

통신네트워크

학과 : 정보보안공학과

학번 : 202121556

이름 : 곽지현

1. LAB1

1.1 실습코드

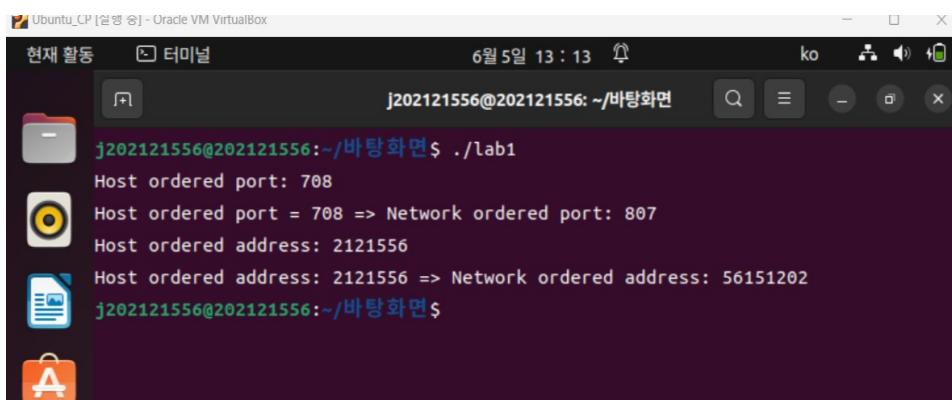
```
#include <stdio.h> // 표준 입출력 함수
#include <arpa/inet.h> // 인터넷 프로토콜

int main(int argc, char *argv[]) // 셸에서 입력되는 인자 갯수와 배열
{
    unsigned short host_port=0x0708; // host_port 변수 선언
    unsigned short net_port; // net_port 변수 선언
    unsigned long host_addr=0x02121556; // host_addr 변수 선언
    unsigned long net_addr; // net_addr 변수 선언

    net_port=htons(host_port); // 포트 번호를 호스트 순서에서 네트워크 순서로 변환하여
                               // net_port에 대입
    net_addr=htonl(host_addr); // ip 주소를 호스트 순서에서 네트워크 순서로 변환하여
                               // net_addr에 대입

    printf("Host ordered port: %x \n", host_port); // 호스트 순서로 정렬된 포트 출력
    printf("Host ordered port = %x => Network ordered port: %x \n", host_port, net_port);
    // 호스트 순서로 정렬된 포트와 네트워크 순서로 정렬된 포트 출력
    printf("Host ordered address: %lx \n", host_addr); // 호스트 순서로 정렬된 주소 출력
    printf("Host ordered address: %lx => Network ordered address: %lx \n", host_addr,
           net_addr); // 호스트 순서로 정렬된 주소와 네트워크 순서로 정렬된 주소 출력
    return 0;
}
```

1.2 출력 결과



The screenshot shows a terminal window titled '터미널' (Terminal) on an Ubuntu desktop environment. The window title bar includes the text 'Ubuntu_01 [실행 중] - Oracle VM VirtualBox'. The terminal window displays the following command and its output:

```
j202121556@202121556:~/바탕화면$ ./lab1
Host ordered port: 708
Host ordered port = 708 => Network ordered port: 807
Host ordered address: 2121556
Host ordered address: 2121556 => Network ordered address: 56151202
j202121556@202121556:~/바탕화면$
```

2. LAB2

2.1 실습코드

```
#include <stdio.h> // 표준 입출력 함수
#include <stdlib.h> // 표준 라이브러리 함수
#include <string.h> // 문자열 처리 함수
#include <arpa/inet.h> // 인터넷 프로토콜
```

```

void error_handling(char *message); // 에러 처리 함수

int main(int argc, char *argv[]) // 쉘에서 입력되는 인자 갯수와 배열
{
    char *addr="010.189.060.120"; // 변환할 IP 주소 선언
    struct sockaddr_in addr_inet; // IP 주소 구조체

    if(!inet_aton(addr, &addr_inet.sin_addr)) // IP 주소를 네트워크 순서로 변환하여
                                                // addr_inet.sin_addr에 저장
        error_handling("Conversion error"); // 변환에 실패한 경우 호출되는 함수
    else // 함수가 성공한 경우 실행된다.
        printf("[aton function] Network Address %s => Network ordered integer addr:
%#x", addr, addr_inet.sin_addr.s_addr); // addr과 변환된 IP 주소 출력

    struct sockaddr_in addr1, addr2; // sockaddr_in 구조체를 사용하는 addr1, addr2
                                    // 변수 선언
    char *str_ptr; // str_ptr 포인터 변수 선언
    char *str_ptr2 // str_ptr2 포인터 변수 선언

    addr1.sin_addr.s_addr=htonl(0x1020304); // 0x1020304를 네트워크 순서로 변환하여
                                              // addr1.sin_addr.s_addr에 대입
    addr2.sin_addr.s_addr=htonl(0x1010101); // 0x1010101을 네트워크 순서로 변환하여
                                              // addr2.sin_addr.s_addr에 대입

    str_ptr = inet_ntoa(addr1.sin_addr); // addr1.sin_addr를 IP 주소로 변환하여 str_ptr에
                                         // 대입

    printf("[ntoa function] addr 1 = %x Dotted-Decimal: %s %#n". addr1.sin_addr.s_addr,
str_ptr);
    // 네트워크 주소로 변환값과 IP 주소로 변환된 값을 출력
    str_ptr2 = inet_ntoa(addr2.sin_addr); // addr2.sin_addr를 IP 주소로 변환하여 str_ptr2에
                                         // 대입
    printf("[ntoa function] addr 2 = %x Dotted-Decimal: %s %#n", addr2.sin_addr.s_addr,
str_ptr2);
    // 네트워크 주소로 변환값과 IP 주소로 변환된 값을 출력
    return 0;
}

void error_handling(char *message) // 에러 처리 함수
{
    fputs(message, stderr); // 에러 메시지를 표준 에러 스트림에 출력
    fputc('\n', stderr); // 개행 문자를 표준 에러 스트림에 출력
    exit(1); // 프로그램 종료
}

