Chapter 4
UDP 소켓 사용법

개요

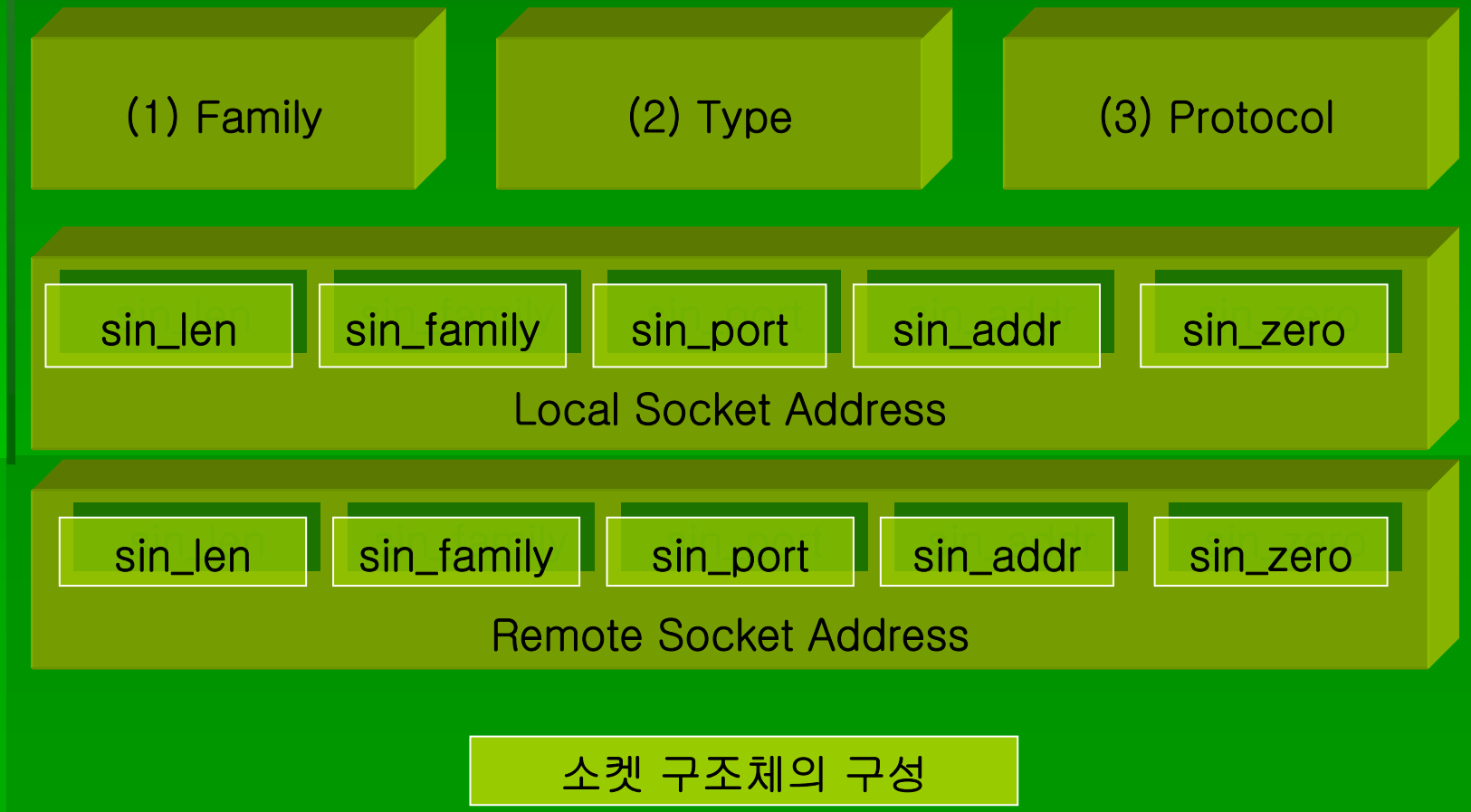
- 소켓이란(Unix 소켓)
- 소켓의 구조
- 소켓의 유형
- UDP에 대한 이해
- 교재 Chapter 4.1 – UDP클라이언트
- 교재 Chapter 4.2 – UDP서버
- 교재 Chapter 4.3 – UDP소켓을 이용한 송신 및 수신

소켓이란?

