고 실패(에러)시 -1을 반환한다.

int accept(int s, struct sockaddr *addr, socklen_t *addrlen); 형식으로 사용함

2) memset() 함수 : 메모리의 내용을 원하는 크기만큼 특정한 값으로 설정할 수 있으며 memory set의 줄임말이다.

memset(포인터, 설정할 값, 크기); 형식으로 사용함

3) write() 함수 : txt파일의 내용을 입력하는 함수. 리눅스에서는 파일과 소켓을 동일하게 취급하므로 소켓을 통해서 다른 호스트에게 데이터를 전송할 때도 사용함

4) read() 함수 : txt파일의 내용을 읽어오는 함수. 리눅스에서는 파일과 소켓을 동일하게 취급하므로 소켓을 통해서 다른 호스트에게 데이터를 수신할 때도 사용함

5) bzero() 함수 : 메모리 공간을 size 바이트만큼 0으로 채우는 함수

6) socket() 함수 : 소켓을 생성하는 함수

socket(domain, type, protocol) 형식으로 사용함

7) strlen() 함수 : char*가 가리키는 주소에서부터 시작해서 '\0' 문자가 나올 때까지의 문자들의 개수를 카운팅하여 최종 길이를 반환하는 함수

8) strcpy() 함수 : 문자열을 복사하는 함수

strcpy(대상 문자열, 원본 문자열) 형식으로 사용함

9) bind() 함수 : 소켓에 IP주소와 포트 번호를 지정해주는 함수로 소켓 통신 준비를 함

bind(int sockfd, struct *myaddr, socklen_t addrlen) 형식으로 사용

10) socket(int domain, int type, int protocol) 함수 : 소켓을 만드는 함수

- Domain : 프로토콜 체계 ex) PF_INET

- Type : 데이터 전송 방법 ex) SOCK_STREAM(TCP), SOCK_DGRAM(UDP)

- Protocol : 호스트와 호스트 사이에 사용할 규칙 ex) IPPROTO_TCP, IPPROTO_UDP

11) listen() 함수 : 클라이언트가 ServerSocket에 부여한 IP와 PORT로 접속했는지를 감시

int listen(int socket, int backlog) 형식으로 사용함

12) send() 함수 : 클라이언트로 데이터를 전송하는 함수

int send(int socket, const char FAR* buf, int len, int flags) 형식으로 사용함

13) recv() 함수 : send 함수를 이용해서 서버로 전달한 데이터를 읽어 들이는 함수
int recv(int socket, char FAR* addr, int len, int flags) 형식으로 사용함

14) send to() 함수 : 지정된 주소로 데이터를 보내는 기능의 함수
int sendto(int socket, const void *msg, int len, unsigned flags, const struct
sockaddr * addr, int addrlen) 형식으로 사용함

15) recvfrom() 함수 : sendto 함수를 이용해서 전달한 데이터를 읽어 들이는 함수
int recvfrom(int socket, const void *msg, int len, unsigned flags, const struct
sockaddr * addr, int addrlen) 형식으로 사용

4. 소스 코드(C)