```

2.2 출력 결과

```
j202121556@202121556:~/바탕화면$ ./lab2
[aton function] Network Address 010.189.060.120 => Network ordered integer addr
: 7830bd08
[ntoa function] addr 1 = 4030201 Dotted-Decimal: 1.2.3.4
[ntoa function] addr 2 = 1010101 Dotted-Decimal: 1.1.1.1
j202121556@202121556:~/바탕화면$
```

3. LAB3

3.1 실습코드

```
// TCP echo server
#include <stdio.h> // 표준 입출력 함수
#include <stdlib.h> // 표준 라이브러리 함수
#include <string.h> // 문자열 처리 함수
#include <unistd.h> // 유닉스 표준
#include <arpa/inet.h> // 인터넷 프로토콜
#include <sys/socket.h> // 소켓 통신 함수

#define BUF_SIZE 1024 // 1024 바이트 버퍼 크기 사용

void error_handling(char *message); // 에러 처리 함수

int main(int argc, char *argv[]) // 셸에서 입력되는 인자 갯수와 배열
{
    int serv_sock, clnt_sock; // 서버 소켓과 클라이언트 소켓 2 개 필요
    char message[BUF_SIZE]; // 메시지 수신용 버퍼 1024 바이트
    int str_len, i; // 수신한 데이터의 길이와 제어 변수 i
```

```

struct sockaddr_in serv_addr; // 서버용 소켓
struct sockaddr_in clnt_addr; // 클라이언트 소켓
socklen_t clnt_adr_sz; // 클라이언트 주소 구조체의 바이트 크기

if(argc!=2) { // 아규먼트가 2 개가 아니면, shell 에서 포트 번호를 지정하지 않았으면
    printf("Usage : %s <port>\n", argv[0]); // 사용자에게 사용법 안내
    exit(1); // 비정상 종료 처리
}

serv_sock=socket(PF_INET, SOCK_STREAM, 0); // TCP 통신용 서버 소켓 생성

if(serv_sock== -1) // 생성 실패하면
    error_handling("socket() error"); // 에러 표시

memset(&serv_addr, 0, sizeof(serv_addr)); // 서버 구조체를 0 으로 채움
serv_addr.sin_family=AF_INET; // 서버 주소 체계를 인터넷 계열로 설정
serv_addr.sin_addr.s_addr=htonl(INADDR_ANY); // 서버는 주소 필요 없음
serv_addr.sin_port=htons(atoi(argv[1])); // 서버가 사용할 포트 번호

if(bind(serv_sock, (struct sockaddr*)&serv_addr, sizeof(serv_addr)) == -1)
// bind() 함수는 소켓과 주소를 묶어줌
    error_handling("bind() error"); // 에러 표시
if(listen(serv_sock, 5)== -1) // listen() 함수는 소켓을 서버용으로 사용할 수 있게 함
    error_handling("listen() error"); // 에러 표시

clnt_adr_sz=sizeof(clnt_addr); // 클라이언트 주소 구조체의 크기

for(i=0; i<5; i++) // 5 번만 클라이언트의 접속을 받음
{
    clnt_sock=accept(serv_sock, (struct sockaddr*)&clnt_addr, &clnt_adr_sz);
    // accept() 함수를 호출하여 클라이언트의 연결 요청을 기다리고, 연결이 수락되면 새로운 소켓인 clnt_sock 을 생성
    if(clnt_sock== -1) // 연결 실패하면
        error_handling("accept() error"); // 에러 표시
    else // 연결이 성공한 경우
        printf("Connected client %d\n", i+1); // 해당 클라이언트의 번호를 출력

    while((str_len=read(clnt_sock, message, BUF_SIZE))!=0)
        write(clnt_sock, message, str_len); // 받은 메시지를 클라이언트에게 다시 보냄

    close(clnt_sock); // 클라이언트와 접속이 끊겼으면 클라이언트 소켓을 닫음
}

close(serv_sock); // 5 번 클라이언트를 받아주고 나면 서버 소켓을 닫고 끝냄
return 0;

```

```

}

void error_handling(char *message) // 에러 처리 함수
{
    fputs(message, stderr); // 에러 메시지를 표준 에러 스트림에 출력
    fputc('\n', stderr); // 개행 문자를 표준 에러 스트림에 출력
    exit(1); // 프로그램 종료
}

// TCP echo client
#include <stdio.h> // 표준 입출력 함수
#include <stdlib.h> // 표준 라이브러리 함수
#include <string.h> // 문자열 처리 함수
#include <unistd.h> // 유닉스 표준
#include <arpa/inet.h> // 인터넷 프로토콜
#include <sys/socket.h> // 소켓 통신 함수

#define BUF_SIZE 1024 // 1024 바이트 버퍼 크기 사용

void error_handling(char *message); // 에러 처리 함수

int main(int argc, char *argv[]) // 셸에서 입력되는 인자 갯수와 배열
{
    int sock; // socket file descriptor 변수 선언
    char message[BUF_SIZE]; // 전송할 메시지 버퍼 선언
    int str_len, recv_len, recv_cnt; // 문자열 길이, 수신된 데이터의 길이, 수신 횟수

    struct sockaddr_in serv_addr; // socket_in 주소 구조체 형태 serv_addr 변수 선언

    if(argc!=3) { // 아규먼트가 3 개가 아니면, shell에서 포트 번호를 지정하지 않았으면
        printf("Usage : %s <IP> <port>\n", argv[0]); // 사용자에게 사용법 안내
        exit(1); // 비정상 종료 처리
    }

    sock=socket(PF_INET, SOCK_STREAM, 0); // TCP 소켓 생성
    if(sock== -1) // 생성 실패하면
        error_handling("socket() error"); // 에러 표시

    memset(&serv_addr, 0, sizeof(serv_addr)); // 서버 구조체를 0으로 채움
    serv_addr.sin_family=AF_INET; // 서버 주소 체계를 인터넷 계열로 설정
    serv_addr.sin_addr.s_addr=inet_addr(argv[1]); // serv_addr 구조체의 IP 주소를 명령줄 인수
    //로 전달 된 첫 번째 인수로 설정
    serv_addr.sin_port(htons(atoi(argv[2]))); // serv_addr 구조체의 IP 주소를 명령줄 인수로 전달
    //된 두 번째 인수로 설정

    if(connect(sock, (struct sockaddr*)&serv_addr, sizeof(serv_addr)) == -1)

