Network Studio 실습 레포트#2

학과 : 소프트웨어학과

학번 : 2009122172

이름 : 신성철

**2차 실습 문제**

◆ 특정한 문자열을 주고받는 서버-클라이언트 프로그램을 만드시오

◆ 서버 프로그램 요구 사항

⦁ TCP 소켓을 이용하여 클라이언트의 접속을 기다린다.

⦁ 클라이언트가 접속하면 클라이언트에게 문자열의 길이를 헤더에 포함하여 문자열을

전송한다.

⦁ 문자열을 전송한 뒤 클라이언트와의 접속을 끊고 다음 접속을 처리한다.

◆ 클라이언트 프로그램 요구 사항

⦁ TCP 소켓을 이용하여 서버에 접속한다.

⦁ 서버의 IP 주소는 프로그램을 실행할 때 arguments를 이용해 입력한다.

⦁ 서버에 접속하면 서버가 전송한 메시지를 읽는다.

⦁ 메시지의 헤더에 포함된 문자열의 길이를 이용해 메시지를 해독한다.

⦁ 문자열을 출력한 뒤 서버와의 접속을 끊는다.

**소스코드**

//header.cpp

#ifndef Net02\_header\_h

#define Net02\_header\_h

# include <stdio.h>

# include <stdlib.h>

# include <string.h>

# include <netinet/in.h>

# include <sys/socket.h>

# include <arpa/inet.h>

# include <unistd.h>

# define PORT 10003

# define BUFFER\_SIZE 100

ssize\_t readvn(int socketFD, char\* buffer, size\_t maxLength);

ssize\_t writevn(int socketFD, char\* buffer, size\_t length);

#endif

//client.cpp

# include "header.h"

int main(int argc, char\*\* argv) {

if (argc != 2) {

printf("Usage: %s IPv4-address\n", argv[0]);

return -1;

}

struct sockaddr\_in connectSocket;

//클라이언트용 소켓(connectSocket)을 set 하시오

connectSocket.sin\_addr.s\_addr = inet\_addr(argv[1]);

connectSocket.sin\_family = AF\_INET; // TCP 이용

connectSocket.sin\_port = htons(PORT); // 포트번호 지정

// 연결 소켓 생성

int connectFD = socket(AF\_INET, SOCK\_STREAM, 0);

// connect 요청

if (connect(connectFD, (struct sockaddr\*) &connectSocket, sizeof(connectSocket)) == -1) {

printf("Can not connect.\n");

return -1;

} else {

char buffer[BUFFER\_SIZE];

// 서버에서 보내중 값을 읽어들인다.

int n = (int)readvn(connectFD, buffer, (size\_t)BUFFER\_SIZE);

printf("%d bytes read\n", n);

buffer[n] = '\0';

fputs(buffer, stdout);

fflush(stdout);

}

close(connectFD);

return 0;

}

//server.cpp

#include "header.h"

# define LISTEN\_QUEUE\_SIZE 5

int main() {

struct sockaddr\_in listenSocket;

//서버용 리슨 소켓(listenSocket)을 set 한다.

listenSocket.sin\_addr.s\_addr = htonl(INADDR\_ANY);

listenSocket.sin\_family=AF\_INET; // TCP

listenSocket.sin\_port=htons(PORT); // 포트번호 설정

int listenFD = socket(AF\_INET, SOCK\_STREAM, 0); // 소켓 생성

// 리스 소켓을 포트번호와 바인딩 한다.

if (bind(listenFD, (struct sockaddr \*) &listenSocket, sizeof(listenSocket)) == -1) {

printf("Can not bind.\n");

return -1;

} // 버퍼를 생성한다.

if (listen(listenFD, LISTEN\_QUEUE\_SIZE) == -1) {

printf("Listen fail.\n");

return -1;

}

printf("Waiting for clients...\n");

while (1) {

struct sockaddr\_in connectSocket, peerSocket;

socklen\_t connectSocketLength = sizeof(connectSocket);