1) TCP 서버(tcp_server)

```
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <unistd.h> //access, read, write 등의 함수를 위한 헤더
#include <stdio.h> //기본적인 입출력을 위한 헤더
#include <stdlib.h> //문자 변환 등을 위한 헤더
#include <string.h> //문자열을 위한 헤더
#define MAXLINE 1024 //문자 배열 크기 지정용 상수

int main(int argc, char **argv){
    int server_sockfd, client_sockfd; //소켓 통신을 위한 변수
    int state, client_len; //소켓 통신시 상태 및 길이 파악을 위한 변수
    pid_t pid;
    FILE *fp;
    struct sockaddr_un clientaddr, serveraddr; //소켓 통신을 위한
구조체
    char buf[MAXLINE]; //Client에서 보낸 메시지 저장할 배열
    char sendMsg[MAXLINE]; //Server가 보낼 메시지 저장할 배열
    //경로 설정이 제대로 되어있는지 인자를 확인 argc = 인자 개수
    if(argc != 2){
        printf("Usage : %s [socket file name]\n", argv[0]);
        printf("example : %s /tmp/mysocket\n", argv[0]);
        exit(0);
    }
    if(access(argv[1], F_OK) == 0){
```

```

        unlink(argv[1]);
    }
    client_len = sizeof(clientaddr); //길이 파악
/*AF_UNIX 프로토콜 사용, SOCK_STREAM(TCP) 방식으로 데이터 전송, 0
은 운영체제가 자동으로 소켓 타입에 맞게 설정하겠다는 뜻
*/
    if((server_sockfd = socket(AF_UNIX, SOCK_STREAM, 0))<0){
        perror("socket error : ");
        exit(0);
    }
    bzero(&serveraddr, sizeof(serveraddr)); //serveraddr의 사이즈만
    큼 0으로 대체
    serveraddr.sun_family = AF_UNIX;
    strcpy(serveraddr.sun_path, argv[1]); //문자열 복사
/*소켓에 IP주소와 포트번호를 지정하여 통신 준비를 함
bind시 setsockopt() 시스템 콜 호출 */
    state = bind(server_sockfd, (struct sockaddr *)&serveraddr,
sizeof(serveraddr)); //bind()는 실패하면 -1을 반환하므로 에러 처리
    if(state == -1)
    {
        perror("bind error : ");
        exit(0);
    }
    state = listen(server_sockfd, 5); //listen() -1 반환 시 에러처리
    if(state == -1)
    {
        perror("listen error : ");
        exit(0);
    }
    while(1){ //스레드처럼 연결될 때까지 계속 반복하고 있음
        client_sockfd = accept(server_sockfd, (struct sockaddr
*)&clientaddr, &client_len);
        pid = fork(); //fork()는 부모일 경우 자식 프로세스 PID, 자식일 경
우 0을 반환함
        if(pid ==0){ //pid가 0이라는 것은 현재 자식 프로세스라는 뜻
            if(client_sockfd == -1){
                perror("Accept error : ");
                exit(0);
            }
            while(1){
                memset(buf, 0x00, MAXLINE); //메모리 크기

```

설정

```

        //client에서 보낸 정보를 수신함
        if(read(client_sockfd, buf, MAXLINE) <= 0){
            close(client_sockfd);
            exit(0);
        }
        //Client에서 보낸 메시지 출력
        printf("client : %s\n", buf);
        //Server가 Client에게 송신할 메시지 입력
        printf("Put Message : ");
        scanf("%[^\n]", sendMsg);
        getchar(); //버퍼 초기화
        //Client에게 메시지 송신
        write(client_sockfd, sendMsg, sizeof(sendMsg));
    }}}
    close(client_sockfd); //연결을 해제함
}

```

2) TCP 클라이언트(tcp_client)

```

#include <sys/types.h>
#include <sys/stat.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <unistd.h> //access, read, write 등의 함수를 위한 헤더
#include <stdio.h> //기본적인 입출력을 위한 헤더
#include <stdlib.h> //문자 변환 등을 위한 헤더
#include <string.h> //문자열을 위한 헤더
#define MAXLINE 1024 //문자 배열 크기 지정용 상수
int main(int argc, char **argv){
    int client_len;
    int client_sockfd;
    FILE *fp_in;
    char buf_in[MAXLINE]; //Server가 송신하는 메시지 저장할 배열
    char buf_get[MAXLINE]; //Client가 송신할 메시지 저장할 배열
    char result;
    int i;
    struct sockaddr_un clientaddr;
    if(argc != 2){ //인자가 2개인지 확인 argc = 인자 개수
        printf("Usage : %s [socket file name]\n", argv[0]);
        printf("example : %s /tmp/mysocket\n", argv[0]);
        exit(0);
    }
    client_sockfd = socket(AF_UNIX, SOCK_STREAM, 0); //소켓 설정
}

