TCP/IP 소켓 프로그래밍

Chapter 01. 네트워크 프로그래밍과 소켓의 이해

* 1. 네트워크 프로그래밍과 소켓의 이해

네트워크 프로그래밍이란 네트워크로 연결되어 있는 서로 다른 두 컴퓨터가 데이터를 주고 받을 수 있도록 하는 것이다. 기본적으로 컴퓨터 간의 네트워크로 연결해야 하는데 각 컴퓨터간 컴퓨터단과 네트워크를 연결해주어야 두 컴퓨터는 데이터 송수신이 가능해진다. 이때 운영체제에서 제공하는 ‘Socket’을 통해 컴퓨터와 네트워크를 연결할 수 있다. 한마디로 **네트워크 망의 연결에 사용하는 도구인 것**이다.

네트워크 프로그래밍에서 연결요청을 허용하는 소켓의 생성과정을 다음과 같이 정리할 수 있다.

**1단계**. 소켓 생성 -> socket 함수호출

**2단계.** IP 주소와 PORT번호 할당 -> bind 함수호출

**3단계.** 연결요청 가능상태로 변경 -> listen 함수호출

**4단계**. 연결요청에 대한 수락 -> accept 함수호출

연결요청을 수락하는 기능의 프로그램을 ‘서버’라고 한다.

기본적으로, **리눅스 운영체제**에서 **서버**에서는 소켓을 생성하는 **socket** 함수, **bind**함수, **listen**함수, **accept**함수 순으로 호출을 진행하고, **클라이언트**는 소켓을 생성하고 **connect**함수를 호출하여 서버에 연결요청을 진행한다.

소켓을 생성하는 순간에는 생성하는 소켓이 서버소켓인지 클라이언트 소켓인지 구분되지 않는다. Socket 함수를 통해 생성 후 bind함수 listen함수가 이어지면 서버소켓이 되는 것이고, connect함수가 이어지면, 클라이언트 소켓이 되는 것이다.

127.0.0.1은 보통 프로그램을 실행한 로컬 컴퓨터의 IP주소를 뜻한다.

**01-2 리눅스 기반 파일 조작하기**

* 저 수준 파일 입출력 과 파일 디스크립터

**저 수준 파일 입출력**이란? 운영체제가 독립적으로 제공하는 함수라는 의미를 가진다.

**파일 디스크립터란**? 운영체제가 파일 또는 소켓의 지칭을 편하게 하기 위해 부여한 숫자라고 생각하면 된다.

**파일 열기**

**Int open**(const char \***path**, int **flag**);

* 성공 시 **파일 디스크립터 반환**, 실패 시 **-1** 반환
* **path** : 파일 **이름**을 나타내는 문자열의 **주소 값** 전달
* **flag** : 파일의 **오픈 모드 정보** 전달 (p28 표 01-2 참고)

**파일 닫기**

int **close**(int fd);

* 성공 시 **0**, 실패 시 **-1** 반환
* **fd** : 닫고자 하는 파일 또는 소켓의 **파일 디스크립터** 전달

**파일에 데이터 쓰기**

**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

**파일에 저장된 데이터 읽기**

**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

리눅스에서는 파일과 소켓을 구분하지 않고 둘다 파일 디스크립터를 반환한다.

* 1. **윈도우 기반**

기본적으로 리눅스 기반의 소켓 관련 함수들과 비슷하지만 매개변수나 반환형이 조금 다르다. 특히나 close함수 같은 경우 윈도우는 closesocket함수명으로 socket별로 지정해서 close한다.

* 윈도우에서의 파일 핸들과 소켓 핸들

리눅스는 소켓도 파일로 취급하도록 내부적으로 설계되어 있다. 그래서 리눅스는 파일을 생성하건 소켓을 생성하건 모두 파일 디스크립터가 반환된다.

마찬가지로 윈도우에서도 시스템 함수의 호출을 통해 파일을 생성할 때 ‘핸들’이라는 것을 반환한다. 리눅스의 파일 디스크립터와 윈도우의 핸들과 비교될 수 있다. 하지만 윈도우에서는 소켓과 파일의 핸들을 소켓핸들과 파일핸들로 구분하고 있다.

SOCKET 자료형 : 정수로 표현되는 소켓의 핸들 값 저장을 위해서 typedef 선언으로 정의된 새로운 자료형

* 윈도우 기반 입출력 함수

리눅스에서는 파일 입출력 함수인 write 나 read함수를 이용해서 데이터를 송수신 할 수 있지만 위도우에서는 파일 입출력과 소켓 입출력 함수가 구분된다.

send(SOCKET s , const char \* buf, int len, int flags) 함수

* 성공 시 전송된 바이트 수 반환 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 옵션정보 전달

recv(SOCKET s , const char \* buf, int len, int flags) 함수

* 성공 시 전송된 바이트 수 반환 실패 시 SOCKET\_ERROR 반환
* s : 데이터 수신 대상과의 연결을 의미하는 소켓의 핸들 값 전달
* buf : 수신할 데이터를 저장할 버퍼의 주소 값 전달
* len : 수신할 수 있는 최대 바이트 수 전달
* flags : 데이터 수신 시 적용할 다양한 옵션정보 전달

!주의점 : 위의 send 함수가 윈도우에만 존재하는 함수가 아니다. 리눅스에도 존재하는 함수이지만 리눅스에서는 write read 함수로도 대체가 가능하다. 하지만 윈도우에서 write read 함수는 사용이 불가능하다.