```

```

// 소켓을 서버에 연결, connect() 함수를 사용하여 소켓과 서버 주소를 전달
error_handling("connect() error!"); // 에러 표시
else // 연결에 성공한 경우
    puts("Connected....."); // 메시지 출력

while(1)
{
    fputs("Input message(Q to quit): ", stdout); // 터미널에서 사용자로부터 전송
                                                // 메시지 입력
    fgets(message, BUF_SIZE, stdin); // 입력받은 메시지를 message 버퍼에 저장
    if(strcmp(message,"q\n") || !strcmp(message,"Q\n"))
        // 입력받은 메시지가 'q' 또는 'Q' 인 경우
        break; // 종료

    str_len=write(sock, message, strlen(message)); // 예코 서버에 전송하고 사이즈
                                                // 계산
    recv_len=0; // 수신된 데이터 길이를 초기화
    while(recv_len<str_len)
        // 수신된 데이터 길이가 전체 데이터 길이보다 작은 동안 루프 반복
        {
            recv_cnt=read(sock, &message[recv_len], BUF_SIZE-1);
            // 서버로부터 수신받은 메시지를 읽은 함수
            if(recv_cnt== -1) // 데이터 읽기 도중 오류가 발생한 경우
                error_handling("read() error!"); // 에러 표시
            recv_len+=recv_cnt; // 수신된 데이터 길이를 업데이트
        }
    message[recv_len]=0; // 수신된 메시지 끝에 NULL 문자를 추가
    printf("Message from server: %s", message); // 서버로부터 수신된 메시지를 출력
}

close(sock); // 소켓 닫기
return 0;
}

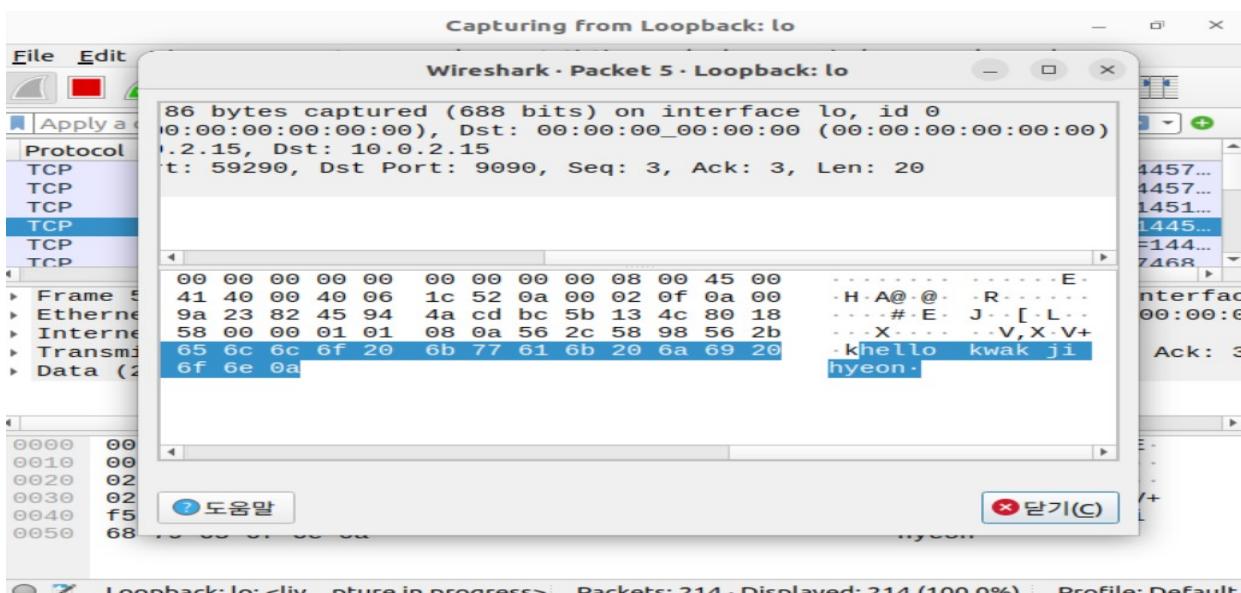
void error_handling(char *message) // 에러 처리 함수
{
    fputs(message, stderr); // 에러 메시지를 표준 에러 스트림에 출력
    fputc('\n', stderr); // 개행 문자를 표준 에러 스트림에 출력
    exit(1); // 프로그램 종료
}

```

3.2 출력 결과

```
j202121556@202121556: ~/바탕화면 $ ./lab3_c 10.0.2.15 9090
Connected.....
Input message(Q to quit): hello, I'm kwak ji hyeon
Message from server: hello, I'm kwak ji hyeon
Input message(Q to quit):
```

