s01.
네트워크 프로그래밍과 소켓의 이해

DCCS320(00) 네트워크프로그래밍및실습
bluepig5@korea.ac.kr

네트워크 프로그래밍과 소켓에 대한 이해

- 네트워크 프로그래밍이란?
  - 소켓이라는 것을 기반으로 프로그래밍을 하기 때문에 소켓 프로그래밍이라고도 함
  - 네트워크로 연결된 둘 이상의 컴퓨터 사이에서의 데이터 송수신 프로그램의 작성을 의미함

- 소켓이란?
  - 네트워크(인터넷)의 연결 도구
  - 운영체제에 의해 제공이 되는 소프트웨어적인 장치

➔ 소켓은 프로그래머에게 데이터 송수신에 대한 물리적, 소프트웨어적 세세한 내용을 신경 쓰지 않게 한다.

소켓 사용 절차 - 수신자

- 연결요청을 허용하는 소켓의 생성과정
  - 1단계. 소켓의 생성 socket 함수호출
  - 2단계. IP와 PORT번호의 할당 bind 함수호출
  - 3단계. 연결요청 가능상태로 변경 listen 함수호출
  - 4단계. 연결요청에 대한 수락 accept 함수호출

소켓의 생성

- 전화 받는 소켓의 생성
  - TCP 소켓은 전화기에 비유될 수 있다.
  - 소켓은 socket 함수의 호출을 통해서 생성한다.
  - 단, 전화를 거는 용도의 소켓과 전화를 수신하는 용도의 소켓 생성 방법에는 차이가 있다.

소켓의 생성

```
#include <sys/socket.h>
int socket(int domain, int type, int protocol);
```

➜ 성공 시 파일 디스크립터, 실패 시 -1 반환

소켓의 생성은 전화기의 장만에 비유할 수 있다.

IP와 PORT번호의 할당

- 소켓의 주소 할당 및 연결
  - 전화기에 전화번호가 부여되듯이 소켓에도 주소정보가 할당된다.
  - 소켓의 주소정보는 IP와 PORT번호로 구성이 된다.

주소의 할당

```
#include <sys/socket.h>
int bind(int sockfd, struct sockaddr *myaddr, socklen_t addrlen);
```

➜ 성공 시 0, 실패 시 -1 반환

연결요청 가능상태로 변경

- 연결요청이 가능한 상태의 소켓은 걸려오는 전화를 받을 수 있는 상태에 비유할 수 있다.
- 전화를 거는 용도의 소켓은 연결요청이 가능한 상태의 소켓이 될 필요가 없다. 이는 걸려오는 전화를 받는 용도의 소켓에서 필요한 상태이다.

연결요청 가능한 상태로 변경

```
#include <sys/socket.h>
int listen(int sockfd, int backlog);
```

➡ 성공 시 0, 실패 시 -1 반환

소켓에 할당된 IP와 PORT번호로 연결요청이 가능한 상태가 된다.

연결요청의 수락

- 걸려오는 전화에 대해서 수락의 의미로 수화기를 드는 것에 비유할 수 있다.
- 연결요청이 수락되어야 데이터의 송수신이 가능하다.
- 수락된 이후에 데이터의 송수신은 양방향으로 가능하다.

연결요청 가능한 상태로 변경

```
#include <sys/socket.h>
int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);
    ➜ 성공 시 파일 디스크립터, 실패 시 -1 반환
```

accept 함수호출 이후에는 데이터의 송수신이 가능하다.

단, 연결요청이 있을 때에만 accept 함수가 반환을 한다.

소켓 사용 절차 - 송신자

- 연결을 요청하는 소켓의 생성과정
  - 1단계. 소켓의 생성 socket 함수호출
  - 2단계. IP와 PORT번호의 설정
  - 3단계. 연결요청 connect 함수호출

연결을 요청하는 소켓의 구현

- 전화를 거는 상황에 비유할 수 있다.
- 리스닝 소켓과 달리 구현의 과정이 매우 간단하다.
- '소켓의 생성'과 '연결의 요청'으로 구분된다.

연결의 요청

```
#include <sys/socket.h>
int connect(int sockfd, struct sockaddr *serv_addr, socklen_t addrlen);
```

➡ 성공 시 0, 실패 시 -1 반환

실습1

- 예제 hello_server.c를 통해서 함수의 호출과정 확인하기
  - 연결요청을 허용하는 프로그램을 가리켜 일반적으로 서버(Server)라 한다.
  - 서버는 연결을 요청하는 클라이언트보다 먼저 실행되어야 한다.
  - 클라이언트보다 복잡한 실행의 과정을 거친다.