```

```

        if(client_sockfd == -1){
            perror("error : "); //에러 처리
            exit(0);
        }
        bzero(&clientaddr, sizeof(clientaddr));
        clientaddr.sun_family = AF_UNIX; //sun_family 는 AF_UNIX를
뜻함
        strcpy(clientaddr.sun_path, argv[1]); //문자열 복사
        client_len = sizeof(clientaddr);
        if(connect(client_sockfd,(struct sockaddr *)&clientaddr,
client_len)<0)
        {
            perror("connect error: ");
            exit(0);
        }
        while(1)
        {
            memset(buf_in, 0x00, MAXLINE); //메모리 크기
            memset(buf_get, 0x00, MAXLINE);
            printf("Put Message : "); //Server로 송신할 메시지 입력
            fgets(buf_in, MAXLINE, stdin);
            //server에 전송
            write(client_sockfd, buf_in, strlen(buf_in));
            //server가 보낸 메시지 수신
            read(client_sockfd, buf_get, MAXLINE);
            printf("server : %s\n",buf_get); //수신받은 메시지를 출력함
        }
        close(client_sockfd); //연결을 해제함
        exit(0);
    }
}

```

3) UDP 서버(udp_server)

```

#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h> //IPv4 전용 기능 사용 헤더
#include <arpa/inet.h> //주소 변환 기능 사용 헤더
#include <unistd.h>
#define PORT 7777 // 포트 번호
#define BUFSIZE 1024 //문자 배열 크기 상수로 선언
main()

```

```

{
    int sockfd;
    struct sockaddr_in servAddr;
    struct sockaddr_in clntAddr;
    char recvBuffer[BUFSIZE]; //클라이언트로부터 받을 메시지 저장 배열
    char sendBuffer[BUFSIZE]; //클라이언트에게 전송할 메시지 저장 배열
    int clntLen;
    int recvLen; //수신할 메시지 크기 지정 변수
    int i = 0;
    int count = 0;
    int sendLen;
    /*AF_INET 프로토콜 사용, SOCK_DGRAM(UDP) 방식으로 데이터 전송, 0은
    운영체제가 자동으로 소켓 타입에 맞게 설정하겠다는 뜻. 인터넷으로 연결된
    프로세스 간에 통신하고 UDP 방법을 이용하는 소켓을 생성
    socket() 시스템 콜을 호출함
    */
    if((sockfd=socket(AF_INET, SOCK_DGRAM, 0)) == -1) { //에러 처리
        perror("socket failed");
        exit(1);
    }
    // servAddr를 0으로 초기화
    memset(&servAddr, 0, sizeof(servAddr));
    // servAddr에 IP 주소와 포트 번호를 저장
    servAddr.sin_family = AF_INET;
    servAddr.sin_addr.s_addr = htonl(INADDR_ANY);
    servAddr.sin_port = htons(PORT);
    /* sockfd 소켓에 주소 정보 연결
    bind 함수는 실패시 -1을 반환하기 때문에 -1일 경우 에러 처리*/
    if(bind(sockfd, (struct sockaddr*)&servAddr, sizeof(servAddr)) ==
    -1) {
        perror("bind failed");
        exit(1);
    }
    // 프로그램 중지시킬 때까지 무한 반복
    while(1) {
        clntLen = sizeof(clntAddr);
        /* sockfd 소켓으로 들어오는 데이터를 받아 recvBuffer에 저장하고
        클라이언트 주소 정보를 clntAddr에 저장 */
        if((recvLen=recvfrom(sockfd, recvBuffer, BUFSIZE-1, 0, (struct
        sockaddr*)&clntAddr, &clntLen)) == -1) {
            perror("recvfrom failed");
            exit(1);
        }
    }
}