Chapter 02. 소켓의 타입과 프로토콜의 설정

02-1 소켓의 프로토콜과 그에 따른 데이터 전송 특성

프로토콜이란? 대화에 필요한 동신규약을 의미하며, 컴퓨터 관점에서는 컴퓨터 상호간의 대화에 필요한 통신규약이라고 할 수 있다.

* 소켓의 생성

int socket(int domain, int type, int protocol);

* 성공 시 파일 디스크립터, 실패 시 -1 반환
* domain : 소켓이 사용할 \*프로토콜 체계(Protocol Family) 정보 전달.
* type : 소켓의 데이터 전송방식에 대한 정보 전달
* protocol : 두 컴퓨터간 통신에 사용되는 프로토콜 정보 전달
* 프로토콜 체계

소켓이 통신에 사용하는 프로토콜도 부류가 나뉘는데, 생성되는 소켓이 사용할 프로토콜의 부류정보를 socket함수의 첫번째 인자로 전달해야 한다. 프로토콜 체계(p51 표 02-1참조)

* 소켓의 타입

소켓의 전송방식을 의미하는 것으로, socket함수의 두번째 인자로 전달한다 대표적으로 두가지의 전송방식이 존재한다.

**타입 1**. **연결지향형 소켓(SOCK\_STREAM)** 의 특징

* 중간에 데이터가 **소멸되지 않**고 목적지로 전송된다.
* **전송 순서대로** 데이터가 수신된다.
* 전송되는 **데이터의 경계가 존재하지 않는다**.
* 데이터가 수신되었다고 **바로 read함수를 호출해야 하는 것은 아니다**.
* **read함수**의 호출횟수와 **write함수**의 **호출횟수는 상관이 없다**.
* 소켓 대 **소켓의 연결은 항상 1대 1**이어야 한다(**하나의 연결지향형 소켓은 하나의 다른 연결 지향형 소켓과만 연결이 가능하다**.)
* **신뢰성** 있는 순차적인 **바이트 기반**의 연결지향 데이터 전송 방식의 소켓

**타입 2**. **비 연결지향형 소켓(SOCK\_DGRAM)** 의 특징

* 전송된 순서에 상관없이 가장 **빠른 전송**을 지향
* 전송된 데이터는 **손실의 우려**가 있으며, **파손의 우려**가 있다.
* 전송되는 **데이터의 경계가 존재**한다.
* 한번에 할 수 있는 **데이터의 크기가 제한**된다.
* 데이터를 **전송할 때 함수가 두 번** 호출되면 **수신할 때도 함수가 두 번** 호출된다.
* **신뢰성**과 순차적 데이터 전송을 **보장하지 않**는, **고속의 데이터 전송을 목적**으로 하는 소켓
* protocol 마지막 인자
* 하나의 프로토콜 체계 한에 데이터의 전송방식이 동일한 프로토콜이 둘 이상 존재하는 경우
* 소켓의 데이터 전송방식은 같지만 그 안에서 또 프로토콜이 나뉘는 상황이 존재한다.
* 이때 3번째 protocol 인자를 통해 구체화 가능
* ex, **IPv4인터넷 프로토콜 체계에서 동작하는 연결지향형,비 연결지향형 데이터 전송 소켓으로 나뉠 때 각각 IPPROTO\_TCP, IPPROTO\_UDP를 3번째 인자로 넘겨주어 구체화한다.**
* 연결지향형 소켓 **TCP 소켓의** 예

tcp소켓의 경우 write함수와 read 함수의 **호출 횟수를 다르게 해도 동작**한다.

02-2 윈도우 기반에서의 이해

* 윈도우 운영체제의 socket함수

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

* 성공 시 소켓 핸들, 실패 시 INVALID\_SOCKET 반환
* 반환형 SOCKET은 정수로 표현되는 소켓의 핸들 값을 저장하기 위해 정의된 자료형의 이름이다 사실 int이지만 SOCKET 자료형으로 선언한 변수에 저장해라.
* INVALID\_SOCKET 도 오류발생을 알리는 상수이다. 실제론 -1이라는 값이다.
* 윈도우 기반 TCP 소켓의 예

리눅스 기반 TCP 소켓의 read 함수가 recv함수로 대체 됨.