- 예제 hello_client.c를 통해서 함수의 호출과정 확인하기
  - 함수의 호출과 데이터가 실제 송수신 됨을 확인하자.
  - 소스코드의 이해는 점진적으로...

리눅스 기반에서의 실행방법

- 컴파일 및 실행방법

컴파일 방법

```
gcc hello_server.c -o hserver
```

➡ hello_server.c 파일을 컴파일해서 hserver라는 이름의 실행파일을 만드는 문장이다.

실행방법

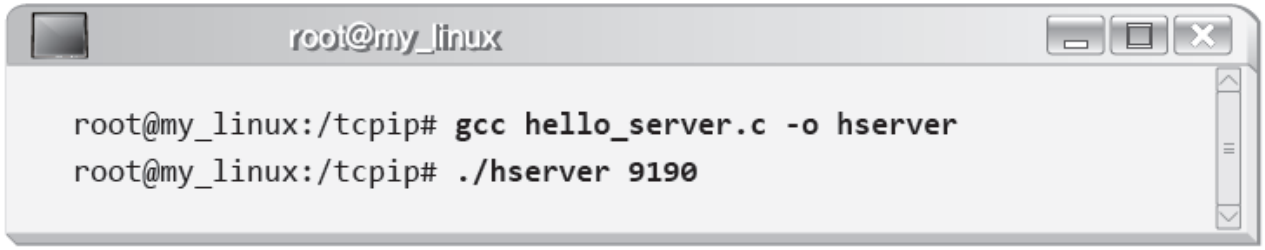
```
./hserver
```

➡ 현재 디렉터리에 있는 hserver라는 이름의 파일을 실행시키라는 의미이다.

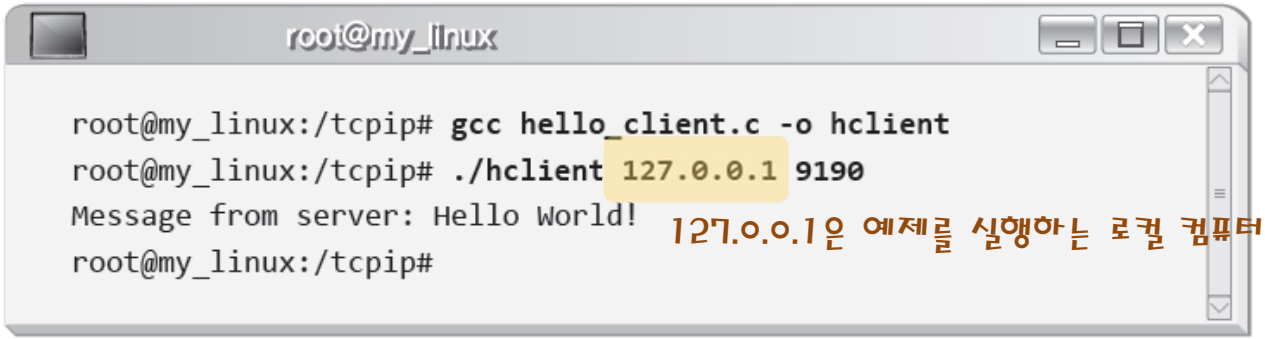
리눅스 기반에서의 실행결과

- 예제의 실행결과

❖ 실행결과: hello_server.c



❖ 실행결과: hello_client.c



127.0.0.1은 예제를 실행하는 로컬 컴퓨터를 의미함

s02.
리눅스 기반
파일 조작하기

저수준 파일 입출력과 파일 디스크립터

- 저수준 파일 입출력
  - ANSI의 표준함수가 아닌, 운영체제가 제공하는 함수 기반의 파일 입출력
  - 표준이 아니기 때문에 운영체제에 대한 호환성이 없다.
  - 리눅스는 소켓도 파일로 간주하기 때문에 저수준 파일 입출력 함수를 기반으로 소켓 기반의 데이터 송수신이 가능하다.