- Socket은 통신을 위한 끝점(endpoint)을 생성하여 파일에 대한 open과 유사한 방식으로 기술자(descriptor)을 반환한다. 소켓 매개변수는 인터넷 도메인(TCP/IP), NovellI의 IPX 또는 listen과 accept 또는 select를 호출하여 클라이언트는 일반적으로 bind와 connect를 호출한다. “유닉스 도메인 소켓”은 실제 네트워크 프로토콜을 나타내지는 않으며, 다만 동일 머신상의 소켓에 접속할 수 있다. 표준 유닉스에서는 1024 미만의 .TCP와 UDP 지역포트(port)넘버로의 바인딩(binding)은 루트 권한을 가져야 하며 모든 프로세스는 1024 또는 그 이상의 바인딩되어 있지 않은 포트 넘버에 바인드할 수 있는 것이 보통이다.

•소켓의 구조

소켓이라 함은 지역적으로 한 컴퓨터에서 네트워크를 통하여 다른 컴퓨터로의 연결을 이루게 하여 주는 전달자의 역할을 담당한다. 이런 전달자의 역할을 수행하기 위해서 소켓은 켲 가지 데이터 필드를 포함하여야 하며, 이는 소켓이 이루고 있는 기본 구성 요소가 된다.



소켓 구조체의 구성

(1) Family

존재하는 프로토콜 그룹 중의 하나를 선택하여 어떤 유형의 프로토콜에 따라 소켓이 진행될 것인지를 정해줄 수 있는 필드이다.

Family Protocol	프로토콜 상수 설명
AF_UNIX	유닉스 기본파일 체계 소켓
AF_INET	IPV4
AF_INET6	IPV6
AF_NS	Xerox
AF_ISO	ISO
AF_IPX	Novell IPX
AF_APPLETALK	Appletalk DOS

Family 프로토콜 상수

(2) Type

소켓은 연결지향형(SOCK_STREAM), 비연결형(SOCK_DGRAM), 저수준 프로토콜 제어(SOCK_RAW)의 세 가지 연결에 대한 Type을 설정할 수 있다. 각각의 타입에 따라 연결되는 소켓의 특징을 결정한다.

Type	각 Type 상수게 대한 설명
SOCK_STREAM	TCP (스트림 소켓)
SOCK_DGRAM	UDP (데이터그램 소켓)
SOCK_RAW	RAW 소켓

소켓 타입 상수

(3) Protocol

기반 전송 기법이 하나 이상의 프로토콜에서 요청된 소켓 형태를 제공하는 경우에는 소켓을 위해 특정 프로토콜을 선택할 수 있다. 일반적으로 Raw 소켓을 제외하고는 “0”으로 설정된다.

소켓의 유형


애플리케이션
Stream 소켓 인터페이스
Datagram 소켓 인터페이스
Raw 소켓 인터페이스
TCP
UDP
IP
물리 계층과 데이터 링크 계층

Stream 소켓

Stream 소켓은 TCP를 사용한 연결형 프로토콜이다. TCP는 인터넷을 통해 다른 애프리케이션과의 연결(connect)을 위해 한쌍의 Stream 소켓을 사용한다. 따라서 데이터의 전송은 원격지 소켓과의 연결 후 이루어 진다. 우리의 일상생활에서는 전화의 개념과 비슷하다.

Datagram 소켓

Datagram 소켓은 UDP를 사용한 비연결형 프로토콜이다. UDP는 인터넷을 통해 다른 애플리케이션에게 메시지를 보내기 위해 한 쌍의 Datagram 소켓을 사용한다. TCP와는 달리 연결 과정이 없기 때문에 보낼 때마다 도착지의 주소가 필요하다. 또한 전송에 대한 신뢰성이 없기 때문에 신뢰성을 유지하기 위해서는 타임아웃을 이용한 적절한 응답확인 절차가 필료하다. Datagram에 대한 쉬운 이해는 우리 생활에서 편지와 비슷하다.

Raw 소켓_byGIN새로운소켓을만들때사용

Raw 소켓은 ICMP(Internet Control Message Protocol) 또는 OSPF (Open Shortest Path First : 일종의 라우팅 프로토콜)같은 저수준의 프로토콜을 액세스할 수 있도록 해준다. TCP와 UDP가 아닌 IP계층을 직접 이용한다. Raw 소켓은 루트(root)만이 쓸 수 있고, 새로운 프로토콜을 만들 때 사용하는 소켓이다.