```

```

recvBuffer[recvLen] = '\0';
// 받은 데이터를 출력
printf("Client: %s\n", recvBuffer);
//udp_client에게 받은 메시지를 출력하고 송신할 메시지 입력
printf("Input Message : ");
fgets(sendBuffer, BUFSIZE, stdin); //stdin은 표준 입력 버퍼
/*sendto함수를 이용하여 udp_client에게 메시지 송신
sendto() 시스템 콜 호출*/
if(sendto(sockfd, sendBuffer, strlen(sendBuffer), 0, (struct
sockaddr*)&clntAddr, sizeof(clntAddr)) != strlen(sendBuffer)) {
    perror("sendto failed");
    exit(1);
//입력한 메시지와 송신된 메시지 크기가 다르면 sendto 실패로 에러 처리
}
}
}

```

4) UDP 클라이언트(udp_client)

```

#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#define PORT 7777 // 서버의 포트 번호
#define BUFSIZE 1024 //문자 배열 크기 상수로 선언
// argv[1]은 수와 점 표기의 IP 주소
main(int argc, char *argv[])
{
    int sockfd;
    struct sockaddr_in servAddr;
//송신, 수신할 메시지 저장하는 문자 배열
    char sendBuffer[BUFSIZE], recvBuffer[BUFSIZE];
//송신, 수신할 메시지 크기 저장하는 변수
    int recvLen, servLen;
    /*argc=인자 수, 인자 수가 2개가 아니면 아래와 같은 방식으로 사용하라고
예문 출력*/
    if(argc != 2) {
        fprintf(stderr, "Usage: %s IP_address\n", argv[0]);
        exit(1);
    }
    /*AF_INET 프로토콜 사용, SOCK_DGRAM(UDP) 방식으로 데이터 전송,
0은 운영체제가 자동으로 소켓 타입에 맞게 설정하겠다는 뜻. 인터넷으로 연

```


결된 프로세스 간에 통신하고 UDP 방법을 이용하는 소켓을 생성

```
*/
    if((sockfd=socket(AF_INET, SOCK_DGRAM, 0)) == -1) {
        perror("sock failed");
        exit(1);
    }
    memset(&servAddr, 0, sizeof(servAddr)); //메모리 크기 지정
    // servAddr에 IP 주소와 포트 번호 저장
    servAddr.sin_family = AF_INET;
    servAddr.sin_addr.s_addr = inet_addr(argv[1]);
    servAddr.sin_port = htons(PORT);
    //Ctrl+c로 정지할 때까지 무한 반복
    while(1) {
        // 송신할 문자열 입력 받아 sendBuffer에 저장
        printf("Input Message : ");
        fgets(sendBuffer, BUFSIZE, stdin); //stdin은 표준 입력 버퍼

        /* sockfd 소켓을 통해 servAddr을 주소로 갖는 서버에게 데이터를
        보냄(sendto 함수를 사용해서). sendto() 시스템 콜 호출*/
        if(sendto(sockfd, sendBuffer, strlen(sendBuffer), 0, (struct
        sockaddr*)&servAddr, sizeof(servAddr)) != strlen(sendBuffer)) {
            perror("sendto failed");
            exit(1);
        }
        servLen = sizeof(servLen);
        // sockfd 소켓으로 들어오는 데이터를 받아 recvBuffer에 저장
        if((recvLen=recvfrom(sockfd, recvBuffer, BUFSIZE-1, 0, (struct
        sockaddr*)&servAddr, &servLen)) == -1) {
            perror("recvfrom failed");
            exit(1);
        }
        recvBuffer[recvLen] = '\0';
        // 서버가 송신한 문자열 출력
        printf("Server: %s\n", recvBuffer);
    }
    close(sockfd); //소켓 종료
    exit(0);
}
```

5. 실행 결과 스크린샷

1) TCP 프로그램 2개(tcp_server, tcp_client)