3.3 와이어샤크 확인 결과



4. LAB4

4.1 실습코드

```
// TCP 계산기 server
#include <stdio.h> // 표준 입출력 함수
#include <stdlib.h> // 표준 라이브러리 함수
#include <string.h> // 문자열 처리 함수
#include <unistd.h> // 유닉스 표준
#include <arpa/inet.h> // 인터넷 프로토콜
#include <sys/socket.h> // 소켓 통신 함수

#define BUF_SIZE 1024 // 1024 바이트 버퍼 크기 사용
#define OPSZ 4 // 피연산자의 크기를 4로 사용

void error_handling(char *message); // 에러 처리
int calculate(int opnum, int opnds[], char oprator);
```

```

// 계산 (계산할 숫자 개수, 계산할 숫자 배열, 연산자+,-,*)

int main(int argc, char *argv[]){ // 쉘에서 입력되는 인자 갯수와 배열
    int serv_sock, clnt_sock; // 서버, 클라이언트 소켓 디스크립터
    char opinfo[BUF_SIZE]; // char 형 배열을 선언
    int result, opnd_cnt, i; // 계산에 관련된 int 변수
    int recv_cnt, recv_len;

    struct sockaddr_in serv_adr, clnt_adr; // 서버, 클라이언트 주소 설정 구조체
    socklen_t clnt_adr_sz; // 클라이언트 주소 셋팅 구조체 사이즈

    if (argc != 2) { // 아규먼트가 2개가 아니면, shell에서 포트 번호를 지정하지 않았으면
        printf("usage : %s <port> \n", argv[0]); // 사용자에게 사용법 안내
        exit(1); // 비정상 종료 처리
    }

    serv_sock = socket(PF_INET, SOCK_STREAM, 0); // tcp 소켓 생성

    if (serv_sock == -1) { // 생성 실패하면
        error_handling("socket error"); // 에러 표시
    }

    memset(&serv_adr, 0, sizeof(serv_adr)); // 서버 소켓에 주소 할당할 구조체 0 초기화
    serv_adr.sin_family = AF_INET; // 서버 주소 체계를 인터넷 계열로 설정
    serv_adr.sin_addr.s_addr = htonl(INADDR_ANY); // 서버는 주소 필요 없음
    serv_adr.sin_port = htons(atoi(argv[1])); // 서버가 사용할 포트 번호

    if (bind(serv_sock, (struct sockaddr*)&serv_adr, sizeof(serv_adr)) == -1) {
        // bind() 함수는 소켓과 주소를 묶어줌
        error_handling("bind() error"); // 에러 표시
    }

    if (listen(serv_sock, 5) == -1) { // listen() 함수는 소켓을 서버용으로 사용할 수 있게 함
        error_handling("listen() error"); // 에러 표시
    }

    char_adr_sz = sizeof(clnt_adr); // 클라이언트 주소 구조체의 크기

    for (i = 0; i < 5; i++) { // 5번만 클라이언트의 접속을 받음
        opnd_cnt = 0; // 총 몇 개의 숫자를 연산할 것인지
        clnt_sock = accept(serv_sock, (struct sockaddr*)&clnt_adr, &clnt_adr_sz);
        // 클라이언트로의 접속을 받음
        printf("accept success \n"); // 메시지 출력

        read(clnt_sock, &opnd_cnt, 1); // 클라이언트로부터 피연산자의 개수를 읽어옴
        printf("opnd_cnt : %d \n", opnd_cnt); // 메시지 출력
        recv_len = 0; // 모든 피연산자와 연산 종류의 총 바이트 수
    }
}

```

```

while ((opnd_cnt * OPSZ +1) > recv_len)
// 피연산자와 연산 종류를 모두 읽어들일 때까지 read 함수 반복 호출
{
    recv_cnt = read(clnt_sock, &opinfo[recv_len], BUF_SIZE-1);
    // 클라이언트로부터 데이터를 읽어옴
    printf("recv_cnt : %d \n", recv_cnt);
    recv_len += recv_cnt;
    printf("recv_len : %d \n", recv_len);
}
printf("opinfo [0] : %d \n", ((int*)opinfo)[0]); // 연산할 첫 번째 숫자
printf("opinfo [1] : %d \n", ((int*)opinfo)[1]); // 연산할 두 번째 숫자
char operatorSign = opinfo[recv_len-1];
// 배열의 제일 마지막에는 +, -, * 연산자가 1 바이트 형태로 들어감
printf("operatorSign : %c \n", operatorSign);

result = calculate(opnd_cnt, (int*)opinfo, operatorSign);
// 연산 : 연산할 숫자 개수, 클라이언트가 보낸 연산할 숫자가 담긴 배열
write(clnt_sock, (char*)&result, sizeof(result));
// 서버는 연산 결과를 4 바이트의 정수 형태로 클라이언트에게 전달

close(clnt_sock); // 클라이언트 소켓 닫기
}
close(serv_sock); // 서버 소켓 닫기
return 0;
}

int calculate(int opnum, int opnds[], char op) { // calcuate 함수 호출
printf("calculate enter \n"); // 메시지 출력
int result = opnds[0], i; // 결과 변수를 첫 번째 피연산자로 초기화
printf("opnds[0] : %d \n", result); // 초기값을 출력

switch (op) {
    case '+': // 덧셈 연산일 경우
        for (i = 1; i < opnum; i++) {
            result+=opnds[i];
        }
        break; // 종료
    case '-': // 뺄셈 연산일 경우
        for (i = 1; i < opnum; i++) {
            result-=opnds[i];
        }
        break; // 종료
    case '*': // 곱셈 연산일 경우
        for (i = 1; i < opnum; i++) {
            result*=opnds[i];
        }
}

```

```

        break; // 종료
    }
    printf("result : %d \n", result); // 최종 결과 값을 출력
    return result; // 계산 결과를 반환
}

void error_handling(char *message) // 에러 처리 함수
{
    fputs(message, stderr); // 에러 메시지를 표준 에러 스트림에 출력
    fputc('\n', stderr); // 개행 문자를 표준 에러 스트림에 출력
    exit(1); // 프로그램 종료
}

// TCP 계산 요청 client
#include <stdio.h> // 표준 입출력 함수
#include <stdlib.h> // 표준 라이브러리 함수
#include <string.h> // 문자열 처리 함수
#include <unistd.h> // 유닉스 표준
#include <arpa/inet.h> // 인터넷 프로토콜
#include <sys/socket.h> // 소켓 통신 함수

#define BUF_SIZE 1024 // 1024 바이트 버퍼 크기 사용
#define RLT_SIZE 4 // 결과의 크기를 4로 사용
#define OPSZ 4 // 피연산자의 크기를 4로 사용

void error_handling(char *message); // 에러 처리

int main(int argc, char *argv[]) // 헬에서 입력되는 인자 갯수와 배열
{
    int sock; // 소켓 디스크립터
    char opmsg[BUF_SIZE]; // char 형 배열 opmsg 를 선언
    int result, opnd_cnt, i; // 연산을 위한 int 변수

    struct sockaddr_in serv_addr; // 소켓에 할당할 주소 구조체

    if (argc != 3) { // 아규먼트가 3개가 아니면, shell 에서 포트 번호를 지정하지 않았으면
        printf("usage : %s <ip> <port> \n", argv[0]); // 사용자에게 사용법 안내
        exit(1); // 비정상 종료 처리
    }
    sock=socket(PF_INET, SOCK_STREAM, 0); // 소켓 생성
    if(sock== -1) // 생성 실패하면
        error_handling("socket() error"); // 에러 표시

    memset(&serv_addr, 0, sizeof(serv_addr)); // 소켓에 할당할 주소 구조체 메모리 할당후, 0 으
                                                // 로 초기화
    serv_addr.sin_family=AF_INET; // 서버 주소 체계를 AF_INET 로 설정
    serv_addr.sin_addr.s_addr=inet_addr(argv[1]);
}