//클라이언트의 접속을 허가하여 소켓 ID(connectFD)를 set 하시오

int connectFD = accept(listenFD, (struct sockaddr \*)&connectSocket, &connectSocketLength); // 요청을 기다리며 sleep

getpeername(connectFD, (struct sockaddr\*)&peerSocket, &connectSocketLength); // peer 소켓의 주소를 클라이언트에게 보내줍니다.

char peerName[sizeof(peerSocket.sin\_addr) + 1] = {0};

sprintf(peerName, "%s", inet\_ntoa(peerSocket.sin\_addr));

printf("Client : %s\n", peerName);

char buffer[BUFFER\_SIZE] = {0};

sprintf(buffer, "Hello, %s\n", peerName);

//buffer를 버퍼 크기만큼 connectFD 소켓을 이용하여 보냄

writevn(connectFD, buffer, strlen(buffer));

close(connectFD);

}

close(listenFD);

return 0;

}

//readvn.cpp

# include "header.h"

# include <errno.h>

//errno를 위한 헤더를 추가하시오

ssize\_t readn(int socketFD, char\* buffer, size\_t length) {

size\_t nleft = length; // 받으려는 문자열의 길이

ssize\_t nread; // 읽은 문자의 수

char \*ptr = buffer;

while(nleft>0){ // 받으려는 문자가 남은 경우

if((nread=read(socketFD,ptr,nleft))<0){

if(errno == EINTR){ nread = 0; } //INTR

else{ return -1; } // 연결 강제 해제

} else if (nread == 0){ break; } //EOF

nleft -= nread; // 한 번에 못 읽을 경우

ptr += nread; // 다음 읽을 수를 지정

}

return (length-nleft);

}

ssize\_t readvn(int socketFD, char\* buffer, size\_t maxLength) {

int reclen; // 헤더에 저장된 문자열의 길이

int rc; // 읽은 문자열의 길이

// 헤더를 읽어 정수형의 메세지 길이를 받았는지 확인

rc = (int)readn(socketFD, (char \*)&reclen, sizeof(int));

if(rc != sizeof(int)) return rc<0 ? -1 : 0;

if(reclen> maxLength){ // 최대길이보다 받은 문자열이 긴 경우

while (reclen > 0) // 버려줍니다.

{

rc = (int)readn(socketFD, buffer, maxLength);

if (rc != maxLength) return rc < 0 ? -1 : 0;

reclen -= maxLength;

if (reclen < maxLength) maxLength = reclen;

}

errno=EMSGSIZE; // 메세지 사이즈 에러

return -1;

}

rc = (int)readn(socketFD, buffer, reclen);

if( rc != reclen ){ // 메세지가 기대한 만큼 읽어지지 않은 경우

return rc<0 ? -1 : 0;

}

return rc;

}

//writevn.cpp

# include "header.h"

# include <errno.h>

//errno를 위한 헤더를 추가하시오

//전부 다보냄 전송할 문자가 없을 때 까지

ssize\_t writen(int socketFD, char\* buffer, size\_t length) {

size\_t nleft = length; // 입력한 문자열의 길이

ssize\_t nwritten; // 한 번의 write로 전송한 문자 수

char\* ptr = buffer;

while(nleft>0){ // 전송할 문자가 남아있다면

if ((nwritten = write (socketFD, ptr, nleft))<= 0) {

if (errno == EINTR) // 인터럽트가 걸린 경우

nwritten = 0; // call write again

else return (-1); // 상대방이 강제로 연결을 끊은 경우

}

nleft -= nwritten; // 한 번의 write로 모두 못 보낼 수

ptr += nwritten; // 있으므로 다음 write를 준비합니다.