- i) 서버가 연결을 기다림
 - ii) 클라이언트가 연결을 시도
 - iii) 연결되면 클라이언트에서 메시지 송신 활성화 및 메시지 입력 후 송신
 - iv) 서버는 클라이언트가 보낸 메시지 수신 후 송신 활성화
 - v) 서버가 클라이언트로 메시지 송신
 - vi) 클라이언트는 서버가 보낸 메시지 수신 및 다시 송신 활성화
- ※ 서버와 클라이언트가 양방향으로 메시지를 주고받는 프로그램

1) tcp_client 대화 전송 전 커서 활성화 장면

```
king@localhost:~$ ./tcp_client /tmp/mysocket
Put Message : Hi server
server : Hi Client
Put Message : Is everything all right?
server : So far, nothing is wrong
Put Message : That's Great.
```

2) tcp_server가 메시지 수신 후 대화 전송 전 커서 활성화 장면

```
king@localhost:~$ ./tcp_server /tmp/mysocket
client : Hi server
Put Message : Hi Client
client : Is everything all right?
Put Message : So far, nothing is wrong
client : That's Great.
Put Message : Thanks for asking dude
```

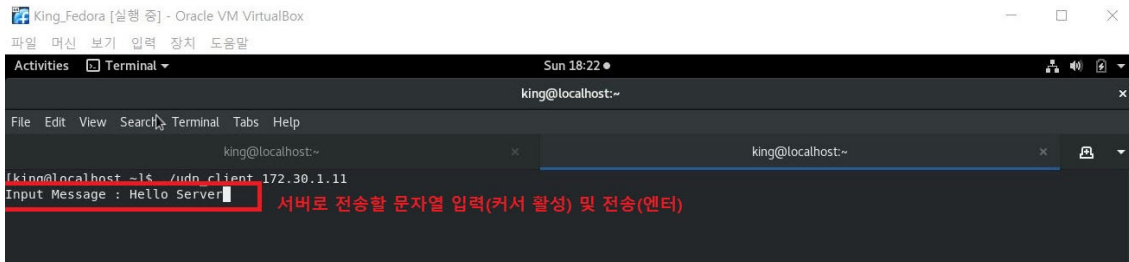
3) tcp_server가 보낸 메시지를 tcp_client가 수신 후에 다시 전송 활성화

```
king@localhost:~$ ./tcp_client /tmp/mysocket
Put Message : Hi server
server : Hi Client
Put Message : Is everything all right?
server : So far, nothing is wrong
Put Message : That's Great.
server : Thanks for asking dude
```

2) UDP 프로그램 2개(udp_server, udp_client)

- i) 소켓을 생성하여 IP주소와 포트 지정
 - ii) 클라이언트가 IP주소와 포트로 데이터 전송
 - iii) 서버는 클라이언트가 보낸 메시지 수신 후 송신 활성화
 - iv) 서버가 클라이언트로 메시지 송신(같은 방법 - sendto, recvfrom)
 - v) 클라이언트가 서버가 보낸 메시지 수신 후 다시 송신 활성화
- ※ 서버와 클라이언트가 양방향으로 메시지를 주고받는 프로그램