```

```

// 서버의 IP 주소를 사용자가 입력한 첫 번째 인자로 설정
serv_adr.sin_port=htons(atoi(argv[2]));
// 서버의 포트 번호를 사용자가 입력한 두 번째 인자로 설정

if(connect(sock, (struct sockaddr*)&serv_adr, sizeof(serv_adr))==-1) // 연결 실패
    error_handling("connect() error!"); // 에러 표시
else // 연결 성공
    puts("Connected....."); // 메시지 출력

fputs("Operand count: ", stdout); // 서버에 연결되면 여기까지 출력됨
scanf("%d", &opnd_cnt); // 사용자로부터 데이터를 입력받음
opmsg[0]=(char)opnd_cnt; // 클라이언트는 서버에 접속하자마자 피연산자의 개수정보를
                           // 1바이트 정수 형태로 전달

for(i = 0; i<opnd_cnt; i++)
// 클라이언트가 서버에 전달하는 정수 하나는 4바이트로 표현
{
    printf("Operand %d: ", i+1); // 콘솔에 출력
    scanf("%d", (int*)&opmsg[i*OPSZ+1]); // 표준 입력받기
}
int enter = fgetc(stdin); // stream 으로부터 1바이트의 데이터를 읽음
printf("enter : %d \n", enter);

fputs("Operator: ", stdout); // 연산자의 종류를 전달
scanf("%c", &opmsg[opnd_cnt*OPSZ+1]);
// 배열의 제일 마지막 인덱스에 연산자 1바이트로 넣음

write(sock, opmsg, opnd_cnt*OPSZ+2); // 계산 관련 정보를 한번에 묶어 서버로 전달
read(sock, &result, RLT_SIZE);
// 서버는 연산결과를 4바이트 정수의 형태로 클라이언트에게 전달

printf("Operation result: %d \n", result); // 결과 출력
close(sock); // 소켓 닫음
return 0;
}

void error_handling(char *message) // 에러 처리 함수
{
    fputs(message, stderr); // 에러 메시지를 표준 에러 스트림에 출력
    fputc('\n', stderr); // 개행 문자를 표준 에러 스트림에 출력
    exit(1); // 프로그램 종료
}

```

4.2 출력 결과

Ubuntu_CP [실행 중] - Oracle VM VirtualBox

현재 활동 터미널 6월 6일 16 : 22 ko

j202121556@202121556: ~/바탕화면

```
j202121556@202121556:~/바탕화면$ ./lab4.c 10.0.2.15 9090
Connected.....
Operand count:3
Operand 1: 02
Operand 2: 07
Operand 3: 08
enter : 10
Operator:+
Operation result: 17
```

Ubuntu_CP [실행 중] - Oracle VM VirtualBox

현재 활동 터미널 6월 6일 16 : 23 ko

j202121556@202121556: ~/바탕화면

```
j202121556@202121556:~/바탕화면$ ./lab4.c 10.0.2.15 9090
Connected.....
Operand count:3
Operand 1: 02
Operand 2: 07
Operand 3: 08
enter : 10
Operator:-
Operation result: -13
```