UDP(User Datagram Protocol)

- 다른 계층의 주소 체계(포트)를 IP계층 위에 추가하고 전송중에 발생할 수 있는 데이터 훼손을 감지하고 훼손된 데이터그램을 제거한다.
- 생성되자마자 한 UDP소켓은 메시지들을 어떤 임의의 주소로/주소로부터, 또 이어서 여러 다른 주소들로 /주소들로 부터 송/수신하는데 사용할 수 있다.
- UDP가 제공하는 종단 간 전송 서비스가 최선형(best effort)이라는 것이다. 즉 UDP소켓에 의해 보내진 한 메시지가 목적지에 도착하는것에 대한 보장이 없다. 이 사실은 UDP소켓을 사용하는 프로그램은 메시지의 유실이나 순서바뀜에 대한 대비책이 있어야 한다는 것을 의미한다.

UDPEchoClient.c

- 소켓생성및 설정

```
If ((sock = socket (PF_INET, SOCK_DGRAM,
    IPPROTO_UDP)) < 0)
    DieWithError("socket() failed");
```

- 하나의 에코 데이터그램을 보냄

```
memset(&echoServAddr, 0, sizeof(echoServAddr));
echoServAddr.sin_family = AF_INET;
echoServAddr.sin_addr.s_addr = inet_addr(servIP);
echoServAddr.sin_port = htons(echoServPort);
```

```
If (sendto(sock, echoString, echoStringLen, 0,(struct sockaddr *)
        &echoServAddr, sizeof(echoServaddr)) != echoStringLen)
        DieWithError ("sendto () sent a different number of bytes than
                        expected");
```

- 에코응답을 받고 출력

1. 메시지 받음

```
fromSize = sizeof(fromAddr);
If ((respStringLen = recvfrom(sock, echoBuffer, ECHOMAX, 0,
        (struct sockaddr *) &fromAddr, &fromSize)) != echoStringLen)
        DiewithError("recvfrom() failed");
```

2. 메시지 발신지 조사

```
If (echoServAddr.sin_addr.s_addr != fromAddr.sin_addr.s_addr)
{
        fprintf(stderr, "Error: received a packet from unknow
                        source.₩n");
        exit (1);
}
```

3. 받은 스트링 인쇄

```
echoBuffer [respStringLen] = '₩0';
Printf ("Received: %s₩n, echoBuffer);
```

UDPEchoServer.c

- 소켓 생성 및 설정

```
echoServAddr.sin_addr.s_addr = htonl(INADDR_ANY);
```

- Echoclient.c

```
echoServAddr.sin_addr.s_addr = inet_addr(servIP);
```

상수의 어떤값이 라도 수용

서버 IP주소 입력

```
for (;;)
{
    cliAddrLen = sizeof(echoCltAddr);

    if ((recvMsgSize = recvfrom(sock, echoBuffer,
            ECHOMAX, 0, (struct sockaddr *)          &echoClntAddr,
    &cliAddrLen)) < 0)
        DieWithError("recvfrom() failed");
    printf("Handling client %s₩n", inet_ntoa(echoClntAddr.sin_addr));

    if (sendto(sock, echoBuffer, recvMsgSize, 0, (struct sockaddr *)
        &echoClntAddr, sizeof(echoClntAddr)) != recvMsgSize)
        DieWithError("sendto() sent a different number of bytes than
                                    expected");
}
```

UDP 소켓을 이용한 송신 및 수신

- TCP와 UDP의 중요한 차이점 : UDP는 메시지 경계를 유지한다
- Client에서 send to()로 송신한 데이터는 server에서 recvfrom()으로 받아서 다시 client로 되돌려 준다. 이것은 client의 주소를 sendto()로 송신하기 때문에 그 주소를 다시 recvfrom()으로 돌려준다. 즉 서로 다른 recvfrom()의 호출은 sendto()로 부터의 호출이 아닌 각각의 sendto()로 부터 데이터가 반환된다
- UDP소켓은 연결을 필요치 않으므로 여러 다른 메세지들이 들어올수 있는데, 이 메세지들은 FIFO(first-in first-out)의 수신버퍼에 저장되기 때문에 각각의 메세지들을 구분할 수 있는 메시지 경계가 필요하다. 이것은 발신지 주소로써 구분할 수 있다.

- TCP와는 다르게 UDP는 오류회복기능이 없다.

-이것은 TCP처럼 버퍼에 자료를 저장하고나서 보내는 것이 아니라 sendto()가 호출된 시점에 메시지 전송을 위해 하부채널로 넘겨져 이미 떠났다는 것을 말한다.

- recvfrom()가 호출이 되었을때, 충분히 큰 버퍼가 잡혀 있지 않다면 그 이상의 데이터들은 수신측에 아무런 통지 없이 사라질 것이다.
- recvfrom()으로부터 반환될 수 있는 최대 데이터량 : 65,507 bytes이것은 하나의 UDP에 의해 운반될 수 있는 가장 큰 용량이다.