파일 디스크립터	대 상
0	표준입력: Standard Input
1	표준출력: Standard Output
2	표준에러: Standard Error

- 파일 디스크립터
  - 운영체제가 만든 파일(그리고 소켓)을 구분하기 위한 일종의 숫자
  - 저수준 파일 입출력 함수는 입출력을 목적으로 파일 디스크립터를 요구한다.
  - 저수준 파일 입출력 함수에게 소켓의 파일 디스크립터를 전달하면, 소켓을 대상으로 입출력을 진행한다.

파일 열기와 닫기

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

int open(const char *path, int flag);
```

➡ 성공 시 파일 디스크립터, 실패 시 -1 반환

- path — 파일 이름을 나타내는 문자열의 주소 값 전달.
- flag — 파일의 오픈 모드 정보 전달.

오픈 모드	의 미
O_CREAT	필요하면 파일을 생성
O_TRUNC	기존 데이터 전부 삭제
O_APPEND	기존 데이터 보존하고, 뒤에 이어서 저장
O_RDONLY	읽기 전용으로 파일 오픈
O_WRONLY	쓰기 전용으로 파일 오픈
O_RDWR	읽기, 쓰기 겸용으로 파일 오픈

```
#include <unistd.h>

int close(int fd);
```

➡ 성공 시 0, 실패 시 -1 반환

- fd — 닫고자 하는 파일 또는 소켓의 파일 디스크립터 전달.

open 함수 호출 시 반환된 파일 디스크립터를 이용해서 파일 입출력을 진행하게 된다.

파일에 데이터 쓰기

```
#include <unistd.h>

ssize_t write(int fd, const void * buf, size_t nbytes);
```

➡ 성공 시 전달한 바이트 수, 실패 시 -1 반환

- fd — 데이터 전송대상을 나타내는 파일 디스크립터 전달.
- buf — 전송할 데이터가 저장된 버퍼의 주소 값 전달.
- nbytes — 전송할 데이터의 바이트 수 전달.

```
int main(void)
{
    int fd;
    char buf[]="Let's go!\n";
    fd=open("data.txt", O_CREAT|O_WRONLY|O_TRUNC);
    if(fd==-1)
        error_handling("open() error!");
    printf("file descriptor: %d \n", fd);

    if(write(fd, buf, sizeof(buf))==-1)
        error_handling("write() error!");
    close(fd);
    return 0;
}
```

실행 결과

```
root@my_linux:/tcpip# gcc low_open.c -o lopen
root@my_linux:/tcpip# ./lopen
file descriptor: 3
root@my_linux:/tcpip# cat data.txt
Let's go!
root@my_linux:/tcpip#
```

파일에 저장된 데이터 읽기

```
#include <unistd.h>

ssize_t read(int fd, void *buf, size_t nbytes);
```

➜ 성공 시 수신한 바이트 수(단 파일의 끝을 만나면 0), 실패 시 -1 반환

- fd — 데이터 수신대상을 나타내는 파일 디스크립터 전달.
- buf — 수신한 데이터를 저장할 버퍼의 주소 값 전달.
- nbytes — 수신할 최대 바이트 수 전달.

```
int main(void)
{
    int fd;
    char buf[BUF_SIZE];

    fd=open("data.txt", O_RDONLY);
    if( fd==-1)
        error_handling("open() error!");
    printf("file descriptor: %d \n" , fd);

    if(read(fd, buf, sizeof(buf))==-1)
        error_handling("read() error!");
    printf("file data: %s", buf);
    close(fd);
    return 0;
}
```

실행결과

```
root@my_linux:/tcpip# gcc low_read.c -o lread
root@my_linux:/tcpip# ./lread
file descriptor: 3
file data: Let's go!
root@my_linux:/tcpip#
```

파일 디스크립터와 소켓

```c
int main(void)
{
    int fd1, fd2, fd3;
    fd1=socket(PF_INET, SOCK_STREAM, 0);
    fd2=open("test.dat", O_CREAT|O_WRONLY|O_TRUNC);
    fd3=socket(PF_INET, SOCK_DGRAM, 0);
    printf("file descriptor 1: %d\n", fd1);
    printf("file descriptor 2: %d\n", fd2);
    printf("file descriptor 3: %d\n", fd3);
    close(fd1); close(fd2); close(fd3);
    return 0;
}
```

실행결과