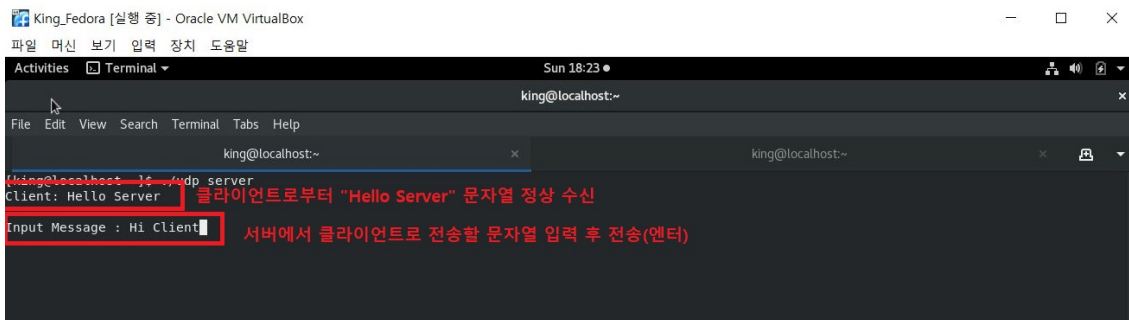
1) udp_client에서 server로 메시지 전송 전 커서 활성화 단계



```
King_Fedora [실행 중] - Oracle VM VirtualBox
파일  머신  보기  입력  장치  도움말
Activities  Terminal
Sun 18:22
king@localhost:~
File Edit View Search Terminal Tabs Help
king@localhost:~
king@localhost:~
[king@localhost ~]$ ./udp_client 172.30.1.11
Input Message : Hello Server
```

서버로 전송할 문자열 입력(커서 활성화) 및 전송(엔터)

2) udp_server 수신 후 대화 전송 전 커서 활성화 단계

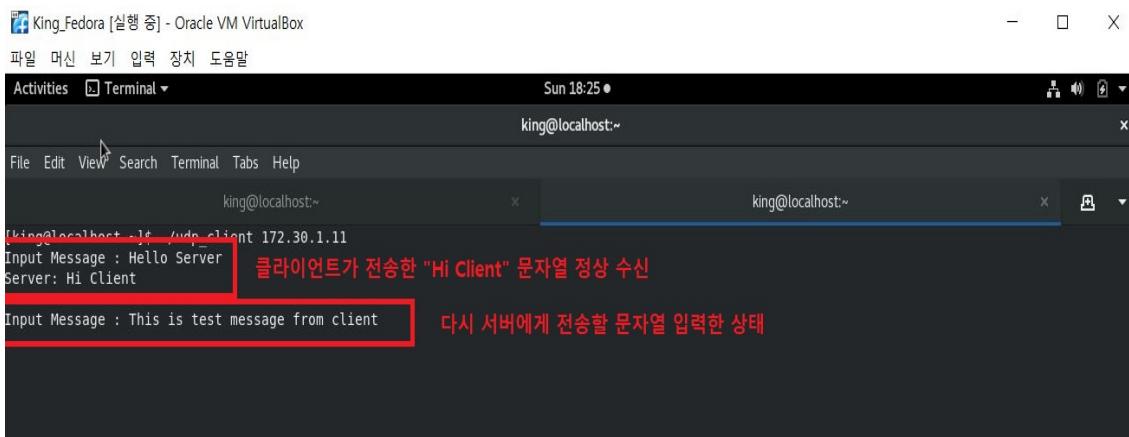


```
King_Fedora [실행 중] - Oracle VM VirtualBox
파일  머신  보기  입력  장치  도움말
Activities  Terminal
Sun 18:23
king@localhost:~
File Edit View Search Terminal Tabs Help
king@localhost:~
king@localhost:~
[king@localhost ~]$ ./udp_server
Client: Hello Server
Input Message : Hi Client
```

클라이언트로부터 "Hello Server" 문자열 정상 수신

서버에서 클라이언트로 전송할 문자열 입력 후 전송(엔터)

3) udp_client에서 서버가 전송한 대화 수신



```
King_Fedora [실행 중] - Oracle VM VirtualBox
파일  머신  보기  입력  장치  도움말
Activities  Terminal
Sun 18:25
king@localhost:~
File Edit View Search Terminal Tabs Help
king@localhost:~
king@localhost:~
[king@localhost ~]$ ./udp_client 172.30.1.11
Input Message : Hello Server
Server: Hi Client
Input Message : This is test message from client
```

클라이언트가 전송한 "Hi Client" 문자열 정상 수신

다시 서버에게 전송할 문자열 입력한 상태