Ubuntu_CP [실행 중] - Oracle VM VirtualBox

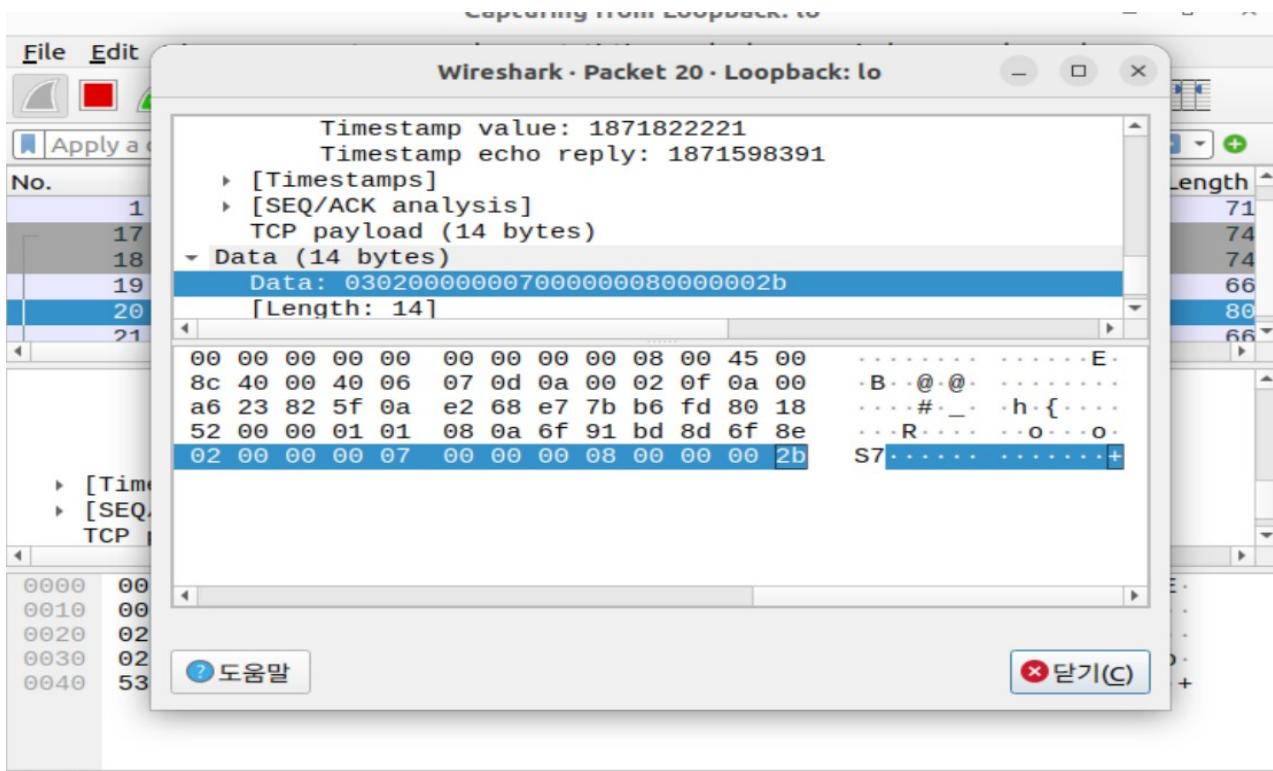
현재 활동 터미널 6월 6일 16:23 ko

j202121556@202121556: ~/바탕화면

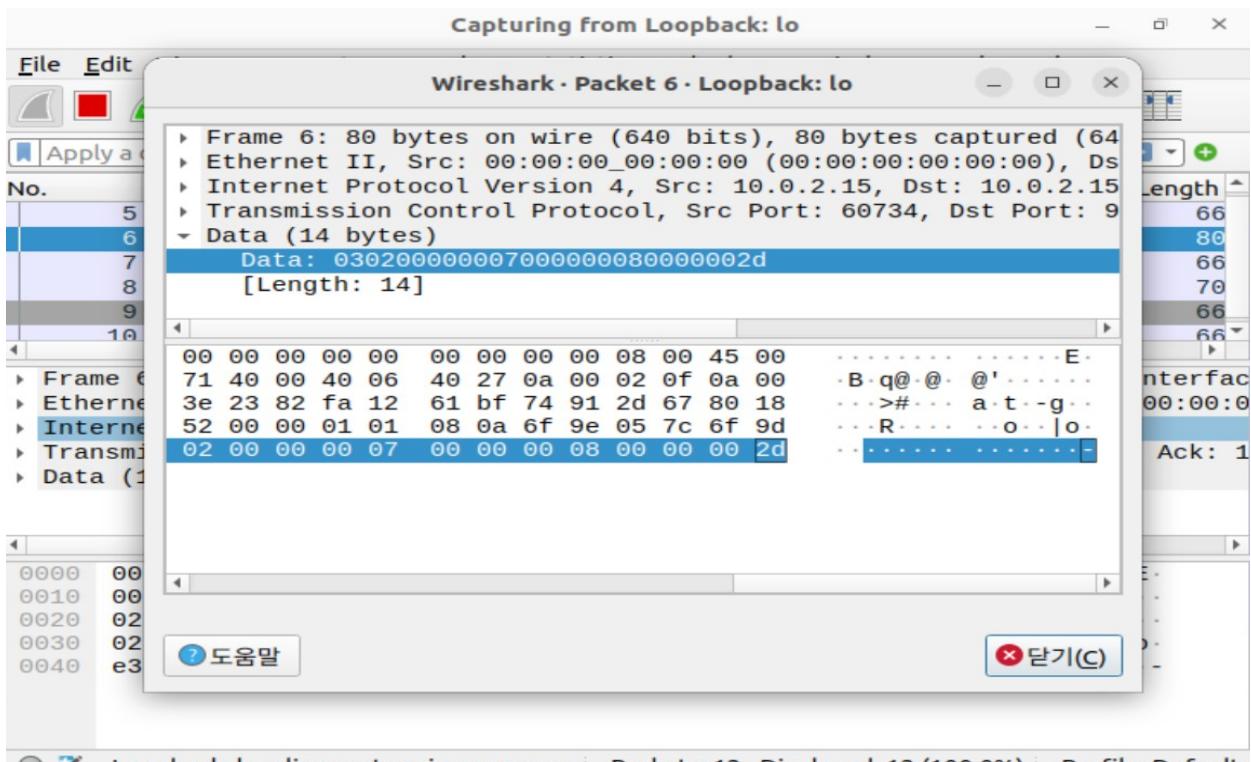
```
j202121556@202121556:~/바탕화면$ ./lab4.c 10.0.2.15 9090
Connected.....
Operand count:3
Operand 1: 02
Operand 2: 07
Operand 3: 08
enter : 10
Operator:*
Operation result: 112
```