```
root@my_linux:/tcpip# gcc fd_seri.c -o fds
root@my_linux:/tcpip# ./fds
file descriptor 1: 3
file descriptor 2: 4
file descriptor 3: 5
root@my_linux:/tcpip#
```

실행결과를 통해서 소켓과 파일에 일련의 파일 디스크립터 정수 값이 할당됨을 알 수 있다.

그리고 이를 통해서 리눅스는 파일과 소켓을 동일하게 간주함을 확인할 수 있다.

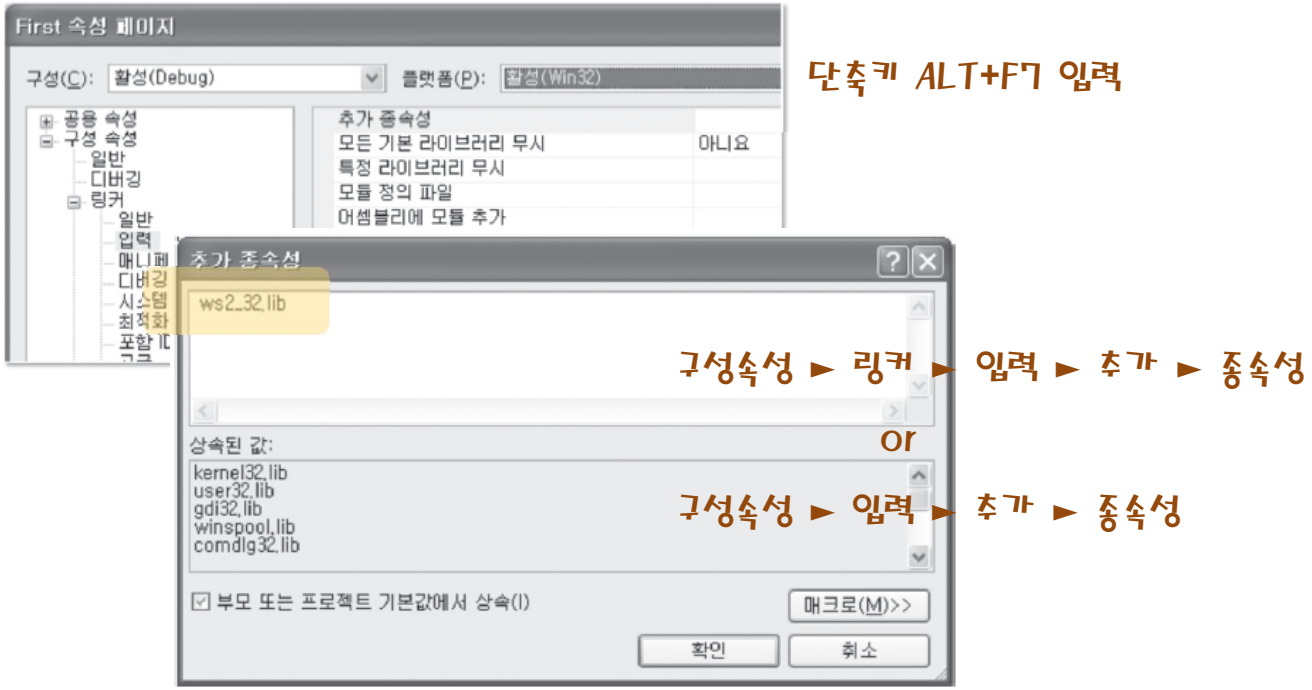
실습2

- 예제 low_open.c, low_read.c를 통해서 저수준 입출력함수의 사용법 익히기
  - 파일 열기 ➔ open
  - 파일에 쓰기 ➔ write
  - 파일의 내용 읽기 ➔ read

s03.
윈도우 기반으로 구현하기

윈도우 소켓을 위한 헤더와 라이브러리의 설정

- 윈속 기반의 프로그램 구현을 위한 두 가지 설정
  - 헤더파일 winsock2.h의 포함
  - ws2_32.lib 라이브러리의 링크



단축키 ALT+F7 입력

구성속성 ► 링커 ► 입력 ► 추가 ► 종속성

or

구성속성 ► 입력 ► 추가 ► 종속성

윈속의 초기화

윈속 초기화 함수

```
#include <winsock2.h>

int WSAStartup(WORD wVersionRequested, LPWSADATA lpWSAData);
```

➜ 성공 시 0, 실패 시 0이 아닌 에러코드 값 반환

- wVersionRequested — 프로그래머가 사용할 윈속의 버전정보 전달.
- lpWSAData — WSADATA라는 구조체 변수의 주소 값 전달.

코드상에서의 초기화 방법