}

return (length); // return >= 0

}

ssize\_t writevn(int socketFD, char\* buffer, size\_t length) {

int size = length;

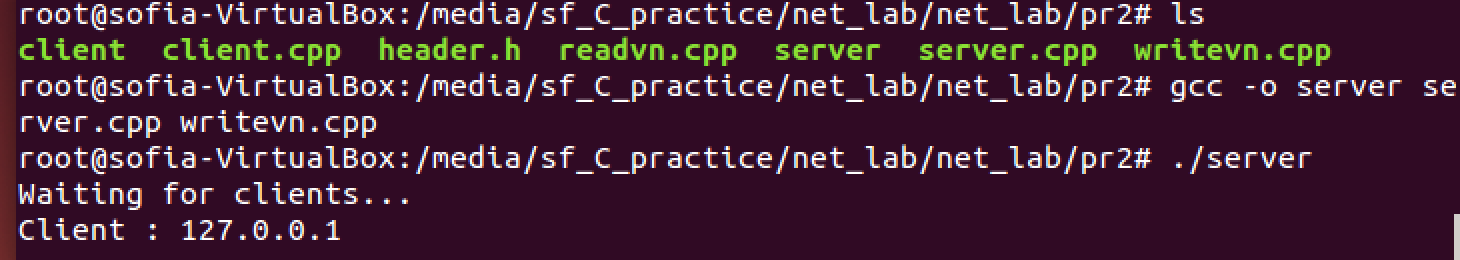
writen(socketFD, (char\*)&size, sizeof(size));// header 부분

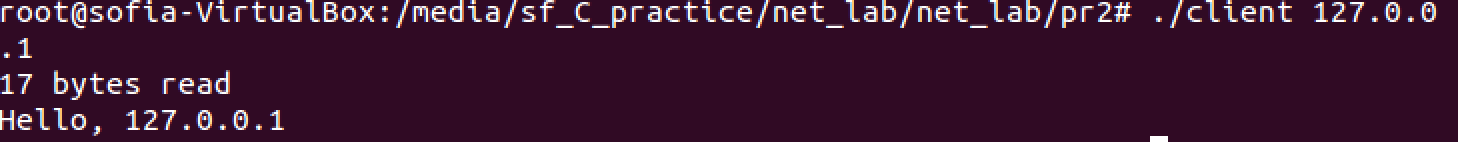
writen(socketFD, buffer, length); // variable(진짜 데이터) 부분

return (length); // return >= 0

}

**테스트 결과**



서버를 작동시키고 클라이언트로 서버에 접근하면 서버 사이드에서 Hello를 클라이언트에게 보내고 성공적으로 받은 것을 알 수 있다.

**느낀점**

ssize\_t : signed int ,size\_t : unsigned int 는 고전적인(primitive) 자료형이고 일반적으로 <sys/types.h> 헤더에 선언되어 있는데, 사실 C의 typedef 선언을 통해서 정의되어 있다는 것을 알았다. 즉 새롭게 등장한 것이 아니라, 잘 알고 있는 int, float, long과 같은 기본 자료형을 가지고, 이름만 바꾸어 정의해 놓은 것이다. 이러한 자료형들은 꾀 오래 전부터 존재하고 있었다. 그렇다면 도대체 왜 이러한 것을 정의해 놓고 사용하는 지 궁금해서 찾아보았다.

이 둘은 아키텍쳐 portability를 위함이다. 보통 x86의 int는 4바이트의 크기를 갖고 있다. 그래서 이러한 사실을 기반으로 프로그래밍을 할 때, 4바이트의 자료형에 대해서 int를 선언한다. 하지만 만약 int가 4바이트의 크기를 갖지 않는 아키텍처에 위의 프로그램이 포팅된다면 제대로 동작하지 않을 것이다.

따라서, 4바이트의 크기를 갖는 자료형에 대해, 아키텍처와 상관없이 사용하는 데이터 타입이 size\_t와 ssize\_t이다. 나중에 포팅이 되었더라도 typdef를 통해서 size\_t의 값을 제 설정해주면 프로그램이 잘 동작할 것이다.