4.3 와이어샤크 확인 결과

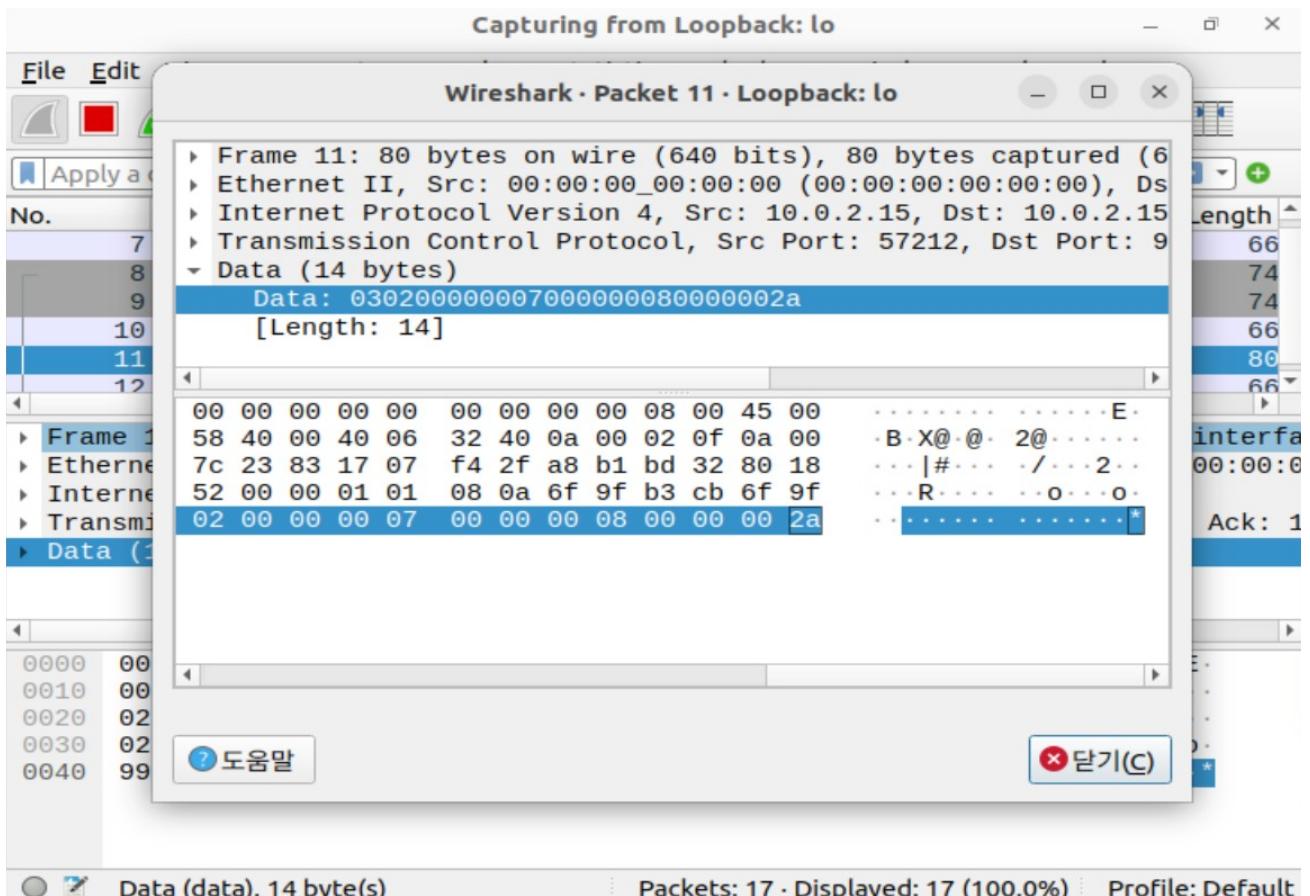
(+)



(-)



(*)



5. LAB5

5.1 실습코드

```

// UDP Server

#include <stdio.h> // 표준 입출력 함수
#include <stdlib.h> // 표준 라이브러리 함수
#include <string.h> // 문자열 처리 함수
#include <unistd.h> // 유닉스 표준
#include <arpa/inet.h> // 인터넷 프로토콜
#include <sys/socket.h> // 소켓 통신 함수

#define BUF_SIZE 10 // 10 바이트 버퍼 크기 사용

void error_handling(char *message); // 에러 처리

int main(int argc, char *argv[]) { // 쉘에서 입력되는 인자 갯수와 배열
    int serv_sock; // 소켓 디스크립터
    char message[BUF_SIZE]; // 보낼 메시지를 담는 배열형태의 변수
    int str_len; // 클라이언트로부터 수신 받은 문자열 길이
    socklen_t clnt_adr_sz; // 클라이언트 주소 셋팅 구조체 사이즈

    struct sockaddr_in serv_adr, clnt_adr; // 서버, 클라이언트 주소 설정 구조체

```

```

if (argc != 2) { // 아규먼트가 2 개가 아니면, shell 에서 포트 번호를 지정하지 않았으면
    printf("Usage : %s <port>\n", argv[0]); // 사용자에게 사용법 안내
    exit(1); // 비정상 종료 처리
}

serv_sock = socket(PF_INET, SOCK_DGRAM, 0); // UDP 소켓 생성

if (serv_sock == -1) { // 생성 실패
    error_handling("udp socket creation error"); // 에러 표시
}

memset(&serv_adr, 0, sizeof(serv_adr)); // 서버 주소 정보 초기화
serv_adr.sin_family = AF_INET; // 서버 주소 체계를 인터넷 계열로 설정
serv_adr.sin_addr.s_addr = htonl(INADDR_ANY); // 서버는 주소 필요 없음
serv_adr.sin_port = htons(atoi(argv[1])); // 서버가 사용할 포트 번호

if (bind(serv_sock, (struct sockaddr*)&serv_adr, sizeof(serv_adr)) == -1) {
    // bind() 함수는 소켓과 주소를 묶어줌
    error_handling("bind creation error"); // 에러 표시
}

while (1) {
    clnt_adr_sz = sizeof(clnt_adr);
    // 클라이언트 주소 구조체의 크기를 계산하여 변수에 저장
    memset(message, 0, BUF_SIZE); // message 버퍼를 0 으로 초기화

    str_len = recvfrom(serv_sock, message, BUF_SIZE, 0, (struct sockaddr*)&clnt_adr,
    &clnt_adr_sz); // 클라이언트로부터 NULL 문자를 제외하고 문자열 수신
    printf("echo count from client : %d\n", str_len); // 문자열 갯수 출력
    printf("receive message : %s\n", message); // 받은 문자 출력

    sendto(serv_sock, message, str_len, 0, (struct sockaddr*)&clnt_adr, clnt_adr_sz);
    // 수신 받은 데이터를 다시 클라이언트로 송신
}
close(serv_sock); // 소켓 닫음
return 0;
}

void error_handling(char *message) // 에러 처리 함수
{
    fputs(message, stderr); // 에러 메시지를 표준 에러 스트림에 출력
    fputc('\n', stderr); // 개행 문자를 표준 에러 스트림에 출력
    exit(1); // 프로그램 종료
}

// UDP client
#include <stdio.h> // 표준 입출력 함수