```
int main(int argc, char* argv[])
{
    WSADATA wsaData;
    . . . .
    if(WSAStartup(MAKEWORD(2, 2), &wsaData) != 0)
        ErrorHandling("WSAStartup() error!");
    . . . .
    return 0;
}
```

윈속의 초기화란, 윈속 함수호출을 위한 라이브러리의 메모리 LOAD를 의미한다.

윈속 라이브러리의 해제

다음 함수가 호출되면 윈속 관련 함수의 호출이 불가능해 지므로, 프로그램이 종료되지 직전에 호출하는 것이 일반적이다!

윈속 라이브러리를 해제시키는 함수

```
#include <winsock2.h>

int WSACleanup(void);
```

➡ 성공 시 0, 실패 시 SOCKET_ERROR 반환

s04.
윈도우 기반의
함수와 예제

윈도우 기반 소켓관련 함수들

리눅스의 socket 함수에 대응

```
#include <winsock2.h>

SOCKET socket(int af, int type, int protocol);
```

➡ 성공 시 소켓 핸들, 실패 시 INVALID_SOCKET 반환

리눅스의 bind 함수에 대응

```
#include <winsock2.h>

int bind(SOCKET s, const struct sockaddr * name, int namelen);
```

➡ 성공 시 소켓 핸들, 실패 시 INVALID_SOCKET 반환

리눅스에서의 파일 디스크립터에 해당하는 것을 윈도우에서는 핸들(HANDLE)이라 한다!

윈도우 기반 소켓관련 함수들

리눅스의 listen 함수에 대응

```
#include <winsock2.h>

int listen(SOCKET s, int backlog);
```

➜ 성공 시 0, 실패 시 SOCKET_ERROR 반환

리눅스의 accept 함수에 대응

```
#include <winsock2.h>

SOCKET accept(SOCKET s, struct sockaddr * addr, int * addrlen);
```

➜ 성공 시 소켓 핸들, 실패 시 INVALID_SOCKET 반환

윈도우 기반 소켓관련 함수들

리눅스의 connect 함수에 대응

```
#include <winsock2.h>

int connect(SOCKET s, const struct sockaddr * name, int namelen);
```

➜ 성공 시 0, 실패 시 SOCKET_ERROR 반환

리눅스의 close 함수에 대응

```
#include <winsock2.h>

int closesocket(SOCKET s);
```

➜ 성공 시 0, 실패 시 SOCKET_ERROR 반환

윈도우 기반 입출력 함수

- 윈도우에서는 별도의 입출력 함수를 사용
  - 리눅스와 달리 파일과 소켓을 별도의 리소스로 구분한다.
  - 때문에 파일 입출력 함수와 소켓 기반의 입출력 함수에 차이가 있다.

```
#include <winsock2.h>

int send(SOCKET s, const char * buf, int len, int flags);
```

➜ 성공 시 전송된 바이트 수, 실패 시 SOCKET_ERROR 반환

- s — 데이터 전송 대상과의 연결을 의미하는 소켓의 핸들 값 전달.
- buf — 전송할 데이터를 저장하고 있는 버퍼의 주소 값 전달.
- len — 전송할 바이트 수 전달.
- flags — 데이터 전송 시 적용할 다양한 옵션 정보 전달.

```
#include <winsock2.h>

int recv(SOCKET s, const char * buf, int len, int flags);
```

➜ 성공 시 수신한 바이트 수(단 EOF 전송 시 0), 실패 시 SOCKET_ERROR 반환

- s — 데이터 수신 대상과의 연결을 의미하는 소켓의 핸들 값 전달.
- buf — 수신된 데이터를 저장할 버퍼의 주소 값 전달.
- len — 수신할 수 있는 최대 바이트 수 전달.
- flags — 데이터 수신 시 적용할 다양한 옵션 정보 전달.

실습3

- 윈도우 기반 서버, 클라이언트 예제 실행하기
  - 예제 hello_server_win.c, hello_client_win.c의 실행

❖ 실행결과: hello_server_win.c

❖ 실행결과: hello_client_win.c

소스코드를 통해서 다음 두 가지 사실을 관찰하자.

1. 소켓의 생성과정에 따른 함수의 호출
2. 리눅스 기반에서 호출했던 소켓 기반 입출력 함수에 어떠한 차이가 있는가?