```

```

#include <stdlib.h> // 표준 라이브러리 함수
#include <string.h> // 문자열 처리 함수
#include <unistd.h> // 유닉스 표준
#include <arpa/inet.h> // 인터넷 프로토콜
#include <sys/socket.h> // 소켓 통신 함수

#define BUF_SIZE 10 // 10 바이트 버퍼 크기 사용

void error_handling(char *message); // 에러 처리

int main(int argc, char *argv[]) { // 쉘에서 입력되는 인자 갯수와 배열
    int sock; // 소켓 디스크립터
    char message[BUF_SIZE]; // 서버로부터 받은 메시지 길이
    int str_len;
    socklen_t adr_sz;
    struct sockaddr_in serv_addr, from_addr; // 주소 구조체

    if (argc != 3) { // 아규먼트가 3 개가 아니면, shell에서 포트 번호를 지정하지 않았으면
        printf("Usage : %s <IP> <port>\n", argv[0]); // 사용자에게 사용법 안내
        exit(1); // 비정상 종료 처리
    }
    sock = socket(PF_INET, SOCK_DGRAM, 0); // UDP 소켓 생성
    if (sock == -1) { // 생성 실패
        error_handling("socket() error"); // 에러 표시
    }
    memset(&serv_addr, 0, sizeof(serv_addr)); // 소켓에 할당할 주소 구조체 메모리 할당후, 0 으
                                                // 로 초기화
    serv_addr.sin_family=AF_INET; // 서버 주소 체계를 AF_INET 로 설정
    serv_addr.sin_addr.s_addr=inet_addr(argv[1]);
    // 서버의 IP 주소를 사용자가 입력한 첫 번째 인자로 설정
    serv_addr.sin_port=htons(atoi(argv[2]));
    // 서버의 포트 번호를 사용자가 입력한 두 번째 인자로 설정

    while (1) {
        fputs("insert message(q to quit) : ",stdout); // 안내 메시지를 출력
        fgets(message, sizeof(message), stdin); // 메시지를 입력받음

        if(!strcmp(message, "q\n") || !strcmp(message, "Q\n")) {
            // 입력된 메시지가 "q\n" 또는 "Q\n"인 경우
            break; // 종료
        }
        sendto(sock, message, strlen(message), 0, (struct sockaddr*)&serv_addr,
               sizeof(serv_addr)); // 소켓을 통해 서버로 메시지를 전송

        adr_sz = sizeof(from_addr); // 클라이언트의 주소 구조체의 크기를 계산
        str_len = recvfrom(sock, message, BUF_SIZE, 0, (struct sockaddr*)&from_addr,
                           &adr_sz); // 소켓을 통해 서버로 메시지를 수신
    }
}

```

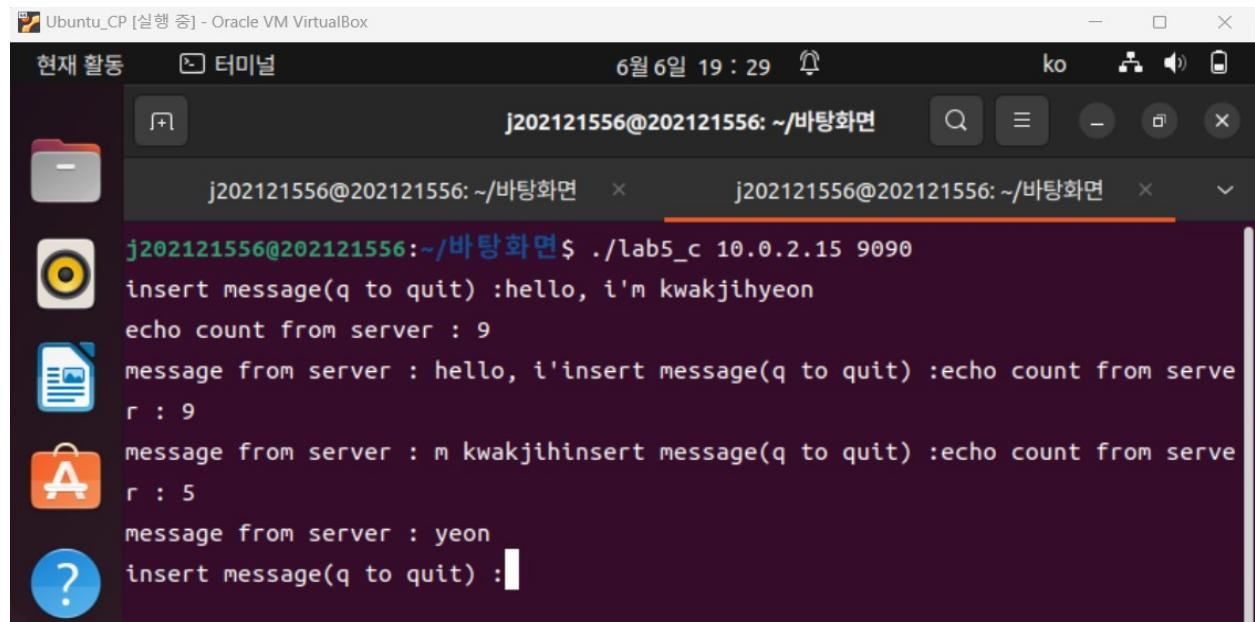
```

        printf("echo count from server : %d \n", str_len);
        // 서버로부터 수신한 메시지의 길이를 출력
        message[str_len] = 0; // 수신한 문자열 뒤에 NULL 문자 추가
        printf("message from server : %s", message); // 메시지 출력
    }
    close(sock); // 소켓 닫음
    return 0;
}

void error_handling(char *message) // 에러 처리 함수
{
    fputs(message, stderr); // 에러 메시지를 표준 에러 스트림에 출력
    fputc('\n', stderr); // 개행 문자를 표준 에러 스트림에 출력
    exit(1); // 프로그램 종료
}

```

5.2 출력 결과



The screenshot shows a terminal window titled "Ubuntu_C [실행 중] - Oracle VM VirtualBox". The terminal interface includes a dock with icons for file, terminal, and application windows. The main window displays a command-line session:

```

j202121556@202121556:~/바탕화면$ ./lab5_c 10.0.2.15 9090
insert message(q to quit) :hello, i'm kwakjihyeon
echo count from server : 9
message from server : hello, i'm kwakjihyeon
insert message(q to quit) :echo count from server : 9
message from server : m kwakjihyeon
insert message(q to quit) :yeon
insert message(q to quit) :

```

5.3 와이어샤크 확인 결과

